Research Reports ESPRIT

Project 813 · TODOS · Vol. 1

Edited in cooperation with
the Commission of the European Communities

B. Pernici C. Rolland (Eds.)

Automatic Tools for Designing Office Information Systems

The TODOS Approach

Springer-Verlag
Berlin Heidelberg New York London
Paris Tokyo Hong Kong Barcelona

Editors' addresses

Barbara Pernici
Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy

Colette Rolland
Université de Paris I, UFR 06
17, rue de la Sorbonne, F-75231 Paris Cedex, France

ESPRIT Project 813 "Automatic Tools for Designing Office Information Systems (TODOS)" belongs to the Subprogramme "Office Systems" of ESPRIT, the European Strategic Programme for Research and Development in Information Technology supported by the European Communities.

The TODOS project develops tools to support office systems design covering all phases from a planning step to the proposal of an architecture of office systems. The tools developed will be used by the system designer and will support the phases of requirements collection and analysis, logical design, rapid prototyping of office systems to validate requirements, and architecture design.

Participating Organisations:
Dornier, D; Italtel, I; Océ, NL; Politecnico di Milano, I; Sema, F; Thom'6, F; IEI-CNR, I; Systems & Management, I; Université de Paris I, F; BIFOA, D.

ISBN-13:978-3-540-53284-2 e-ISBN-13:978-3-642-84323-5
DOI: 10.1007/978-3-642-84323-5

Publication No. EUR 13107 EN of the
Commission of the European Communities,
Scientific and Technical Communication Unit,
Directorate-General Telecommunications, Information Industries and Innovation,
Luxembourg

2145/3140 – 543210 – Printed on acid-free paper

Foreword

The market for information technology products is rapidly changing from a manufacturer-driven market where new products were determined by the evolution of technology, to a user-driven market where users buy only products corresponding exactly to their needs and where competition is very strong.

Confronted with this market situation, hardware and software producers are being obliged to adopt new strategies, and to make a large number of products available on the market in response to a variety of different needs.

As a result of the multiplicity of choice available, the design of an office system which corresponds precisely to user needs is becoming an increasingly complex task.

With exactly this in mind, the Commission, as early as 1985, invited submissions of projects aiming at the development of such adequate tools in its Call for Proposals for the ESPRIT Programme, in order to assist companies in the design of their office systems. This topic was recognised as being of strategic importance, considering the low level of penetration of Information Technology in European enterprises compared to the United States and Japan.

Following this strategy, the project TODOS was selected and launched. This project has successfully developed tools and methods for the definition of the functional specification of the office system, as well as the system architecture and user interface - results which can be of great interest for the IT community at large.

For this reason, the TODOS consortium, in accordance with the Commission of the European Communities, has decided to publish the main results of the project in this book.

We take this occasion to congratulate the consortium's team for the success of the project, as well as the excellent spirit of collaboration that has animated the participants.

Didier Bouis

ESPRIT Project Officer
Commission of the European Communities

Preface

The TODOS methodology and design support environment for office information systems development are presented in this book. In TODOS, a method for the definition of an office system is proposed, and tools support the developer in the different design phases in making design choices, analyzing results, and presenting these design results to final users of the office systems.

The book is mainly oriented to professionals looking for a method for office system development. The TODOS approach provides a framework within which each designer can evaluate and tailor his own method. In addition, a set of design tools is proposed, which can be realized by professional groups interested in developing their own tools, or requested directly to the TODOS team partners who proposed them.

The TODOS team was formed answering a Call for Proposal from the ESPRIT (European Strategic Programme for Research and Development in Information Technologies) Programme of the Commission of European Communities, within the Office Systems Area. The team worked from January 1986 to December 1988, and it proposed an original method for office systems development and a set of tools for the different design phases in the TODOS method.

The main ideas underlying the TODOS approach are the goal of avoiding unnecessary reimplementation of hardware and software components available on the market, and the use of computer based design support tools to achieve this goal. Data and knowledge bases are used in TODOS to store information about available components and about design choices. A prototyping tool is used to present the office system to the users before its actual realization, to obtain users' evaluation. The result of a project developed with the TODOS approach is the definition of the functional specifications of the office system, of the user interfaces, and of an architecture for office hardware and software components.

The book may also be valuable for office system managers interested in exploring possibilities of developing office information systems in their organizations. Moreover, it offers material for study in the academic world: researchers in the field can extend the approach defining new advanced design support tools; students can examine the state of the art in office system development support tools, and study office design support environments.

The TODOS consortium is composed of partners and subcontractors from several European countries: Dornier GmbH (D) acted as the main contractor in the ESPRIT

TODOS project; other partners are the following: Italtel (I), Océ (NL), Politecnico di Milano (I), Sema (F), Thom'6 (F). IEI-CNR (I) and Systems & Management (I) were subcontractors of Italtel, and Université de Paris I (F) of Thom'6. BIFOA joined the Consortium when the project was under way, providing experience from another ESPRIT project, FAOR, in which a methodology for functional analysis of office requirements has been developed, and giving precious integration and evaluation suggestions to improve on-going work.

The book illustrates methods and tools to support requirements collection and analysis, conceptual modeling, rapid prototyping, and architecture design for office systems. The goal of the method is to improve the quality and facilitate the design of office systems, minimizing the development effort. The complete TODOS methodology was validated on an office information system development case study, which is presented in detail in the book.

The TODOS team acknowledges the constant support of the Commission of the European Communities, and in particular of J. Roukens and J. Machnik.

We also thank the reviewers of the TODOS project for their patient work and valuable suggestions, aimed in particular at achieving an integrated approach to OIS design: L. Bhabuta (UK), J.P. De Blasis (F), H. Niessen (D).

Besides the authors of the papers in this book, contributions to ideas presented in the book came from the many persons who participated in some of the TODOS phases. Prof. G. Bracchi from Politecnico di Milano helped to start the project with his invaluable suggestions and to drive it in its initial phases. W. Vogel from Dornier GmbH as a project manager created a supportive and cooperative environment among the partners. Other researchers participated in the project: G. Benci, C. Richard (Thom'6), H. Malherbe, J.J. Roubiere (Sema), C. Antonelli (Systems & Management), M. Aksit, E. Bledoeg (Océ), F. Barbic, R. Maiocchi, S. Pozzi (Politecnico di Milano), M. Habon, W. Kerber, G. Mischke, J. Shaw, W. Scherer (Dornier).

We thank all of them for their cooperation.

Milano and Paris, July 1990

Barbara Pernici - Colette Rolland

Table of Contents

Chapter 1

The TODOS Environment

Barbara Pernici

1 Office Information Systems Design

Management of information and data exchange in the office are based more and more on the use of advanced technologies. The office of the future will provide many computer supported services for data acquisition, information retrieval, message exchange, data and activity distribution. Modern office are already based on the use of one or more of these services, but their effective integration is still an investigation issue. The major problems towards integration are not only technological, but also methodological, since it is necessary to provide criteria for the evaluation of systems being proposed to support office activities.

Methodological issues have been investigated in the design of office information systems. Office Information Systems (OIS) are office systems in which some established procedures to perform office work can be identified. This definition does not imply that all office activities can be classified as a part of one of the office procedures, but rather that many office activities are oriented towards the realization or support of well defined office goals. The office activity as such is well integrated with the overall organization goals. Therefore, the design of Office Information Systems is driven by the goals of the organization and by its structure.

Activities performed in the office have been traditionally classified into two categories (Panko 1984): routine activities and creative activities. Tools provided by modern technology to support these two types of activities may be either distinct or integrated. It is essential to consider both types of activities during office systems design, to achieve good quality of work in the resulting office systems, and flexibility in performing it.

The goals of a office design methodology are manifold. The main aim is that of making the OIS design process easier and more reliable (Bracchi and Pernici 1984). A methodology has also to take into consideration the complexity of the office environment. First of all, it is necessary to define office and business goals, which are not always evident, in order to understand what the work performed in the office is, and how the system to be designed will affect this work. The analysis of office work that must be carried out in order to gain this knowledge is also complex, due to the nature of the office work itself, including routine and creative activities, with a large number of exceptions to established procedures. The need for methodologies is evident in office systems design, since they provide a guidance for analysis and design of complex office systems. The result of their application is in terms of more clearly understood needs of the organization and better justified proposals for technical solutions.

The first goal of an office design methodology is that of obtaining a description of the office. A complete and formal description of all aspects of office work is unfeasible owing to the large number of exceptions, special cases, which are hard to identify, and also due to the difficulty of evaluating the impact of the use of new technologies in the pre-existing work environment. Therefore, to support office description, office design methodologies use models, whose purpose is that of describing as many aspects of the office as possible in an unambiguous way. The descriptions of the office can be performed at different levels of detail and consider multiple views, to capture various aspects of office work. A second goal of a methodology is that of guiding the designer in using office models as a basis for the analysis of the office. The result of analysis, aimed at improving office work, are suggestions to change the organization of work within the office, integrated with technical solutions based on modern office support technology. One of the problems to be considered by a methodological approach is the ability to support the designer in choosing among technical solutions in a continuously changing market, in which new products are added and supersede previous proposals very quickly. A methodology should therefore offer a set of criteria to enable the designer to evaluate possible solutions.

OIS design is strongly affected by the availability of new technologies to support office work. For instance, workstations are becoming more and more widespread in office systems, due to their increasing performances and their decreasing cost. Multimedia data are more and more often handled electronically within office system, replacing previous archiving techniques, such as for instance file cabinets and microfiches. Storage and retrieval of multimedia documents requires sophisticated technical solutions to store documents efficiently and in order to be able to retrieve them according to their contents with acceptable response times. Other types of data which are not manipulated in conventional information systems concern the need for storing information along the temporal dimension. Temporal information is associated to versions of documents, to times and durations of activities, to work plans. Integration with data stored in the organization databases is needed. Another important aspect in office systems is communication. Electronic communication links are established among office workers of the same organization and with the external world. Rapid exchange of information due to electronic communication links modifies the way work is performed in offices. All the above discussed aspects of new technologies must be identified and supported by modern OIS design methodologies.

Several methodological approaches to OIS design have been suggested (Bracchi and Pernici Winter 1984, Ellis and Naffah 1987). A possible classification of these methodologies is based on the design phases they cover (Fig. 1).

A *feasibility analysis* is necessary before starting an office system development project on a wider scale. Once the need for providing technology-based support has been identified, and the organization goals stated for the office system to be developed, the OIS development process can start with a *requirements collection and analysis phase*. The goal of this phase is that of collecting data about the present organization of office work, and to suggest potential areas for improvement, supporting these suggestions with adequate motivations. The *design phase* is concerned with the realization of the potentialities for improvement identified in the previous phase. Design of an OIS covers several different aspects: definition of the functionalities to be provided by the office system, and automation of procedural steps where this is possible; design of the user

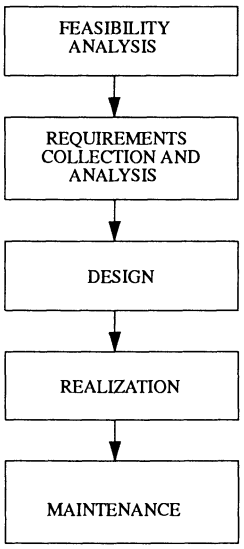


Figure 1: OIS design phases

interfaces to be provided to office workers; design of the architecture of the system, evaluating the different possible technological solutions. *Realization* of an OIS is rarely based on the development of new ad hoc solutions for a specific office. The office system will be constructed on the basis of a set of existing hardware and software components. This aspect characterizes office systems development, in particular if compared to software development in information systems of conventional type. The realization phase, therefore, is based more on integration and personalization of existing hardware and software components, rather than on the development of new components. To this purpose, the design choices proposed in the previous phase have to be formulated in terms of items available on the market. Alternative solutions have to be evaluated during design according to cost and performance requirements defined in the requirements collection and analysis phase. In the case of office systems, *maintenance* is mainly in terms of adjusting the realized system to new requirements due to modified organization goals, and, most importantly, to new available technological proposals. These changes have to be evaluated with the same criteria used in the design phase mentioned above.

A number of methodologies has been proposed since the beginning of research and application of Office Automation, in the early eighties. A classification of available methodologies can be based on the development phases they cover. Early design methodologies, such as OAM (Sirbu et al. 1984) and OFFIS (Konsynski et al. 1982), focus their attention on the initial phases of OIS development. The goal of OAM (Office Analysis Methodology) is the identification of OIS requirements and the definition of the functionalities needed by office units, OFFIS is oriented in particular to the description of the organization of the office and on activities to be supported, thus focusing on later development phases. The common characteristic in these approaches is that they follow a traditional approach to the development life-cycle, with the assumption than the design phase implies a major effort in software development. More recent approaches focus instead mainly on the first phases of OIS development. Two projects

aimed at OIS development methodologies in ESPRIT are FAOR (Schaefer et al. 1988) and OSSAD (Conrath et al. 1988, De Antonellis and Zonta 1990). In FAOR (Functional Analysis of Office Requirements) a methodology has been developed that provides a set of methods tailorable to different office requirements. FAOR covers the phases Feasibility Analysis and Requirements Collection and Analysis. FAOR results have been integrated in the TODOS methodology to cover the initial development phases in OIS design. OSSAD (Office Support Systems Analysis and Design) focuses on organizational problems, tightly connected with technical aspects: the aim is to bridge the gap between the vendors and the user organizations.

TODOS proposes a new approach to OIS design based on two major assumptions:

- the development activity is aimed at integrating and personalizing *existing components*, rather than developing new ones. This approach is distinct from other proposals, which aim principally at defining the functional requirements of OIS. As a consequence, TODOS covers Feasibility Analysis, Requirements Collection and Analysis, and Design. Realization is intended to be only in terms of composing and personalizing available components.

- the development process must be supported by *computer aided design tools.* While the importance of support instruments was recognized also by previous approaches, only limited computer-based assistance was proposed to support the activity of the designer. In TODOS, a toolkit is proposed to cover all phases from requirements analysis and design to the selection of an adequate architecture for supporting office work.

The TODOS proposal is therefore articulated in a set of tools to support design and evaluation of technical solutions.

2 The TODOS Life-Cycle

TODOS proposes a design life-cycle that is particularly oriented towards reuse of existing components and user evaluation of results of the design (Pernici and Vogel 1987). The goal of the TODOS methodology is to define the functional and architectural requirements of office information systems. In TODOS, we envision a design activity performed in strict cooperation by a design team composed by two different types of professional figures:

- *developers.*
  Developers collect and analyze requirements from the target organization and design technical solutions.

- *user representatives.*
  User representatives evaluate technical proposals from the user side. This evaluation is particularly critical in the first development phases, i.e., feasibility analysis and the definition of requirements, and in evaluating user interfaces to the office system.

The TODOS development phases are iterated until the resulting design is accepted by user representatives and considered adequate by developers.

The TODOS development life-cycle is composed of two design loops (Fig. 2.):

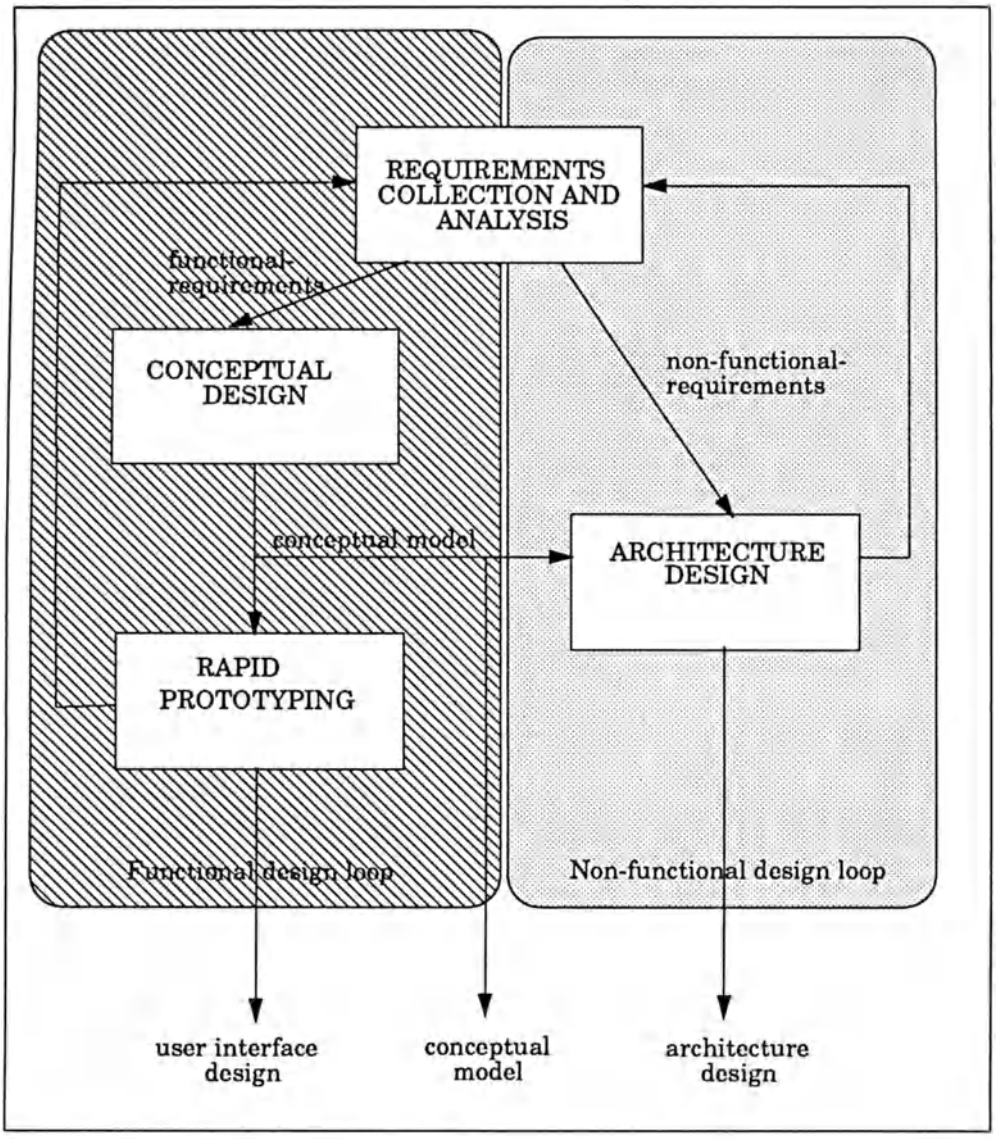


Figure 2: The TODOS life-cycle

- *functional design loop*
  The functional design loop has the goal of defining the functionalities to be provided by the office system. The result of the functional design loop is the definition of an *OIS conceptual schema* and of *interfaces* to the office system, derived from an office *prototype*. The office conceptual schema represents activities performed in the office, in terms of documents to be manipulated, office workers, message exchanged, and actions triggered by events. Interface requirements define, for each office worker, the more appropriate interface to the OIS.

- *non-functional design loop*
  The non-functional design loop defines non-functional requirements, to allow the developer to examine possible alternative architectural choices, and providing criteria to evaluate them, mainly in terms of technical feasibility and performance. The result of the non-functional design loop is the definition of an *OIS architecture.*

As shown in Fig. 2., four design phases are iterated in TODOS:

- *Requirements Collection and Analysis*
- *Conceptual Design*
- *Rapid Prototyping*
- *Architecture Design*

The *Requirements Collection and Analysis* phase focus both on functional and non-functional requirements. Functional requirements include the office organizational structure, description of the office goals and products, tools being used to achieve them and requirements for additional functionalities. Non-functional requirements are mainly in terms of cost and performance parameters, office layout, and include constraints on technical characteristics of the OIS to be realized, such as compatibility with already existing equipment.

The *Conceptual Design* phase is concerned with designing an OIS with the required functionalities. Such functionalities are represented in the OIS conceptual schema.

The *Rapid Prototyping* phase produces a prototype of the OIS to be developed. Such prototype demonstrates the functionalities provided to each office worker, focusing mainly on interfaces.

The *Architecture Design* phase has the goal of proposing an OIS architecture, in terms of hardware and software office components. Such an office architecture must be compatible both with functional requirements expressed in the Conceptual Design phase and with non-functional requirements.

The *functional design loop* includes the phases of functional requirements collection and analysis, conceptual design, and rapid prototyping.

The *non-functional design loop* includes non-functional requirements collection and analysis and architecture design.

The two design loops are strictly connected, since required functionalities have implications on the choice of the architecture, and architectural choices can restrict possible functionalities. The OIS conceptual schema produced by the Conceptual Design phase is the basis for Architecture Design, since it describes required functionalities. On the

other hand, some of the required functionalities may be changed according to available equipment and architecture choices. For instance, requiring advanced image manipulation functionalities subsumes the availability of a graphical interface. As shown in Fig. 2., the OIS conceptual schema is particularly relevant in the TODOS life cycle: it is the link from the functional design cycle to the non-functional design cycle, being the basis for rapid prototyping and interface design and the basis for architecture design.

As discussed in the previous section, TODOS is concerned only with design of functional and architectural characteristics of OIS systems. A system design and development phase is outside the scope of TODOS, since a major assumption is that the OIS is only built out of pre-existing components. The outputs of a project developed with the TODOS methodology indicate which are the hardware and software components available on the market to realize the required functionalities. A further implementation step will consist on the construction of the proposed architecture and personalization of the software components according to the results of Conceptual Design and Rapid Prototyping.

3 The TODOS Development Environment

3.1 TODOS Tools

The principal characteristic of TODOS is to to provide not only a methodology, but also computer assisted design support. TODOS development aids can be classified in three broad categories:

- *Models*

  Each design phase in TODOS is based on a model used to structure relevant information and to construct analysis instruments on top of it. Due to their purpose, models in different phases have different characteristics, but nevertheless they present some common features. First of all, all design phases consider offices *open systems* (Hewitt 1986). In a continuously changing environment, such as Office Information Systems, future extensions should be easily included in design models. Flexibility, tailorability and extensibility are essential characteristics for an OIS model. New organizational and technological approaches can require basic changes to the design approaches being followed at present. To ensure the feasibility of such extensions, each TODOS model is associated with a *meta-level* description of the model itself, thus enabling redefinition of existing concepts and adding new features, without the need to reimplement existing design support tools from start. *Semantic models* provide flexibility characteristics which are particularly indicated to achieve the goals mentioned above. The approach to semantic modeling used in TODOS presents some common aspects in all design phases: generalization hierarchies are a particularly powerful concept derived from semantic modeling (Brodie et al. 1984); other abstraction concepts, such as aggregation and association of entities, and semantic links between entities have also been included in TODOS models. We refer the readers to chapters discussing specific design phases for details about characteristics of models used in a particular development phase.

- *Languages*

  To each model proposed in TODOS a language is associated. The language associated to a model is mainly used internally by each design support tool, while interfaces of developers to tools are designed in order to provide a user-friendly access to tools during OIS development. Some discussion about internal representation languages for the different design phases is presented in the book, and, for further details, the reader is referred to technical reports describing implementation of TODOS tools.

- *Design Repositories*

  Models are used as a basis for describing information needed for analysis and design in the different development phases. The TODOS environment provides a set of integrated design repositories, in which design information is stored. Each development phase has a repository containing several levels of information. *Meta-level* information specifies the characteristics of the model being used; such information is needed due to the extensibility approach discussed above. *Schema* information specifies design information for a particular OIS development project, organizing information around the model described at the meta-level. *Instances* (real office data) can also be inserted, for instance to give examples of an item described at the schema level.
  The above mentioned approach is used in each development phase. Details about the application of this approach to each of the phases are given in the relevant chapters.

- *Analysis Tools*

  Models and repositories are proposed in TODOS to support analysis of design information and formulation of design solutions.
  Tools may either provide generic analysis functionalities, common to all design phases, or be peculiar to a particular design phase. A generic analysis tool, used in all TODOS design phases, is a tool for retrieval of information stored in design repositories. Such a query interface is usually tailored to each design phase in order to support specific queries on repositories needed in that phase. It is important to notice that design repositories are not only used by developers working in the design phase in which information contained in the repository is stored, but also by developers working in other related design phases. For instance, the Specification Database, containing OIS Conceptual Schemas (see Fig. 2.), is not only used during conceptual design, but also during rapid prototyping and architecture design. Since the TODOS phases are iterated, the results of a design phase can also be used by developers in earlier phases in the development life-cycle: for instance, OIS Architectures produced during Architecture Design (see Fig. 2.) can be examined by analysts during iterations of the Requirements Collection and Analysis phase, in order to analyze previously proposed architectural solutions.
  Views on repositories, specific to the needs of developers in a given design phase, are defined. In general, several views are needed during a given design phase. Such views provide different perspectives on the OIS being analyzed. For instance, it is useful to analyze collected requirements both to examine which documents are relevant to a given office unit, and to analyze how a given document flows in the organization. The definitions of views on a given repository are associated with the description of the model, in the repository meta-level.

Other analysis and design tools are specific to different design phases. The characteristics of these tools are described in the relevant chapters of this book. Analysis tools can be used to identify abnormal situations, inconsistencies in collected information, and provide relevant explanations, suggesting appropriate modifications. Design tools provide the developer with a support to construct OIS design proposals.

All TODOS tools are proposed within the methodological framework developed in TODOS. In the following section, an overview of the TODOS is presented. Methodological and computer based design support provided within each phase is briefly illustrated. Each design phase is then presented in the following chapters of the book.

4 TODOS Development Phases

4.1 Requirements Collection and Analysis

The TODOS design methodology starts with the Requirements Collection and Analysis phase. The objective of this design phase is to identify and specify requirements, transforming unformatted information into an organized, although not rigid, structure which is useful for analysis and the preparation of the following design phases.

First of all, the task of this phase is to assess initial feasibility of the OIS with new technologies in an organization (Heijmink et al. 1988). Collection of requirements inside an organization in the scope of an automation process is a cooperative process between employees, their managers, and the team of analysts in charge of the study. Social aspects are very important, so the computer based tools for this phase assist the analyst in his work, but automation can hardly be proposed. Initial feasibility assessment is followed by a further collection phase, in which informal or low formalized information is organized into a structure, both to enable requirements analysis, and to prepare the basis for subsequent design phases.

The process of requirements identification is a complex one. In fact, office data are collected along several dimensions, to be able to provide information about functionalities, activities performed in the office, interface requirements, cost and performance requirements. The methodological guidelines for the early phases of requirements identification and collection are provided by the Activity Framework developed within the FAOR project (Schaefer et al. 1988). Office workers and their activities are represented, volumes and frequencies of document flows memorized, distribution of work and layout of the office collected. To enable to organize collected material, this task is performed at different levels of details.

Requirements Collection and Analysis may be performed in TODOS along different perspectives. A functional perspective indicates *why* given office activities are performed, a procedural perspective *what* is done, and a physical perspective *how* work is executed. Another way to examine and structure data is proposed in relation to design phases: a view collects and analyzes information to perform a feasibility study, while a different level of detail and different types of information are needed to collect data to be used in the following phases of Conceptual Design and Architecture Design. In TODOS, a number of observable entities has been classified within the framework of Requirements Collection and Analysis: *activities* are organized according

to the *product-objectives* they are related to; information about *people* in the office concerns characteristics of *human resources* and the *organization structure*; technological aspects are structured in terms of *office tools* used by office workers.

As discussed in the previous section, it is important to be able to tailor analysis and design tools according to changing technology and to different organization situations. Therefore, an extensible analysis model is proposed for the Requirements Collection and Analysis phase in TODOS (TODOS Analysis Model - TAM). The Interpretative Model of TAM makes it possible to define different perspectives to describe functional characteristics of OIS, in terms of the categories of observables mentioned above. For each perspective (or interpretation), a structure for requirements is specified: the structure defines relations between observables, types of observables, and analysis information. Analysis information is specified in terms of qualitative and quantitative indicators, such as for instance cost and performance parameters, adequacy indicators (e.g., reliability, timeliness, completeness), volumes, and so on. The information modeled according to the Interpretative Model corresponds to meta-level information. Actual information is structured according to the Descriptive Model of TAM, which allows to collect requirements under a given interpretation. The TODOS Analysis Model is oriented principally to collection and analysis of functional requirements. For aspects concerning non-functional requirements, TAM is complemented with the TODOS Performance Model (TPM). In TPM, performance measures, office goals in terms of performances, response time requirements, ergonomic aspects are specified. Also in TPM it is possible to specify meta-level information, e.g., new performance evaluation rules, to tailor the analysis to a given office and to adjust it according to possible changes in the way OIS performance is evaluated.

The results from the application of the functional and performance analysis are in terms of areas of interest for OIS development. *Critical areas* are identified and problems to be solved specified.

After identification of critical areas, information needed by the subsequent design phases is identified and collected, completing previously identified requirements, and storing it within the same frame of reference.

The repository used during Requirements Collection and Analysis is called *Office Data Dictionary* (ODD). Tools for requirements analysis utilizing the contents of the ODD have been developed. Several query interfaces have been defined to support ways of analyzing requirements which were traditionally performed with paper and pencil. Moreover, some analysis tools specific to this phase have been developed. For functional analysis, TAM defines indicators to identify dysfunctions. To analyze non-functional requirements, a tool for performing performance evaluation to identify bottlenecks and critical areas is provided.

4.2 Conceptual Design

The goal of the Conceptual design phase is to design an OIS in terms of the functionalities to be provided, defined in the previous Requirements Collection and Analysis phase. Conceptual Design should be as independent as possible of the characteristics of the architecture of the target system. The result of the Conceptual Design phase is an OIS Conceptual Schema, which is constructed on the basis of the functional requirements collected in Requirements Collection and Analysis phase and stored in the

ODD. The OIS Conceptual Schema is used as an input from two subsequent design phases: the Rapid Prototyping phase helps to validate the OIS Conceptual Schema, the Architecture Design phase has the goal of providing an architecture to support the required functionalities.

In TODOS, a conceptual model has been defined (TODOS Conceptual Model - TCM), supporting the definition of two types of conceptual entities: *static entities* describe structural aspects of the office, such as for instance documents, office workers, messages; *dynamic entities* describe how these entities are interrelated for performing office activities. TCM provides a basic set of concepts (predefined static and dynamic entities), which can be extended according to the needs of specific developments.

OIS Conceptual Schemas are stored in a Specification Database (SDB). The SDB stores both meta-level information about predefined entities and information about entities in the schema of the specific office being designed.

A set of tools is provided to access and analyze the contents of the SDB. A powerful query language is the basis of the query interface (TODOS Query Language - TODQuel). Actual queries are performed using a designer oriented interface, which provides the developer with a set of predefined queries. Such queries allow the designer to retrieve information from the SDB to analyze characteristics of the OIS Conceptual Schema. The principal goal during conceptual design is the derivation of a consistent schema, which is also satisfactory according to some design criteria. Consistency and quality analysis are therefore included in the query interface to the SDB.

Another important requirement of conceptual modeling is the ability to represent the conceptual schema in an understandable form, both for the designer and for the user. A *graphical interface* is particularly appropriate to this purpose. A graphical representation of specifications has proved essential in information systems and software design; most of the Computer Aided Software Engineering (CASE) tools provide a graphical interface based on a particular specification model (Sommerville 1989). Such a need has also been since long recognized for OIS design (Nutt and Ricci 1981). The graphical editor for TCM schemas allows user representatives' evaluation of conceptual schemas.

Other tools developed in TODOS are oriented to the evaluation of particular characteristics of the OIS being designed, such as for instance the communication protocol followed in the office between the OIS and the office workers.

The SDB is a source of information for the following design phases. Its contents can be accessed in bulk, such as in the case of office prototyping, where all information in the SDB has to be processed to produce an office prototype, or using the query interface, as in the case of Architecture Design, where data about functionalities have to be aggregated in several ways, depending on the architectural aspect being considered.

4.3 Rapid Prototyping

The Rapid Prototyping phase has the goal of presenting the user a rapid view on a possible realization of his requirements; working with a prototype, the user has the possibility of better assessing how the required functionalities will be provided in the final office system. The result of this design phase is the realization of a prototype of the office being developed, with the main goal of demonstrating office functionalities. A derived result is the design and evaluation of user interfaces to the office system.

In these terms, we may say that the rapid prototyping phase defines OIS interface requirements. The rapid prototyping phase derives necessary information from the SDB, where functional requirements are stored, and from the ODD, where requirements about office interfaces and document layouts are memorized.

The rapid prototyping tool uses a knowledge base to store both general information about how a prototype is derived from a conceptual schema and specific information about a prototype being created. Generic rapid prototyping knowledge defines how the translation from OIS Conceptual Schemas described according the TCM model to the prototyping tool knowledge base has to be performed; basic activities performed in an office, such as printing, editing, and so on, have predefined implementations in the rapid prototyping tool (office primitives), which can be used to set up the appropriate user environments. The main attention in the prototyping tool is towards the definition of user interfaces. An interface generator is provided, to map from TCM static entities to user interfaces in the prototype system. Some support is also provided to enable the user to evaluate sequences of activities performed in the OIS. Sequences of activities may be triggered in the prototype, if necessary with user intervention, according to the specifications of the TCM dynamic submodel.

4.4 Architecture Design

Architecture Design is a complex activity. It is necessary to consider the present state of technology, to make use of hardware and software components available of the market; moreover, it is necessary to evaluate different architecture proposals according to different criteria, among which cost and performance are the most important. Architecture Design is disregarded in the principal OIS design methodologies, while it is given in TODOS a particular attention. The result of the Architecture Design is a proposal for an office architecture, in terms of components available on the market. The proposed architecture is guaranteed to be realizable, and performance analysis is provided.

Architecture Design is performed in steps: first, a manual method is proposed to map from an OIS Conceptual Schema and non-functional requirements to a possible architecture. The Architecture GEneration Methodology (AGEM) provides the designer with a series of steps for deriving a set of office architectures for the OIS being developed. The steps guide the designer in identifying, in succession, individual office workers and their action, needed functionalities, service access points, subsystems and networks, their internal structure, and finally office architectures. The approach is based on a top down approach, in which, from general requirements to perform office activities, a more and more detailed architecture is derived. Each derivation allows the designer to propose alternative choices, to refine previous choices, and to discard inconsistent architectures.

The main assumption in developing office architectures is that they are based on a set of workstations networked together, principally through Local Area Networks (LAN). Components for the architecture are chosen using an Architecture Specification System (ASPES). This system supports the last step of the AGEM methodology, where specific hardware and software components are selected. The ASPES tool is based on a knowledge base of office components. The individual characteristics of components are specified, together with their interface constraints. Architectures are described in ASPES using an Architecture Specification Language (ASL). Using the ASPES tool

in generating an architecture, the developer is sure that the selected architecture is technically realizable.

The last step of Architecture Design consists in the evaluation of the performances of each of the alternative architectures proposed. The evaluation is performed associating performance measures to each component, derived either from the ODD or from the components knowledge base, or directly entered by the designer. The evaluation of performances is done with a Performance Evaluation (PE) tool, based on a queuing network analysis package, which evaluates workloads, response times, and utilization of resources. Using the results of simulations, the designer can choose among alternative solutions, or modify proposed solutions to solve identified problems. Required modifications may be fed back to the non-functional Requirements Collection and Analysis phase, and be further evaluated. Such modifications may also have consequences on the types of functionalities provided to office workers, and therefore affect the functional design loop, as well as non-functional design.

5 Structure of the Book

Each chapter of the book is devoted to illustrate the TODOS methodology and design support tools for one design phase.

Chapters 2-4 are devoted to the functional design loop. In Chapter 2, the Requirements Collection and Analysis phase is presented; the method for requirements collection and the analysis support tools based on the Office Data Dictionary are illustrated. In Chapter 3, conceptual modeling of office systems is discussed, showing tools associated with interaction with the Specification Database. Chapter 4 illustrates rapid prototyping tools, focusing on definition of office user interfaces.

The non-functional design loop is presented in Chapters 2 and 5. In Chapter 2, non-functional requirements collection and analysis with the TODOS Performance Model is illustrated. In Chapter 5, Architecture Design is discussed in detail: the first part presents the TODOS Mapping Methodology, from requirements and functional specifications to an architecture proposal; the second part illustrates how architecture components are selected with the Architecture Specification System; finally, the performance modeling phase is presented.

In Chapter 6, a case study is presented. The case study shows an example of application of the TODOS method to the design of an office system for a large regional bank, and in particular for offices in the bank supporting the proposal and preparation of loan contracts to large clients. Such application requires good message exchange facilities, interaction with external archives, tools for decision support. In all the chapters of the book, examples largely refer to material presented in the case study. Results from different design phases, interrelation between phases, and iterations of the TODOS design loops are illustrated in Chapter 6.

Finally, in Chapter 7, suggestions for future work on computer based office development support tools are discussed.

Chapter 2

Requirement Collection and Analysis

Gianluca Bassanini, Fabio Di Stefano, Pascal Henry, Giancarlo Lunghi, Edda Pulst, and Gerd Wolfram

1 Requirements analysis for the design of office information systems (OIS): State of the art

Requirements determine the fundamental characteristics of an office information system. They are an informal step to systems specification and design as they focus on features and characteristics of the technical system that will be developed (Hirscheim and Schaefer 1987). The following chapter therefore will outline both the characteristics of requirements and the requirements analysis in order to show the necessity of methodological and automatic support of requirements analysis.

1.1 Characteristics of requirements

Requirements are the result of a dialectical process where an analyst, in consultation and in concert with the user(s), decides which office aspects ought to be changed and which ought to be preserved. The decision is made by studying the existing office situation and considering necessary and/or desirable characteristics of the future office information system.

Requirements and requirements analysis play a key role in the office information system life cycle: "Defining user requirements accurately and completely is probably the most essential task in the whole systems project, and one most critical to its success." (Jeffery and Lawrence 1984, Page 125). Its output on the one hand, constitutes the results of the analysis phase where the actual office situation and problems as well as desired changes are investigated. On the other hand, the requirements document what the designer has to know in order to build the system. Requirements are thus a kind of abstract specification of what the future system must be able to do.

To focus on the requirements for office information systems does not imply that only technological aspects are specified. Because of the socio-technical nature of the office, requirements must also address the social and organizational environment in which the office information system functions. A very narrow view on technology which ignores its environment can cause serious deficiencies in the design: firstly, the system may not be appropriate for the task in hand for example by imposing rigid standards and procedures which do not fit the task and thus creating extra work or reducing flexibility to provide the required service; secondly, the system may make work more unpleasant and/or less satisfying for the users than previously by removing areas of discretion,

	Technical	Economic	Organizational	Social
Strategic	Be IBM compatible Service level should be better than 99.5 %	Invest in IT only when it is economically justifiable	Responsibility for IT should be delegated to lowest level possible	System should support cooperation in teams
Tactical	Network installation must support future upgrade to ISDN	Systems proposals cost-benefit evaluations should meet strict requirement	Systems should provide for one user expert for app. 10 users	Systems should observe such ergonomic standards that 4 hours work will not pose any risks
Operational	Access rights should be definable on the basis of individual documents	that pay back period for any IT project should be less than 4 years	Daily back-up of changed documents	Detatchable key-boards No glare from screens

Figure 1: Examples for requirements on different levels

variety and interest. This, in turn, may decrease user's motivation to produce work of the required quality, or, more importantly, to make the system a success.

Consequently requirements have to take into account the following aspects:

- technical aspects, i.e. compatibility with already implemented systems, processor speed, amount of storage,
- organizational aspects, i.e. the purpose of the office information system and its major application area in the specific organizational setting,
- functionality aspects, i.e. the essential or desired office information system functions expressed in terms of information object form and handling capabilities the office information system must have in order to support the business operations,
- social aspects, i.e. what the users want or agree upon,
- economic aspects, i.e. a fixed budget for the workplace.

All those aspects form the necessary comprehensive view of requirements. Besides this the requirements can be considered on different relevance levels: strategic, tactical and operational levels (Fig. 1).

These differentiation of requirements captures the fact that there are very general or fundamental requirements, such as basic technical standards, which in some organizations are decided upon at the general policy-making level, while others are formulated at the departmental or office level and focus on a particular task in the office. Requirements are of a very complex nature and consequently their specification is a complex process encompassing a number of problems the people involved have to tackle with.

1.2 Complexity of the requirements analysis

Requirements analysis comprises the identification and specification of requirements.

Its complexity due to the presence of human and technological factors can be reduced by employing models of abstraction and computer based tools. The cooperative process

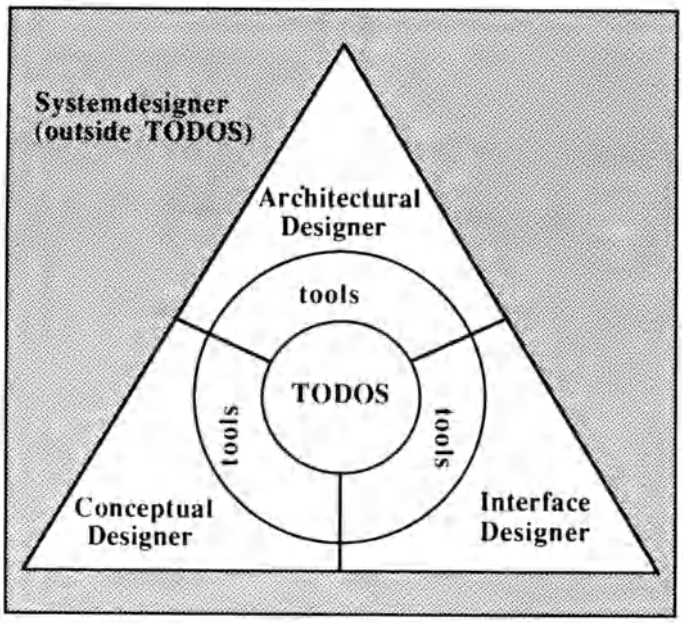


Figure 2: Types of designers within TODOS

encompasses an information exchange between the studied organization, the people involved and the analyst conducting interviews, selecting people and setting the time and sequence for interviews as well as the relevant instruments.

The resulting model should always be as independent as possible from the implementation details in order to represent an effective communication tool between users and designers being as invariant as possible from the technology. A complete specification is reached by stating why an office information system needs to be introduced and what functions of the office information system are necessary for achieving the intended support. Up to now these instruments are mainly used in a paper version.

Requirements collection in analysis is part of the four design phases of TODOS which have been outlined in Chapter 1. with focus both on functional and non-functional requirements for the different design tasks (Fig. 2):

The conceptual designer is concerned with the required functionalities of an OIS. Central concepts are the representation of specific roles of office workers and of communication among office workers within their business goals. The architectural designer has the goal of proposing an OIS architecture in terms of hardware and software components. This architecture must be compatible both with functional requirements expressed in the conceptual design phase and with non-functional requirements like technical feasi-

bility and performance. The interface designer produces a prototype of the OIS to be developed. A prototype system is constructed to illustrate the feasibility of new ideas or design (Kieback 1987).

1.3 Methodological support of the requirements analysis

Diverse methodologies offer support for executing a requirements analysis. The following provides a brief review of the major office information system requirements methodologies (Schaefer et al. 1988, p. 24). Two groups of methodologies can be distinguished in principle:

- **Frame-oriented methodologies**: They concentrate specifically on how to represent the object system in terms of a model. Methodologies considering this aspect are: ICN (Information Control Nets; (Ellis 1979)), SCOOP (System for Computerization of Office Processing; (Zisman 1982), FFM (Form Flow Model; (Tsichritzis 1980), OMEGA (Barstow 1985) and OFFIS (Konsynski et al. 1982).

- Principles of operation-oriented methodologies are focusing instead on the principles one should use in eliciting office information system requirements. Examples of such kind of methodologies are: ISAC (Information and Office Systems Specification; (Lundeberg et al. 1979)), ETHICS (Mumford 1983), OAM (Office Analysis Methodology; (Sirbu et al. 1984)).

There are also methodologies which can be located in between these two orientations. An example is the FAOR approach (Schaefer et al. 1988).

Further methodologies try to capture requirements from the view of the designer. They mainly focus on the future system chosen by the system designer employing models like information flows, activities and data stores as the foundation for evaluating requirements. Some of the methodologies focus specifically on the information objects. In summary, with respect to the different methodologies, it can be concluded the following: They considerably adopt different perceptions on the role of requirements and specification.

Mostly common to this methodologies is the fact that they don't integrate a suitable language or a graphical representation for specifying requirements which supports the achievement of the objectives connected with the subsequent systems specification and design processes. The requirements specification therefore is connected both to the analysis of offices and the design of an office information systems. On the one hand, knowledge about both parts of the life cycle is necessary for the requirements specification while, on the other hand, the specified requirements are a prerequisite for the later design. Requirements also reflect back onto the analysis by allowing to check the completeness of the analysis or its consistency (Fig. 3).

2 Requirements collection and analysis in TODOS

The main idea of the requirements analysis in TODOS is the orientation towards both application and design needs for the later system taking into account the functional and non-functional criteria in a structured way. The TODOS tool for requirements collection and analysis has been developed to help the analyst in his work. Main characteristics of

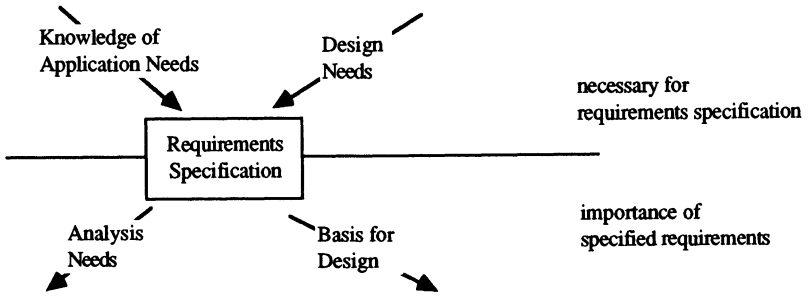


Figure 3: The focus of requirements specification

the work is to transform unformatted or low formalized information in an organized but not rigid structure useful for analysis and the preparation of design phases by expressing property requirements for the future office information system.

The provided structure bases on a data dictionary concept called TODOS ODD (Office Data Dictionary). It contains entities of the office encompassing static and dynamic aspects, human and technical dimensions, qualitative and quantitative data, geographical and hierarchical positions of offices. The provided structure can be easily adapted to every organization in analysis.

In the following chapters the ODD will be presented. This starts with the elaboration of the underlying office models and the description of the phases and instruments of the requirements analysis process. This will be followed by the ODD implementation and an outlook on how the ODD supports the analyst within the requirements analysis process.

2.1 Overview of the TODOS Analysis Model

An Analysis Model should both identify dysfunctions or problems and support feasibility studies which give hints upon aspects and constraints of the future office information system. The collection of data should help the analyst in obtaining the information needed in the later "Conceptual Design" phase for the conceptual model of the office and the architectural modeling of the required solution.

Therefore the tool should guide the selection of those areas in which the other phases of the design will be applied. This model is the **TODOS Analysis Model (TAM)**, which frames the requirements analysis phase inside a reference plan. TAM also provides quantitative parameters synthesizing the overall operation of the "office machine". Naturally the kinds of measurements which will be processed by the tool are the ones required to describe the work flow from a requirements analysis point of view.

It is divided into three views performing the office analysis according to three different perspectives: "Feasibility Study" view, "Functional Analysis" view, "Architectural Modeling" view. On the basis of this scheme, a prototype of an automatic tool for carrying out the Requirements Analysis was developed. The main feature of TAM prototype is to allow the collection of data and information describing the office modeled using TAM by means of suitable forms. A form enables the analyst to insert, retrieve, delete and modify entities describing aspects of the office environments according to TAM.

The results of testing TAM prototype (Bassanini et al. 1988) showed the utility of an automatic tool aiding the Requirements Analysis of an OIS and pointing out critical areas to be later analyzed during a phase of Functional Analysis:

- Goals to be pursued by the new OIS
- Results expected from the phase of Requirements Analysis
- Industrial sector which the analyzed office belongs to (e. g. banking, health care, manufacturing, insurance, and so on).

The TODOS Analysis Model is further divided into two distinct models being strictly connected from a logical point of view:

- The **interpretative model** providing a key to the interpretation of the working activity, a logical criterion to be used for the identification of the essence of office work showing in different perspectives the working environment.
- The **descriptive model** pointing out all the functional information of the interpretative model. This information is needed in order to be able to identify the critical areas requiring the introduction of an OIS.

2.1.1 Interpretative Model

On the basis of what is suggested by the analogy with the physical sciences an interpretative model of office reality has the priority on defining one of its descriptive models. Let us think, as examples of interpretative models in the physical sciences sphere, to the corpuscular and the undulatory models of light propagation. Choosing the first model advantages a descriptive model of reality founded on observables which are typical of rational mechanics (mass, quantity of motion, kinetic energy, cross section and so on). Choosing the second model advantages a descriptive model founded on observables which are typical of the electromagnetic wave (wavelength, frequency and so on). On the same basis the term **"interpretative model"** is - from a conceptual point of view - synonymous with the term "interpretative hypothesis". An interpretative model (hypothesis), indeed, keeps its effectiveness until the descriptive model it inspires is able to describe all the observable phenomena and to furnish data that do not show false the hypothesis itself.

Trying to give a definition of office, a wide number of various aspects must be considered: the place, the set of activities performed, the resources required for performing an operation, the personnel's satisfaction, the control procedures, the overall organization, and so on. At least two different levels of description can be distinguished (Ciborra 1984): a macro level, where the office is seen as the set of activities needed to coordinate the true productive activities; a micro level where the office is seen as a behavioral setting where several activities are carried out by cooperating people, characterized by their own cultures, interest, knowledges. Just examining this second point of view, three different approaches to the office work analysis can be pointed out (Ciborra 1984). These families of models show as many possible strategies in the support of office procedures, on the basis of the different analyzed degrees of complexity in the office work.

The variables characterizing office reality allow a large spectrum of possibilities to define interpretative models. Their name can be derived from the relative importance

of a particular observable in respect of the other ones. Inside the office at least four sets of observables can be pointed out (Leavitt 1964): people, the tools they use to carry out their office work, the tasks they are supposed to accomplish, the organizational structure fixing the procedures of communication and the sharing of responsibilities among the human resources. These sets of variables are rather generic and can be easily divided into subsets: "people" can be meant as professional curricula, natural attitudes, personal relationships with other employees; "tools" stands for traditional manual instruments, computer-based tools, paper forms (or document); "tasks" is for elementary activities, operational procedures, communications between the office workers, and so on.

Generally speaking, the simplest kinds of models for requirements' analysis are those which attach a relevant importance to only one of the possible subset of observables:

- **tool-based models**: the main emphasis is put on the manual execution of the office activities: all the work flowing inside the working environment is modeled as a series of elementary operation whose execution can be optimized by means of suitable instruments (Taylor 1911);
- **document-based models**: the relevant elements of this family of models are the data and the documents processed inside the office (Zloof 1982);
- **procedure-based models**: the office work is seen as a continuous flow of work which should be controlled by an office worker in order to correct disturbs in the shortest response time (Zisman 1978);
- **speech-based models**: the office work is analyzed from the point of view of the conversations developing among the people lining inside the office environment (Flores and Ludlow 1981).

The most general and complex ones underline the mutual relationships between two or more observables: coherently with what is shown by FAOR (Schaefer et al. 1988) the TODOS approach recognizes that office reality is a highly complex system which badly fits for rigid representative schemes. Therefore the TODOS interpretative model matches fairly well with the FAOR approach of individuating a Generic Office Frame of Reference GOFOR (Schaefer et al. 1988): GOFOR employs the concept of perspective for specifying the relevant aspects of the office domain. The perspectives are means for coping with the complexity of offices by separating domains of office functioning, one at a time, and for bringing them together as building blocks for constructing a model of the given office. Within this sphere it is possible - according to the analysis objectives and perspectives - to define a "base of observables" necessary and sufficient for describing the considered phenomena and interpreting them through suitable "filters".

The **goal** to be achieved by an OIS design and implementation is the improvement of the productivity of office work. By productivity it is not merely meant the measure of the units of product per employed resource, but, in more general terms, the suitability of the office performances to the demands coming from the external organizational environment. Therefore, the more the outputs of the office being analyzed will be convenient for users' needs and expectations, the higher the productivity will be. By users it must be meant both offices belonging to the same corporation and external organizations. The office is seen as an environment where a group of people using suitable tools carries out a set of coordinate activities. Office workers do not conform

to pre-established rules and rigid procedures, like e.g. IF situation "s1" THEN action "a1". On the contrary they are responsible for attaining fixed goals and they will be appraised on the basis of the achieved results. In this ideal representation the human resources are managed by objectives and not by the way they adhere to organizational rules. This kind of control system offers a series of advantages. In fact:

- It does not require a well known and stable reality (the more turbulent reality becomes, the more it turns to be difficult to define precise procedures).
- It presupposes a high degree of acceptance and sharing of business missions and constraints; in this way the identification of the office worker with the company to which he belongs will be encouraged.

The office, congruously to this ideal kind of management and control, is seen as an organizational unit providing services to other offices of the same company or to external organizations (**reference scheme**). Theoretically the provided services, should not suit the market standards but the users should address their requests to the external environment in order to be able to satisfy their needs. This marketing oriented approach gives the following benefits:

- an easier measurement of the performances provided by the office;
- the possibility to take into consideration also the users' requirements, that is, the results of the office working activity will be brought up in discussion again if they would not turn out to match with users' needs.

A **filter** characterizing the interpretative model enables the analyst to cluster information functional to the selected approach, according to the goals to be reached by the design of a new OIS. The clustering operation is needed to easily characterize reality by means of quantitative indices. Only using these parameters it will be possible to develop a computer based tool helping the analyst during the requirements analysis.

The conceptual basis used by TAM is called **product-objective (p-o)**, developed in order to achieve the goals of the analysis. A p-o is a product that the office being analyzed must supply in form of a service, document, information, etc. with sufficient quality, in order to effectively satisfy the business goals fixed by the organization to which it belongs.

The product-objective is not a merely physical object such as a form to be filled in or a report to be produced, but it is an objective to be reached to fulfill the goals assigned to the office without regarding the media which is used to deliver the p-o to the user (in this sense also a decisional process can be regarded as a p-o). The selected approach will aim at evaluating the role played by the office being examined inside the company. The outputs of the activities carried out in order to come up to the business expectations should be as suitable as possible to the users' needs complying with the constraints to which the office is subject. These outputs are the p-o's. As it will be shown afterwards, the p-o's generated by the office will be specified through a process aiming at aggregating the activities carried out by the employees. The p-o concept has been developed because:

- it allows to carry out meaningful measurements of the performances of the office, to be meant both as adequacy of the contributions given by the office to the operation of the firm and as a degree of exploitation of the resources;

- it lets to effectively synthesize collected information, looking as a whole to activities, tools, human resources cooperating in generating the individual p-o;
- it focuses the mission of the office being analyzed, the very reasons of existence of the office inside the organization to which it belongs;
- it allows to redesign the modality of carrying out work, in order not to execute more quickly the same activities, but to attain the same goals in a more efficient and effective manner.

The term product-objective has been intentionally chosen since the analysis process will aim at verifying the congruency of the products of the working activity to the business needs: in fact the objective of the office under examination is to satisfy these requirements. So it becomes possible to involve in the analysis also the receivers, the users of work carried out by the office being analyzed.

In the previous paragraph it has been stressed that productivity is "the suitability of the office performances to the demands coming from the external organizational environment." Therefore quantitative parameters are needed in order to get a description of the office environment that can be easily processed by a computer based tool. These indexes are called "critical indicators" inside TAM, and they are calculated starting from the information collected during the analysis of the office environment. They allow to evaluate the performance of the office, considering it from a functional point of view, congruously to the chosen approach to office work interpretation.

Only if criteria of measurement can be defined it will then be possible to carry out a diagnosis allowing to specify critical areas indicating the introduction of new technologies. The most important critical indicators that will be adopted during the analysis are the following:

- **cost of p-o.** It includes all the costs ascribable to the production of p-o. For instance it concerns cost of human resources, cost of used tools, etc. Considering a p-o characterized by a manufacturing cost of $ 1 and a second one costing $ 100, the efforts of analysis will be concentrated on the second one, on obvious grounds of expediency of the study. In fact it is sufficient to gain a small percentage cut in the cost of the second product to obtain savings higher than the whole cost of the first one.
- **value of p-o.** The value of p-o indicates the importance attached by the organization to p-o provided by the office. A method enabling the analyst to classify the order of importance of information inputs entering an office consists in relating these inputs to the goals fixed by the company. The more the requirement of an improvement in the qualitative standard of a product is felt, the more a further and more detailed step of analysis is needed. Once again it should be clear how it is important, in this kind of approach, to keep in mind users' point of view.
- **productive topology of p-o.** If the productive cycle of a p-o involves several offices, the chance that the communication channel connecting the resources becomes a critical aspect increases. Therefore it will be necessary to evaluate the possibility of reducing the number of offices involved in p-o production, the rationalization of information flow and the suitability of a proper communication network.

- **degree of technological covering of p-o.** The instruments used by office workers could be divided in three categories: manual tools (M), Office Automation tools (OA), EDP tools (EDP) (another classification is presently being considered: M, OA, EDP, CE - Communication Equipment such as telephone and telex, SE - Special Equipment, such as copier and color slide projector). For every product the Measurement Covering Degree (MCD) will be defined by a term of numbers representing the percentages of tools of different technologies used during the generation of p-o. For instance the product "invoicing" could be characterized by the following MCD: 40 % M, 30 % OA, 30 % EDP. Then as Functional Quality Degree (FQD) it will be defined as the estimate of quality of the support provided by tools: for instance a p-o belonging to OA class could have a FQD-1 if it meets users' needs completely, FQD-0.8 if some changes in its functional features are desired, and so on. Lastly, the Real Covering Degree (RCD) will be defined as the product of the two previous indexes. The terms of numbers measuring the covering degree show the analysts the timeliness of a deeper analysis aiming at redesigning the technologies assisting p-o production. The analysis could point out the need to introduce automatic tools unknown inside the working environment under examination; to update, or to replace the existing ones; to redefine the percentages of covering (for instance the analysis process could emphasize the need of executing a corporate EDP procedure in a local environment).

- **volumes of production of p-o.** It is the number of p-o's of the same type generated per unit of time. A high value of this indicator suggests the need of a deeper study of p-o productive process, in order to examine how efficiency gains could be made possible.

- **adequacy of p-o.** Let us estimate a p-o on the ground of its reliability, completeness, timeliness. The evaluation of these characteristics made by the office generating p-o being analyzed might not be coincident with the one made by the office which will use the product. Different points of view show the need of a better tuning between producer and consumer, that is the requirement to carry out an analysis aiming to designing a product characterized by qualitative standards requested by the users.

2.1.2 Descriptive Model

The descriptive model congruent to the interpretative model is identical the one developed by Leavitt (Leavitt 1965). The main hypothesis of this approach to the introduction of changes in an organization already existing and working is that an effective change can be carried out only if several organizational variables are at the same time analyzed and modified. Referring to the system theory of organizations, Leavitt's model points out a set of interacting variables which allow to consider the working environment from several perspectives (Fig. 4): "first of all it must be pointed out the need to envisage organization as multiple variables systems. At least four kinds of variables seem of special importance: variables concerning tasks, structure, technology and actors".

"If we want to give a rough definition of these variables, we will say that task signifies the raisons d'etre of the organization, and it includes a lot of different subtasks, meaningful if considered from an organizational point of view. By agents or actors I

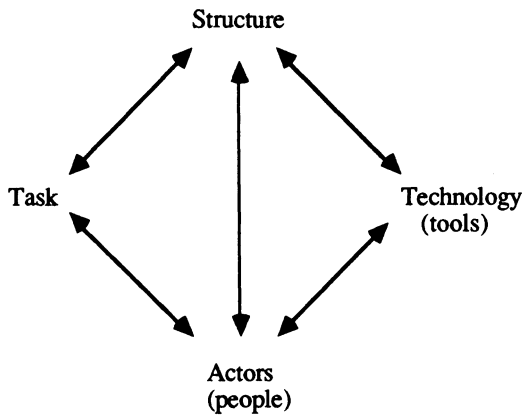


Figure 4: The Leavitt model (Leavitt 1965)

mean especially people, but stating precisely that the acts usually carried out by people must not necessarily pertain to this variable, but they concern also the other variables. By technology I mean the instruments and the equipment, as well as the inventions and the techniques allowing to solve problems, such as, for example, the electronic calculators or the measurements of work. Lastly, by structure I mean the systems of communication, systems of authority (sharing of roles), systems of work flow. These four variables are strongly interdependent: a changing in any one of them will most likely produce a changing in the other ones due to adjustment or counter-reaction". For instance "the introduction of new technological tools can give raise to changing in the structure (the system of communication or the decision system could be affected), in people (as far as number or workers, skills, attitude and activities are concerned), in the accomplishment and even in the definition of new tasks going to be carried out for the first time. Most of the efforts aiming at producing changes, regardless of their original startpoint - that is they could be addressed to people, technology, structure - will have to very soon involve also the other variables. More than once it was noticed that a change in only one of these variables caused alternations in all the other ones. These modifications led to unexpected and often expansive directions" (Leavitt 1964).

The variables pointed out by Leavitt are translated in the following way into the TODOS descriptive model:

- The concept of task has been replaced by the concept of product-objective (to be generated in order to satisfy customer offices' needs). On the contrary subtasks have been renamed activities carried out in order to accomplish the tasks assigned to the office.

- The actors or agents are coincident with human resources, their actions are linked to the other variables being analyzed. Inside the descriptive model relationships

will be established between resources and activities, resources and instruments, etc. taking into account that human resources are the main informative source concerning the office work and the environment.

- The concept of technology allows to consider tools, equipment, forms to be filled, archives, communication network.

- Structure is coincident with organization, seen as organizational charts, goals to be reached, decision systems, list of duties, etc.

It must be clearly stressed that these variables represent different ways of viewing the office work, studying it from different perspectives. These points of view, naturally concur to define an overall picture of the environment being analyzed. But, once again, it must be emphasized that the variables taken into consideration by Leavitt, and the links binding them, must not be intended as entities and relationships of a formal Entity-Relationship (E-R) model. Till now, therefore, we have justified the choice of the p-o concept as the interpretative criterion of the office work. Subsequently the theses of Leavitt have allowed to adopt as descriptive model the group of variables and links previously described.

On the basis of these considerations inside the office we will collect information concerning the following variables: Organization, Tools, Human Resources, Activities, Product-objectives (Fig.5). The data and information which are analyzed within the TODOS Analysis Model are identified and gathered in the early phases of the requirements analysis process. The FAOR approach has been used as a basis to support the analyst in this task. The different activities and actions of this process are therefore guided by the FAOR Activity Framework (Schaefer et al. 1988). This Activity Framework provides the basis for executing a study in a particular client organization. It provides guidelines on how to structure an investigation in order to elaborate and interrelate partial results while defining typical baselines of a study and including aspects of project organization and management.

It should however be stated that at the beginning of the analysis the objectives to be attained should be clearly known whereas it can be ignored how these objectives are reached. Lastly the analysis could point out that some products are useless, or that they can be included inside another product, or that complete p-o's should be redesigned. It is necessary to compare the characteristics of the different p-o's with the requirements of the users (both to be inserted via the Analysis Model): from this comparison information can originate that, requiring a redistribution of the activities among the resources, will heavily affect the whole organization of the office. P-o's therefore are objects that are dynamically defined during the development of the analysis. Nevertheless it should be presupposed that at the moment of the re-design of the office they will be defined in a certain way.

In TAM all the information needed during the phase of Requirement Analysis of an OIS design is described. This information is structured using the Entity-Relationship model (Bassanini et al. 1987).

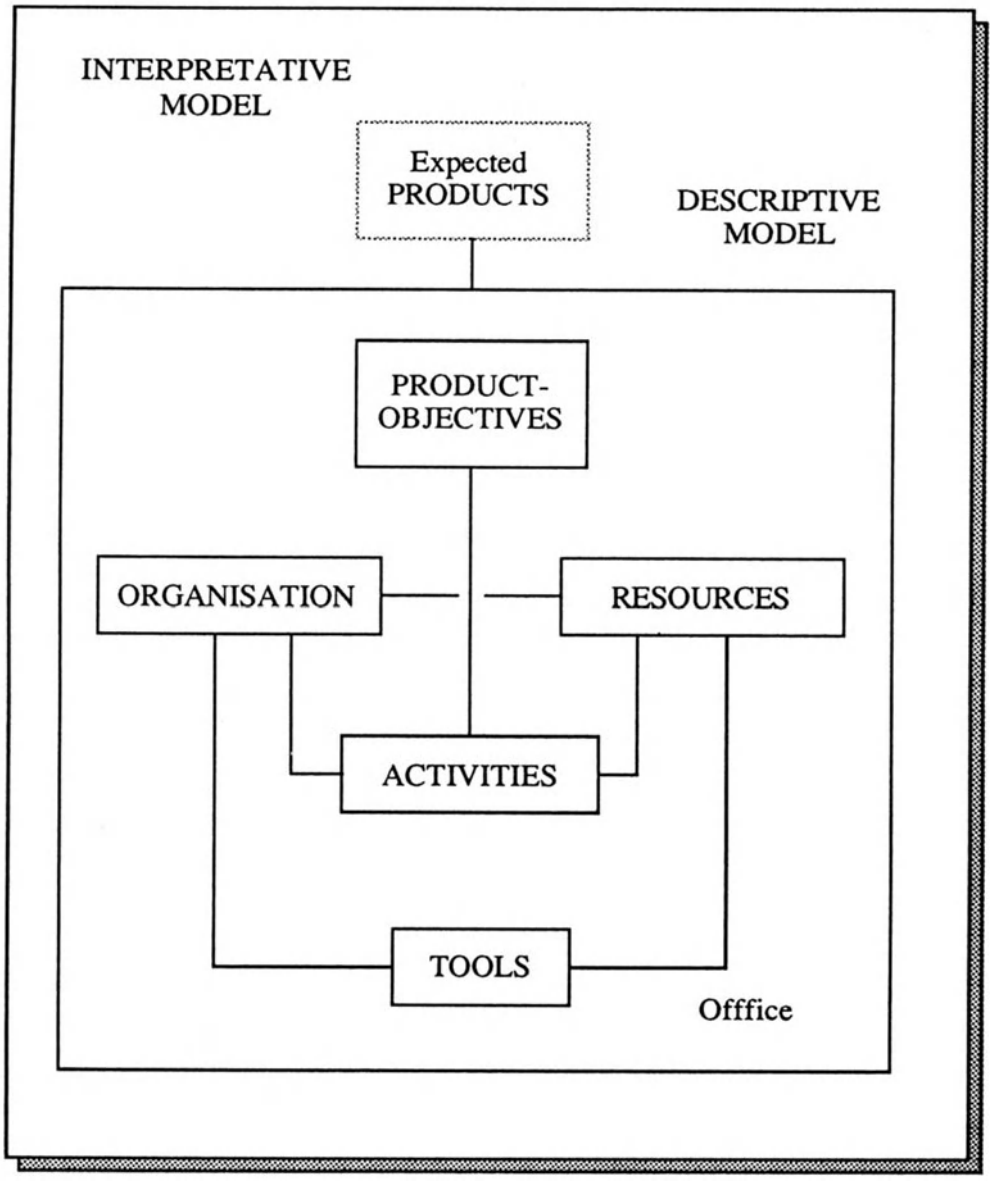


Figure 5: Variables for the TODOS Analysis Model

2.2 TODOS Performance Model: Evaluation of requirements and connection to later design phases

During the analysis, especially with respect to interviews of users, the analyst will get requirements about the future system but without any control of consistency between them. Such level of requirements may be better called "expectations" because it inventories all the expectations of the users about a possible (and not very well defined) future system. At the end of feasibility study, the results and the following debates inside the organization will allow to confirm, update or redefine the initial guidelines. The resulting new version of guidelines will trigger the definition of a set of requirements both functional (modeled with the descriptive model) and non-functional ones.

The TODOS Performance Model (TPM) therefore provides important support in two ways:

- Definition of each elementary requirement
- Explicit and preliminary definitions of both measuring methods and judgment rules
- Computation of judgment rules for a set of solutions

As indicated hereabove, definition of requirements is a progressive process, so the TPM will handle a status for each performance requirement allowing to update the status according to analyst decision and to query requirements considering their status. Main sorts of status are guideline, constraint and expectation.

Level classification is used to have a scale of the scope of a requirement in order to ease arbitration between conflicting requirements during the consolidation operation. Different lists of levels can be defined at will. In a first approach, main levels are strategic, tactic and operational.

Domains are used as a way of classing requirements according to the people in charge of specifying and evaluating them, being of economic, social, technical and organizational nature.

Measurement rules allow individual evaluation of each solution and for each individual performance requirement.

Judgment rules allow to compare between the measured solution for a given performance requirement and to allocate a mark for it. Parameter of judgment rule is the **weight** of the requirement. "Mandatory" could be a kind of weight.

A list of predefined rules are defined with the TPM. During requirements definition, the analyst can choose rules for each individual requirement and fix relevant parameters.

To be able to measure and judge solutions, a prerequisite function is the capability of storing architectural solutions (designed with Architecture Design-tools) in the requirements data base. The storage should fit with the structure of requirements data. So, only a subset all the data of Architecture Design is needed like the **name of a solution**, the **solution performance** for each requirement, the **query** and **deletion** of a solution.

Typical reasons of failures of office information systems is the lack of clear definition of evaluation criteria for the correspondence between the implemented system and the previously defined requirements. The TPM provides a support for these activities by implementing a multi-criterion analysis capability. The rules of analysis should be

fixed simultaneously with requirements definitions during the investigation of an organization. The evaluation report offers the automatic computation of the marks of the different solutions stored in the data base, every defined requirement being processed separately. For other requirements, assumed requirements weights are applied to the marks. Ordering of requirements could be done at will in a report parametrization dialog.

A key point of the "requirements analysis" phase is the definite distinction between the analysis of the existing organization and the progressive definition of a consistent set of requirements to be further used by the other TODOS design tools. Nevertheless it should be stated that the analysis and requirement definition processes are recursive: they are often interleaved, the study of the present situation being the source of ideas for possible future systems and organizations. Both analysis and requirements are based on the same modeling tool provided by the descriptive model. This modelization describes in a generic but evolutive way the static and dynamic characteristics of any complex organization (actual or imagined).

This approach allows to specify Office Information Systems with or without changes in the socio-organizational environment. This is fully consistent with the concept of the Leavitt model used as foundations of the set of tools defining the office as a complex system mixing social and technical problems. During the analysis phase it is essential that the support encompasses all aspects mentioned above. Major decisions about future organization, technical constraints and budget of the project should have been stated during requirements base definition.

The developed **Office Data Dictionary (ODD)** is the starting point for the design phase which benefits from the data stored inside. The interface is determined as a top-down interface since it goes from general specifications to detailed design activities. This procedure should not be fully automated, as this would render the requirements base redundant with design oriented data bases used by logical and architectural design activities. The role and scope of an Office Data Dictionary is clearly different from design problems as it is not only used by designated specialists but also by the people within the investigated organization (organizational analyst, management).

The designers are working onto the ODD after the introduction of consultants and management. Within that work they use the specific extensions of the descriptive model in order to connect typical objects or attributes useful for their design tools to a Leavitt oriented description. In a second step they transfer these data into their specific design environment as a starting description to be worked upon by their individual design tools. Therefore consistency between requirements and design is maintained by the designers but their hypotheses and choices are memorized and stored in the requirements base in order to be queried or modified in the following way:

The Performance Model will give information for the architectural design concerning the performance objectives for the equipment, the technical constraints to be fulfilled as well as the budget constraints.

The **prototyping phase** can be served by ergonomic requirements stored in the requirements base. The bottom-up interface will be used for a regular updating of requirements according to users verification during the prototyping process. The **Conceptual Design** gets useful information from the ODD represented by agents (human resources), processes (activities) and documents handled.

Those intersections will be shown in detail by presenting the important features of the ODD.

3 The Office Data Dictionary (ODD)

Analysis of a real office is a complex process which varies greatly according to organizational, social and technical aspects even if some topics and trends are permanent. Furthermore it consists of clearly distinct kinds of thinking processes. The **feasibility study** is composed of both an analysis of the real situation and an evaluation of the analyzed situation. The **initialization of design tasks** is a collection of constraints and guidelines about future organization on the one hand and an imagination of a future situation on the other. For the final judgment of **architectural solutions** is an objective definition of criteria.

All these processes deal with a common set of data describing different aspects of the office. Therefore the intention for the tool was to provide a computer based support for the well-structured processes of the analysis without impacting creativeness of the users (analysts) by narrow-minded rules or restrictions. Even taking into account AI techniques, computers are still better in repetitive actions and data base management than in brainstorming. Therefore the tool is definitely data base oriented but has a very flexible structure clearly distinguished from traditional data base programs.

It is termed Office Data Dictionary (ODD) with respect to its efficiency and flexibility for office related data storage. It is not a pure dictionary where words are stored without considering their semantic but an encyclopedia with a thematic storage.

3.1 Overview of the ODD structure

ODD is composed of three kinds of software components:

- **Initialization programs** are to be run only once in a fixed sequence. The generic models determine the form of the dictionary. All programs are performing both data entry and data administration functions.

- **On line programs** - according to their scope - comprise domain management, semantic management, population and model management. **Domain management** contains a directory of properties and domains as well as tables of domain data. **Semantic management** consists of a directory of trees, a directory of branches and a directory of properties. **Population management** administrates any office data table via a directory of partitions.

- **Report programs** produce practical reports upon the feasibility study, the interface for design tools and the judgment of architectural solutions to be obtained either on screen or on printer

The ODD - with respect to **domain management** - eases data entry through an active dictionary giving total control on data field entry. The entered value is checked against a list or an interval defined as the authorized domain of values of the property. In case of error, windows with a description of the domain are displayed on the screen. The concept of domain allows the definition of a consistent list of values corresponding to the terminology of the office studied. A library of domains each adapted to a given

organizational branch (bank, insurance, EEC administration, and so on) can be created and stored. With each project or study the necessary domains can be named and inserted into the library. The screens of domain management allow to create, populate and update domains. Domains are linked to properties being grouped with respect to entities handled by semantic management.

The **semantic management** therefore is the definition of aggregated properties handling semantic is-a relationships. Consequently it is possible to add properties first and then define them in classes and subclasses through a partition in the table. The operations can be iterated without limit: Extension of the table can be done even if properties are already stored, however the deletion is controlled to maintain the integrity of structure and contents of the ODD.

These capabilities are extremely useful in order to cope with unplanned situations while analyzing the office. Starting with a generic analysis model the analyst may complete the ODD by areas of interest which he detected during his field work .This modification will never lead to a loss of previously collected data or complex operation on them but to a structured enrichment of the data base while maintaining is overall structure and consistency.

Population management allows to enter, update, delete or query properties in the tables which have been defined by the semantic management before. Each data field entry is checked according to the domain of the property associated to it. Relevance of such operations is on behalf of the analyst. A major item is to make the ODD really ergonomic.

The analyst will perform this operation after a preliminary survey when deciding to study a deeper part of the organization: Different departments of a direction, substeps of a work process and so on. The service given by the ODD handles the operation, keeps track of it, automates part of the creation of new properties according to the initial definitions of the properties of the table, this can be iterated without restriction.

Finally, when working with all the data stored in the ODD the analyst may infer general results from field data. For instance a set of similar office workers can be summarized in an anonymous office worker type. ODD allows storage of such types and manages the link between type and instances.

Within **model management** a set of functions is provided to define semantic relationships. Relationships can be defined at any time. Cardinality rules are checked to ensure consistency of the relationships.

Additional functions have been added to support the analyst, i.e. to reduce repetitive typing, to perform a step by step refinement and conceptualization.

3.2 Implementation of the ODD

A relational data base model for the ODD has been chosen as a basis for implementation for its flexibility and wide use. The relational database is a logical structural limitation in order to facilitate the storage and administration of the complex (various) data.

Due to portability reasons the developed tool runs on an IBM AT with at least 2 MB-RAM and a 60-MB hard disk. The database application has been written on INFORMIX which needs a XENIX SCO Development version. The analyst can execute an automatic creation and initialization of the data base when it does not exist in conjunction with a property dictionary according to the TAM scheme. The analyst,

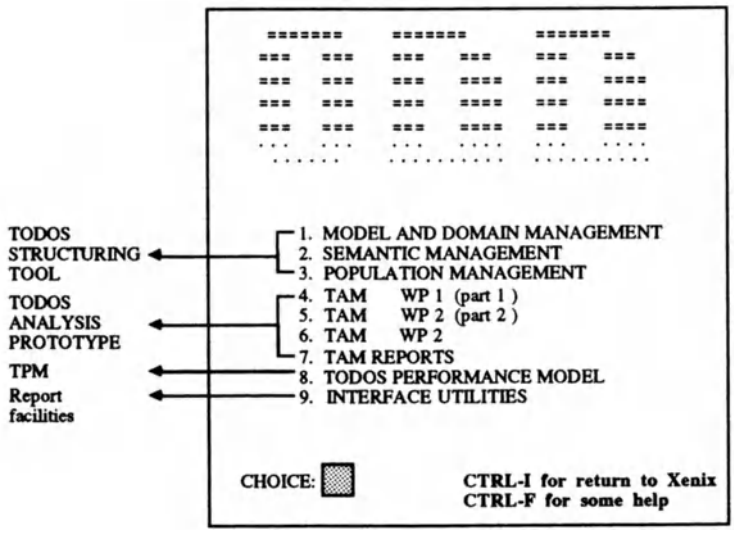


Figure 6: ODD-Menu

wishing to use the querying capabilities of the ODD has to populate the database first by means of simple creation of objects, duplication of objects and partition of objects.

Hierarchical mechanism can be modeled with their corresponding properties in order to systematize the office reality. The creation of domains of values and of properties follows the schema of TAM as well as the introduction of objects (population management), partitions of objects, creation of templates and duplications, updates and queries and editing of reports (Fig. 8).

Choosing **1**, the analyst will get the appropriate mask of the Model and Domain Management in order to fill in further properties and domains of values By typing **2**, semantic relationships can be created. Choice **3** allows the detailing of the different objects according to the real circumstances in the office (Fig. 7). This acts also as the data source for functionalities being used within the specification database of the conceptual design phase. With **4** and **5** - TAM - the scope of possible query masks is demonstrated and ready for choice. Choosing **6** gives further details on documents and processes for the conceptual design. **8** offers the possible insertion into the TODOS Performance Model (TPM).

Further details on the practical use of the ODD within the Case Study context may be found in Chapter 6. In the following we describe how the ODD is used in a requirements analysis.

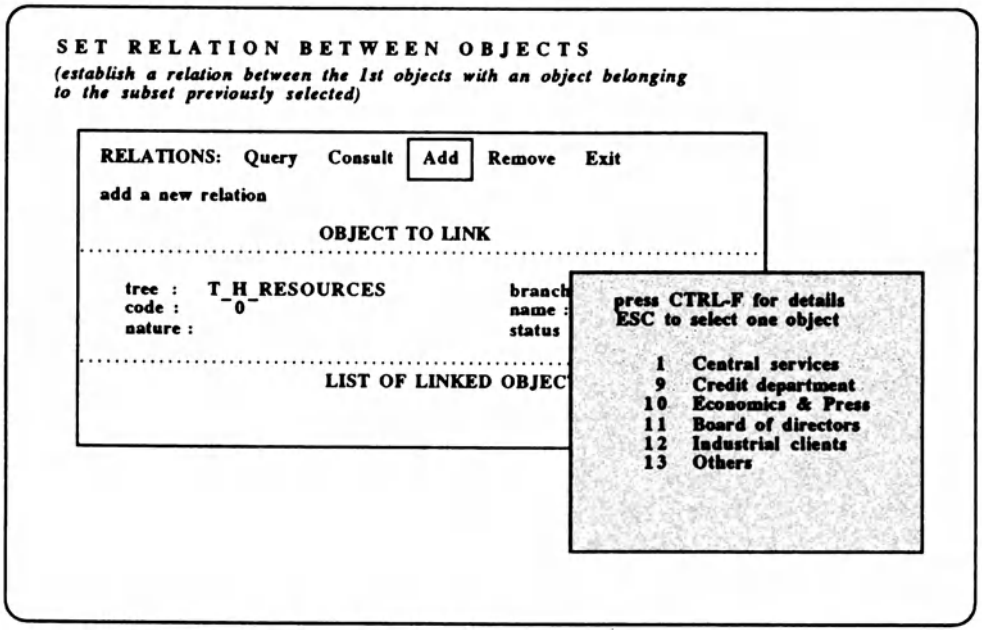


Figure 7: Population Management with TODOS Structuring Tool

4 The use of the ODD within the requirements analysis process

As stated before the requirements analysis is performed according to FAOR Activity Framework. The Activity Framework views office analysis as a learning process and recognizes that objectives in real world analysis situations are not necessarily well-defined and well-structured. The background for problem-solving in the Activity Framework is the concept of soft systems thinking: its application means that the outcome of a learning process is not an optimal solution but leading to purposeful actions to bring about improvement. Each activity within the Activity Framework employs SSM concepts in order to analyze and describe the office problem situation, to understand the situation and to design and structure the analysis method.

The Activity Framework comprises the activities exploration, method tailoring, analysis and evaluation. Going from one activity to the next does not imply a strict termination of this activity. The analyst can always come back to further elaborate on a previous activity if he feels the need to do so (Fig. 8).

The practical intersection between the FAOR Activity Framework and the ODD has a dual meaning. The data gathered during the FAOR activities extend the ODD. At the same time the analysis is well supported by queries to the entities which are already in the ODD. Any new data resulting from the current analysis can be fed into the ODD and hence improve its content. The advantage of using the ODD within a FAOR analysis is the acceleration of the analysis process, easy report generation of i.e. maximum, minimum and average ratios as well as cost indicators for the feasibility study (Fig. 9).

4.1 ODD within the Exploration

Within this first phase of the Analysis Framework a broad understanding of the context and background of problems the client organization is facing should be obtained. Aspects which the analyst think relevant will be further explored by structuring and discussing them with the client. This helps the analyst to structure the problem situation conceptually and to analyze it for consistency and completeness.

On this basis, one or more "relevant systems" are outlined. They have the status of an hypothesis of the eventual improvement of the problem situation by means of changes which seem likely to be both "feasible and desirable". Thus, the relevant system defined in this phase outlines a preliminary and rough model of how support by an office system can be achieved. This model, constructed as a "notional system", is the basis for discussing the study scope with the client. This will also reveal constraints both for the study, e.g. resource limitations, and for the requirements, e. g. the need to take into account the equipment of a certain manufacturer. Of particular importance are the study objectives. They determine the desired results of the study, e.g. the level of requirement detail and the comprehensiveness of the requirements specification.

The results of the exploration phase are summarized in a study brief. This study brief contains:

- a thorough and multiperspective understanding of the investigated office;
- problem aspects or issues to be addressed;

FAOR Activity Framework

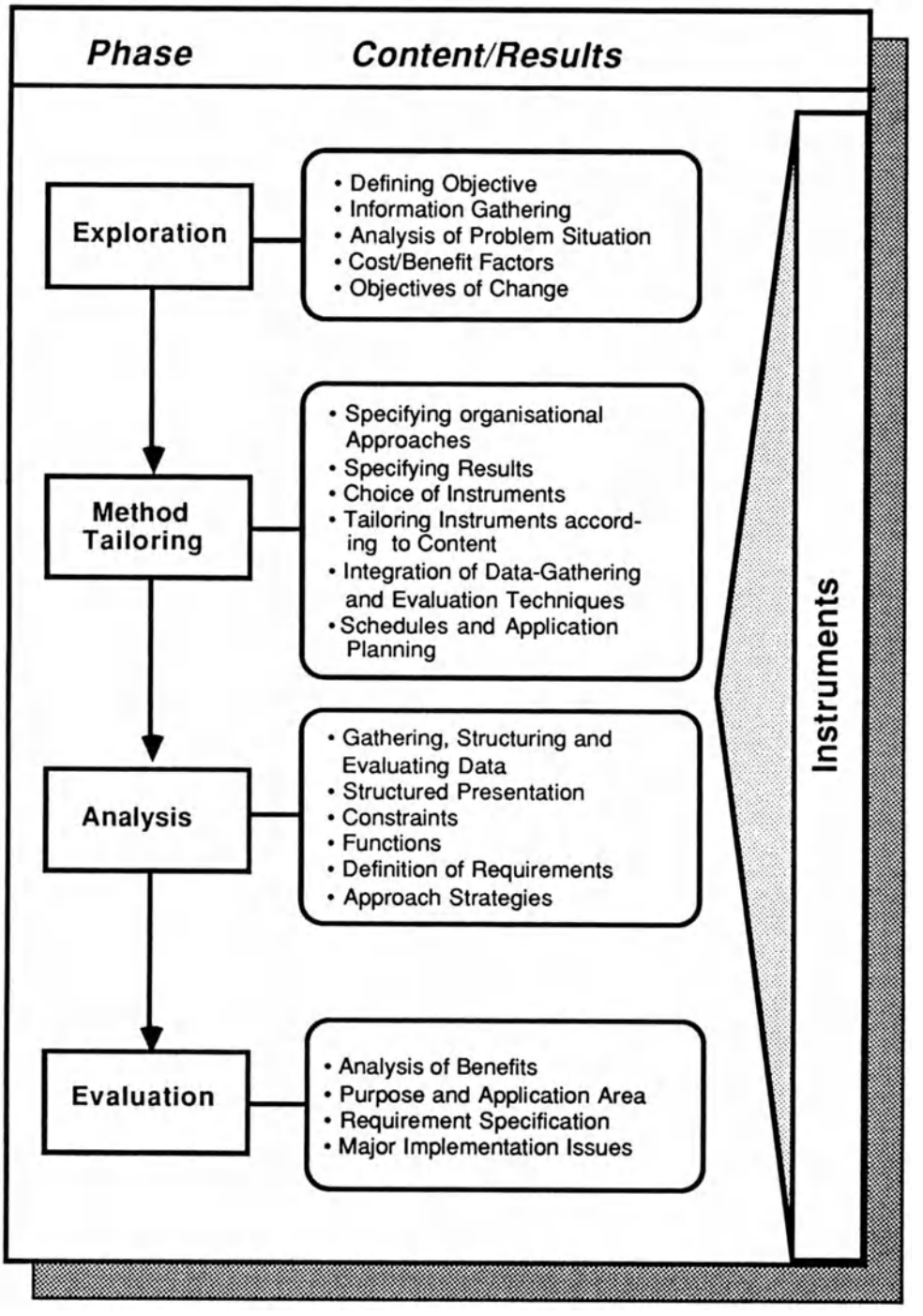


Figure 8: FAOR Activity Framework

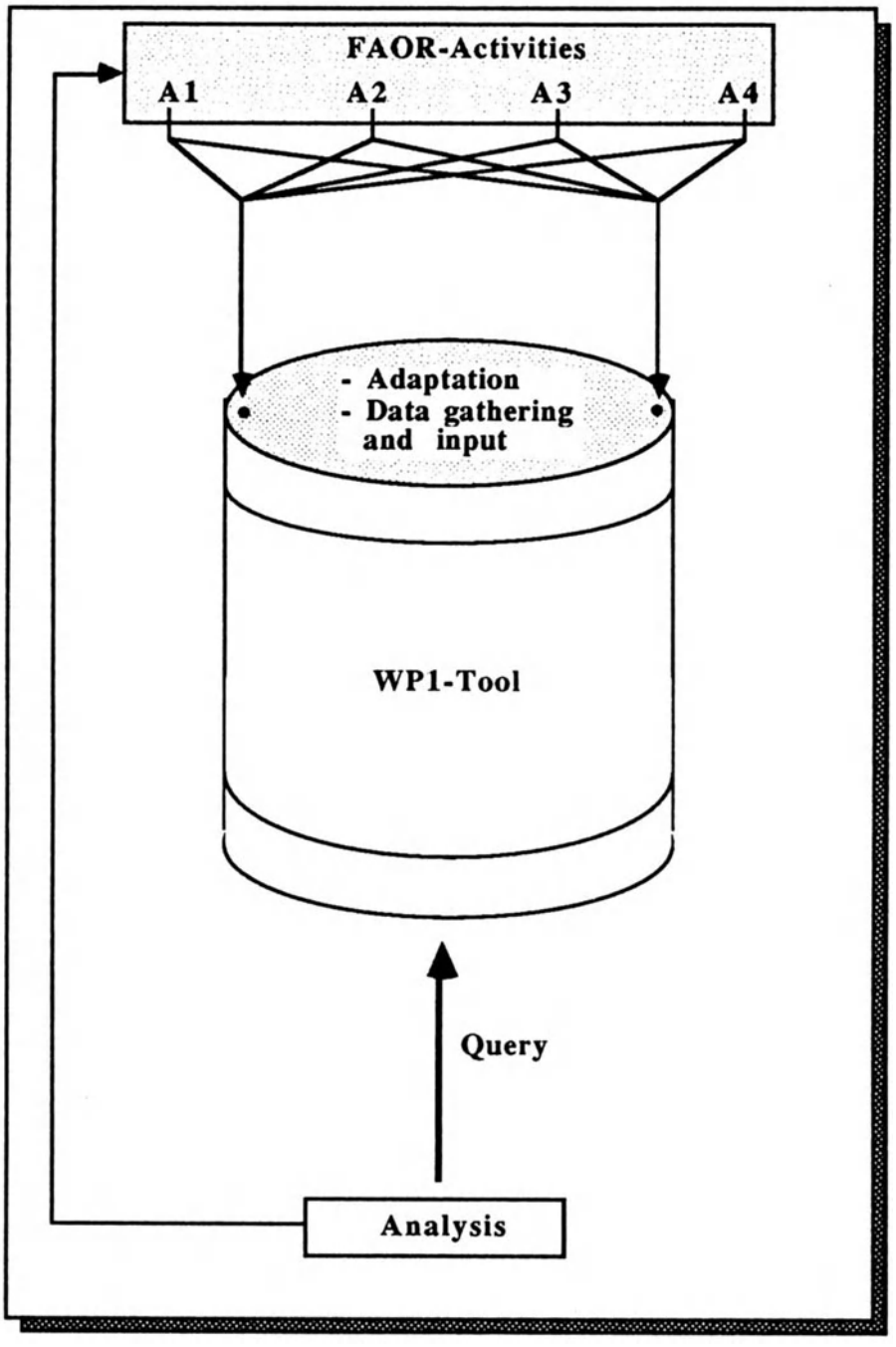


Figure 9: FAOR/TODOS intersection

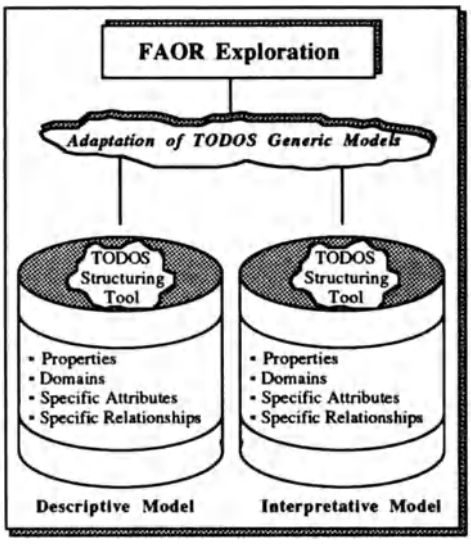


Figure 10: TODOS tool within the Exploration Phase

- a high-level model of the office system reflecting desirable and feasible changes;
- basic criteria for evaluation of changes;
- constraints on a project relating to the office problem situation.

The exploration provides the groundwork for the next phase "Method Tailoring" and for the later detailed analysis.

The *Exploration Phase* can be only marginally supported by the ODD. Aspects which the analyst thinks to be relevant will be further explored and the findings structured and discussed with the client. The systemic analysis is based on different perspectives - according to GOFOR - relevant to the office problem situation. They are preliminary models of the office information system (and its environment) and describe the logical activities which need to be carried out for reasons of completeness. According to the needs of the specific analysis the descriptive model can be further extended by specifying new properties (i.e. new class of equipment or human resource) with their corresponding objects and domains of values. The analyst is always allowed to integrate new properties (semantic-, model- and domain-, population-management) which he thinks to be important (Fig. 10).

TAM programs are the prototype of a defined analysis schema, which may be used as long as it is suitable for the individual analysis. During the exploration phase the need may arise for enter more attributes than actually available in TAM. TAM can be dynamically extended with a refinement of structure and objects according to the

situational necessity. TST allows to enlarge further the descriptive model describing the real world of the office by specifying new properties and domains.

4.2 Use of the ODD within the Method Tailoring

In Method Tailoring, the analyst has to *determine the means* for solving the issues addressed in the study brief. The analyst has to decide what he needs to know about the organization and with what tools and techniques for investigation, modeling and analysis he can obtain this knowledge. At the same time, he must overcome the constraints of the situation. The primary purposes of this phase is Method Tailoring and the setting of the project organization.

Method Tailoring is essentially a comparison between analysis problems identified in the previous phase and analysis capabilities available through instruments. Available instruments constitute possible "pre-structurings" of the analysis covering different aspects. For a detailed discussion of several instruments see Sect. 4.3.

During Method Tailoring different activities occur:

- *selection* of instruments
- *combination* which points out overlapping areas between instruments and possible inconsistencies,
- *adjustment/completion* by pointing out additional aspects which must be taken into account in order to integrate the instruments.

The results of this phase comprise the choice of instruments according to the content of the investigation, an integration of data gathering and evaluation techniques as well as schedules and application planning for the analysis and a short characterization of the output of the following phase. The results are specified in a *study plan* which contains:

- the formal project contract
- a detailed project plan
- the tailored method
- clarification of the client-analyst relationship.

The results of the *Method Tailoring* phase can also be deferred to the ODD descriptive model in order to add new attributes and domains by completing the relationships of office (organizational structure) and document entities (information categories). Furthermore this phase could provide product-objectives and critical indicators for the interpretative model, i.e. the information which is required by the different departments and the degree of utilization anticipated.

Quantitative attributes for phases are also defined in phase 2. The task of the analyst will be to classify the phases according to a certain perspective in order to allow a communication analysis to be made with the descriptive model. The strengths of the TODOS Requirements Collection and Analysis tool are powerful queries on the ODD. Therefore it will mostly support the analysis itself providing queries for the determination of requirements on information flow, communication pattern and hardware constraints.

4.3 Use of the ODD within the Analysis

The analysis phase - as the major activity within the Activity Framework - is the application of tailored instruments with the objective of deriving a requirements catalogue for the planned OIS.

The need for instruments as the practical means for investigation can be better understood if their task and functions are clear. Instruments have to:

- provide the tools and techniques together with the application guidelines necessary to use the instrument for investigating a particular area;
- support the entire process of investigation starting from the gathering of information, through the documentation or modeling of (partial) results, and terminating with the interpretation and the subsequent identification of requirements.

The results of this phase are detailed requirements, which already imply resources for the implementation of the proposed system and thereby offer a possible schedule.

The *Analysis Phase* involves further data gathering and therefore the TODOS descriptive and interpretative model can be filled up with mostly field provided data and more propositive data concerning the future office information system (by the population management of Requirements Collection and Analysis tool). The final outcome of phase 3 is the requirements specification, these requirements will also enter the database. The TODOS support will provide the possibility of complex queries in order to get more detailed requirements on information flows, communication patterns and hardware, whereas FAOR additionally deals with the overall level of requirements. Therefore it is possible to simulate feasible changes by querying the TODOS Analysis Model .

FAOR instruments provide practical means of using aspects of the GOFOR perspectives as part of a multi-perspective office analysis. Instruments support the data gathering on relevant aspects of the office and organizational situations in order to generate models and an adequate specification of requirements. In exploring the office environment, the instruments are related to each other: They cover the same aspects of the office and they are supplementary and supportive in terms of both the scope and the results of the investigation.

The ODD therefore can be interpreted as the support given to the FAOR instruments in order to be flexible and fit to divergent situations and thus facilitate mainly the analysis process (Fig. 11).

FAOR instruments provide practical means of using aspects of the GOFOR perspectives as part of a multi-perspective office analysis. Instruments support the data gathering on relevant aspects of the office and organizational situations in order to generate models and an adequate specification of requirements. One of the FAOR principles is that tools and techniques should not become too deterministic so that they are always open to the integration of more adequate ones. There is always the danger that by pre-specifying the kinds of aspects to be investigated, the range of problems which an instrument is able to solve is significantly reduced; this, in turn, introduces the possibility that the relevant issues may lie beyond the capabilities of the instrument.

In exploring the office environment, the instruments are related to each other: They cover the same aspects of the office and they are supplementary and supportive in terms of both the scope and the results of the investigation (Fig. 12).

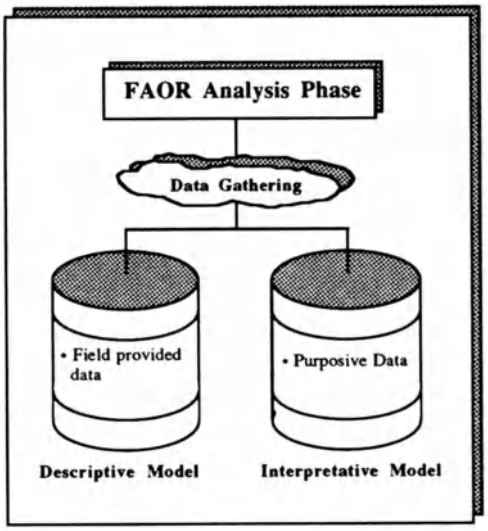


Figure 11: TODOS Models within the Analysis Phase

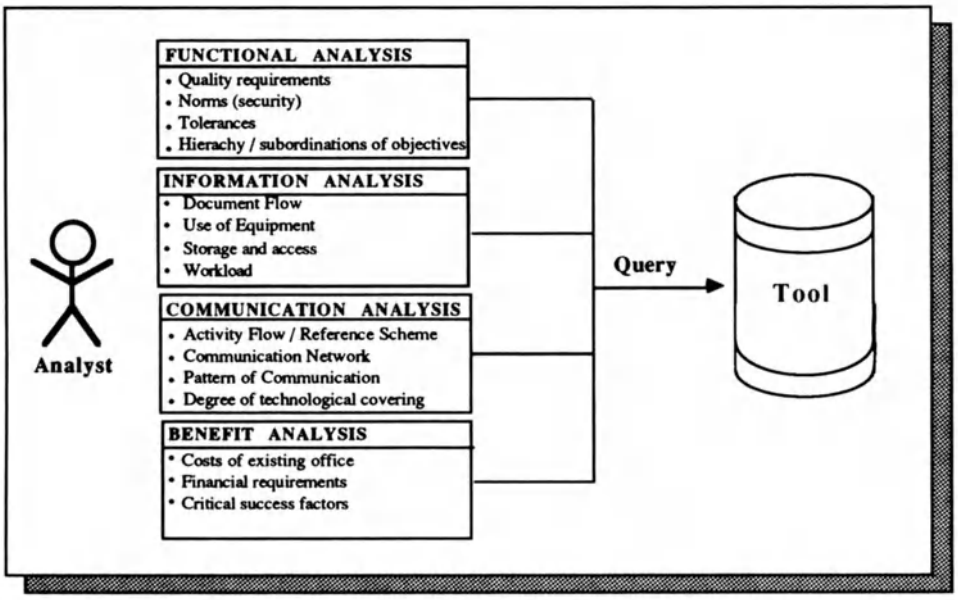


Figure 12: TODOS Analysis tool and FAOR instruments

The Functional Analysis Instrument (FAI) identifies, defines and characterizes the essential functions of the office in order to fulfill the organizational objectives and to provide a rational basis for requirements specification. By focusing on the purposiveness of office work in relation to the needs and requirements of the wider system, the application of the function perspective abstracts from the implementation. The analyst distinguishes conceptually between tasks and activities on one hand and functions on the other.

The result of the Function Analysis Instrument is a determination of the pertinent properties in terms of characteristics like objectives, norms, values, constraints, quality requirements, contradictory objectives, conflicting goals, controversies, structuredness and interdependency. The ODD will provide informations on the implemented information system, the number of diffused documents, different phases and their classification as well as critical indicators and quality standards concerning the diverse product-objectives. The Function performances, represented by norms, values, constraints and quality requirements, can be easily inserted via the TODOS Analysis Model.

The output of the Communication Analysis Instrument (CAI) is a specification of the communication requirements of the future office information system. TODOS ODD contains a number of means of modeling communication activities in an appropriate way: It is able to list the basic activities of individuals. The analyst will acquire a reference scheme with a list of the human resources involved in the analysis as well as a documentation or relationship allowing him to determine which unit a human resource belongs to, what kind of tool it uses etc. The ODD therefore helps to illustrate the communication patterns during the communication analysis. One result of the communication analysis is the overview of people's access to archives which can be easily stored via the TAM screens. Accesses to certain archives, periodicity, frequency and the time, a folder is for example, withdrawn from an archive can be related to the person investigated. By using the report generator it is possible to obtain information of the total of the use of archives. TAM also helps to get an overview of the equipment supporting the communication process by storing the use of instruments related to the communication activity of a person. The ODD is able to list the basic activities of individuals in the investigated organization. The results of the application of the Information Analysis Instrument (IAI) comprise several degrees of relationships in order to determine how well the information object supports the activities of the office workers; these findings enhance part of the investigation carried out by the Function Analysis Instrument. The ODD provides useful information upon the life cycle of documents, their physical characteristics, periodicity, frequency and duration as well as the use of equipment by the different documents. The objectives of the User Needs Analysis Instrument (NAI) within the FAOR framework are an identification of user needs and expectations for fulfilling them in a future office information system, office information systems requirements especially related to the fulfillment of human needs and adequate indicators for a human benefit evaluation as part of the FAOR Benefit Analysis. The NAI particularly evaluates the current situation, the expectations concerning the future and attitudes in order to identify needs of an office worker with some degree of reliability. A moderating factor is the background of the person. The evaluation of the current situation by an individual shows where the individual as well as the analyst see room for improvement. It shows whether the person is ready to get actively involved in making improvements or whether he feels frustrated so that the current situation inhibits his

perception of any improvement. The NAI is an instrument which differs entirely from other instruments in that it is dedicated to the collection of social information, whereas TODOS is able to offer only mild support as being oriented towards (technical) system design.

5 Concluding remarks

The use of the ODD is twofold: it is not just a means for supporting the data gathering process via inserting data but also an information supplier via queries to the database. It is a helpful support within the analysis phase itself. The data resulting from the analysis of an organization can be easily inserted into the ODD so that it consists of mostly field provided data and more data necessary for the future office system. The final outcome of an analysis is the requirements specification. These requirements will also enter the database.

The TODOS ODD is very powerful and its complexity causes difficulties in practical handling. In future emphasis will be put on the user interface as it is not simple and needs more support through windowing techniques and mouse, as the analyst is no system-specialist. However the ODD is a useful support to former manual analysis instruments (FAOR) providing a common database for the total of processes during an analysis with a flexible user-mode. Those data are the basis for preliminary architectural solutions and evaluations in order to get a common subject of discussion between analyst/designer and management of the investigated organizations.

Chapter 3

Conceptual Design

Mariagrazia Fugini, Barbara Pernici, Silvano Pozzi,
Jean René Rames, Colette Rolland, and André Vignaud

1 Introduction

This chapter illustrates the TODOS conceptual design phase. Such phase aims at building a **conceptual schema** of the Office Information System (OIS) that represents the functions of the OIS in a formal way, as independently as possible of implementation details. This schema is checked for consistency and correctness and for quality of the design, then is passed on to the prototyping and architecture selection phases of the TODOS environment (see Chapter 1).

The tool that supports conceptual design in TODOS is **C-TODOS** (standing for Conceptual-TODOS). This is organized around a specification database which stores the OIS specifications formalized according to the **TODOS Conceptual Model (TCM)**. TCM concepts derive from both the SOS office model (Bracchi and Pernici Winter 1984) and the REMORA methodology for Information System (IS) design (Rolland and Richard 1982). TCM represents office functional requirements in their static (structure of office elements) and dynamic (related to system behavior) aspects. The **TODOS Specification Language (TSL)** provides constructs bound to TCM concepts; it allows for a formal description of office conceptual schemas based on TCM.

Since the specification database is the core tool of the conceptual design phase, one of the key objectives of C-TODOS is to provide easy-to-use interfaces to interact with the database. Interfaces allow the designer to insert, modify, delete specifications in the database and to query the stored specifications in order to extract information and reports on the design activity.

In addition, C-TODOS includes mechanisms to handle incomplete specifications thus supporting **incremental design**, and to check the **consistency**, **correctness**, and **completeness** of the specifications. At the end of the conceptual design phase, the contents of the specification database is taken as a correct input by the rapid prototyping and architecture design phases.

The outline of this chapter is as follows. In the remainder of this section, we first discuss the role of conceptual design in the development of an OIS (Sect. 1.1); then we present the features that an OIS conceptual model should exhibit to support such role (Sect. 1.2). Next, we introduce a taxonomy of models and tools for conceptual design of OIS (Sect. 1.3); the references to models, tools, and methods for Information System conceptual design are obviously frequent in this taxonomy, because of the strong relationship and influence existing between IS and OIS.

The sequel of the chapter presents in detail conceptual design in TODOS. In Sect. 2, the conceptual model TCM and its specification language TSL are illustrated. Sections

3 and 4 focus respectively on the architecture of C- TODOS and the support for the modeling activity provided by the tool, and on the interfaces of C-TODOS to its users. Methodological guidelines for the TODOS conceptual design phase are presented in Sect. 5. C-TODOS as a cooperative design tool is discussed in Sect. 6: OIS applications can be designed separately by different analysts skilled about various aspects, and then merged into a global OIS schema. This is a cooperative design activity that benefits from computer based support. Finally, a brief discussion on the TODOS conceptual design method is included in Sect. 7.

1.1 Role of Conceptual Design in the Development of an OIS

The OIS can be considered as a **model** of the office world which provides office workers with a common understanding of their work environment. In other words, while every individual in the office may have his own, subjective view which may (or may not) match with the perspectives of the other workers, it is necessary for them to create a common view of office data and activities. This common view is provided through the OIS which supports them in their activities.

More abstractly, the OIS is considered as a set of **elements** having **structural** and **behavioral relationships** with each other and with the environment. The structure describes the components and properties of an element. Behavior refers to state transitions caused by the occurrence of facts externally and internally to the office world.

For OIS, conceptual design has the same meaning that it does in the Information Systems area (Schneider and Wasserman 1982). In particular:

a) *Conceptual design activities are modeling activities requiring formal specifications*

Designing an OIS means to model and describe those aspects and parts of the office reality which are relevant to the construction of an automated system. Thus, conceptual design means **real-world modeling** and it is an **abstraction activity**.

Specifying an intended OIS means to prescribe its properties (structure and behavior) and the interface with its environment (office agents and communications from/to the external world). A conceptual specification defines a class of systems which functionally fulfills the requirements of the specification.

The **OIS conceptual schema** is the result of the modeling activity; it formally describes the set of elements of the target OIS, their relationships, and their behavior. It is both an abstract representation of the office real world (world model) and an abstract view of the target OIS (system model).

b) *Conceptual design deals with semantics and functionalities, not with technological or implementation issues*

The OIS conceptual schema is a **conceptual** - in the sense of the ISO report (ISO 1982) - **specification** of the OIS. Such specification is not a completely detailed description of the target system, rather it prescribes its intended structure, behavior, and interface with the environment.

The conceptual schema should not deal with technological details of the OIS, rather with the **functions** of the OIS. For example, conceptual design should

concentrate on providing a complete description of the office activities (e.g., conditions for the execution of the activities, agents responsible for them, documents manipulated) that can be supported by hardware or software tools. It should not contain any suggestions about these tools. The technological issue has to be solved in other, subsequent phases of the development process, namely in a logical design phase and during implementation of the OIS. In TODOS we devote one phase to technological issues, the architecture selection phase, as described in Chapter 1.

c) *Methodological tools for conceptual design are models, formal languages and computer aided design tools*

According to the nature of OIS conceptual design, models and formal languages are necessary to deal with the modeling and specification activities. In addition, the TODOS approach puts the emphasis on the synergy between the methodological and modeling issues, and the software tools supporting each step of the method.

1.2 OIS Conceptual Models: What do we need from them?

An OIS conceptual model is intended to support the designer in the conceptual design activity, which is a modeling activity requiring a clear formalism, abstraction capabilities and semantic power (Brodie et al. 1984, Hull and King 1987).

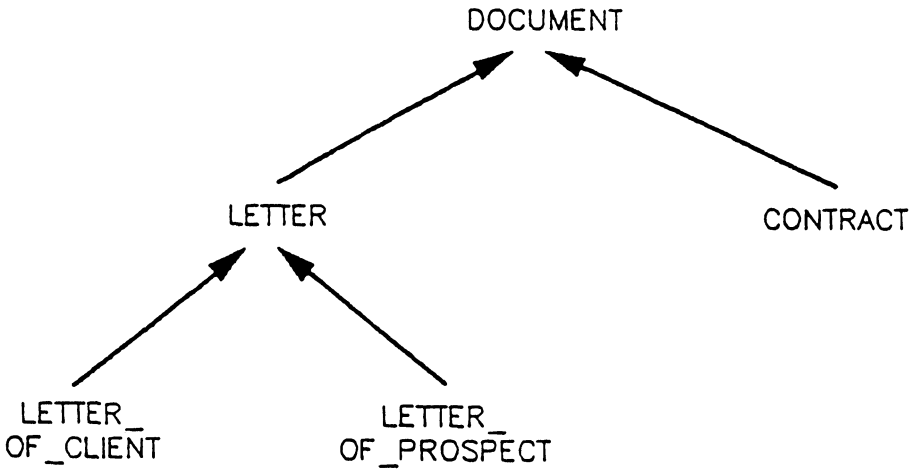
a) First, the designer must be able to represent similar individual elements of the office world into **classes of elements**, and, analogously, to represent **classes of relationships** among the element classes. Hence, the OIS conceptual model should provide the analyst with **classification mechanisms** that allow him to distinguish the **instance** level from the **type** level. An instance is a value or a particular individual of an entity type. An example is that "22 years" is a particular instance of the "age" type. Classification is needed to define a type from a class of similar instances. For example, viewing a set of individual client instances as one CLIENT is a classification.

We use the term **entity** to refer to a **class**, and **entity instance** to refer to an instance of a class.

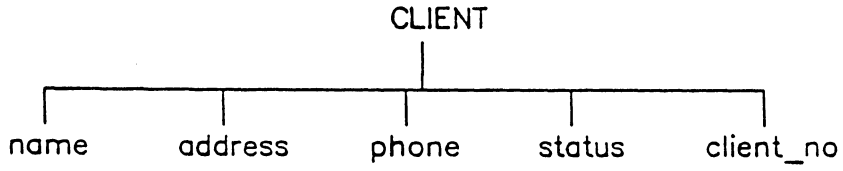
b) During the modeling activity, the analyst needs to use **abstraction** to hide some details of the application and to concentrate on general, common properties of set of entities, thus representing this set as one unique entity. Abstraction is usually included into conceptual models through the **generalization** mechanism.

Generalization is an abstraction mechanism that allows to consider a set of entities (named **specialized entities**) as a complex entity (called the **generic entity**). For example, the types LETTER OF_CLIENT and LETTER_OF_PROSPECT can be viewed as two specialized entities of the generic entity LETTER through generalization (see Fig. 1a).

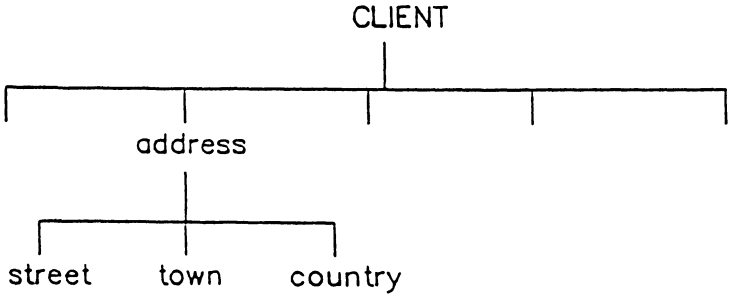
By using generalization, the emphasis is on the similarities of entities. Furthermore, the repeated use of generalization allows the designer to build hierarchies of types, as shown in Fig. 1a. Generalization mechanisms are needed in an OIS conceptual model to help the analyst in the classification of types of office elements in a given conceptual schema.



a) generalization hierarchy



b) the aggregation mechanism



c) aggregation hierarchy

Figure 1: TCM abstraction mechanisms

c) The design process can be simplified if complex entities are viewed as the synthesis of simple entities. This view is provided by another abstraction form called **aggregation mechanism**. Aggregation is the abstraction by which an entity is built from its component entities. For example, the CLIENT entity of Fig. 1b is built from the types: name, address, phone, status and client_no.

Aggregation as an abstraction form is very helpful because it gradually makes visible the structure of an entity and the way individual components of the entity relate to the aggregate entity and to each other.

The repeated use of aggregation allows to build **nested structure hierarchies**: the components of an aggregate entity can in turn be defined as aggregate components. For example, the address type can be built upon the simple types street, town, and country (Fig. 1c).

d) In order to guide the analyst in selecting those aspects and parts of the office world which are relevant to the design, the OIS conceptual model should be based on a set of **predefined types**. For example, "DOCUMENT" is a usual predefined type in OIS conceptual models: it represents documents produced, filed, received and transmitted in the office. "AGENT" is another common predefined type. "ACTIVITY", "EVENT", "MESSAGE", "DOSSIER" are other examples of needed predefined types which are at the basis of several existing office models.

e) The set of predefined types determines to a large extent the **semantic richness** of the OIS conceptual model. The main criteria for the evaluation of the semantic richness of a model are illustrated in the following.

- **Minimality of the set of concepts:**
  In order to produce "canonical" or "normalized" OIS conceptual schemas, the model concepts must be orthogonal, i.e. non redundant to each other. This helps the designer in that he is guided in identifying the concept of the model that best represents a given element of the office.
- **Semantic power of the set of concepts:**
  Predefined types play the role of filters in the analysis of the real office world: only those parts which fit into the model concepts will be represented in the OIS. Thus, in order to obtain a satisfactory representation of all aspects of the office world which are perceived as relevant to the design process and to the target application, it is fundamental that the semantic richness of the set of concepts provided by the model be powerful enough.
  In particular, models providing only **static concepts** (i.e. predefined types related to the description of office elements and their structure) proved inadequate. Concepts are needed that describe the *behavior* of the OIS and to express how and when changes occur to the OIS elements.

f) The set of concepts of the model must be able to deal with the **key aspects of OIS**, which distinguish OIS applications from traditional IS applications. Let us mention them:

- *unstructured data* like texts, annotations, graphics, phone calls are often manipulated in the office. They are frequently used in groups rather than being

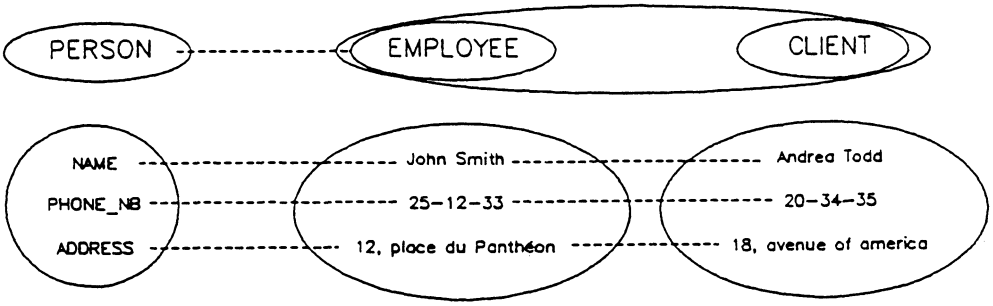


Figure 2: Using classification, aggregation and generalization during the design process

manipulated as individual pieces of information. For example, a signature, or a text have no real meaning if considered independently from one another: they are meaningful when grouped together into a letter.

- *multimedia data*: documents manipulated in offices integrate purely numerical data, texts, images, graphics, and voice.
- *temporal aspects of office activities*: most of the office activities are scheduled, either according to absolute time (July 1st, 1988; every first monday of a month, etc.) or relatively to other activities or events (one week after the order has been sent, two months after the appointment, etc.).
- *communications* among office workers and of these with their environment.

g) The model should support two kinds of **approaches to the modeling** activity, a bottom-up and a top-down approach. As an example, consider the PERSON entity shown in Fig. 2. Using a **bottom-up** approach, abstraction is viewed as a synthesis of simple entities: this enables us to understand a complex object. We start with observations, the instances (John Smith, 25-12-23, 12 place du Pantheon, Andrea Todd, 20-34-35, 18 avenue of America) to which we apply classification to produce types (EMPLOYEE, NAME, CLIENT, PHONE_NB, ...). Aggregation and generalization can then be used to classify and structure types into new generic and aggregate types. For instance, NAME, PHONE_NB, ADDRESS are aggregated to define PERSON; EMPLOYEE and CLIENT are specialized types from the generic object PERSON.

Alternatively, an analytical **top-down** approach may be used to decompose complex types. We start with a complex type, and decompose it into its components, through specialization and instantiation, down to the instance level. For example, starting with PERSON, using aggregation we decompose it into NAME,

PHONE_NB and ADDRESS, while using generalization we structure it into EMPLOYEE and CLIENT. Instantiation of NAME will give <John Smith> and <Andrea Todd>.

1.3 State of the art in models and tools for OIS conceptual Design

Conceptual design has progressively increased its role in methodologies for IS and Database (DB) development methodologies and is today a largely-accepted phase in the development of these systems. Since the major directions in IS and DB methodologies have a deep relationship with methods and tools for OIS development, because of the many similarities existing between the IS, DB, and OIS world, in this section we give an overview of these directions.

a) Models and methods

The current trend is to base IS and DB conceptual design on **models** able to capture the **semantics** of the real world in a precise yet natural way. Thus, "semantic" data models have been proposed, such as SHM, SHM+, RM/T, SDM , TAXIS (see (Borgida et al. 1982 and Brodie et al. 1984) for a survey) or OICSI (Rolland and Proix 1986). "Semantic" stands for independence of the model of system implementation issues, and for expressiveness of the model constructs and properties in capturing the semantic contents of and the relationships among the objects of the real-world being modeled. Often, in these models, concepts can be grouped together using constructs borrowed from the models used in Artificial Intelligence, such as entity, classification, aggregation and generalization.

The emphasis put by traditional data modeling on the structural properties of objects has gradually extended to include **system behavior**. Most recent models provide concepts to integrate in the same modelization both structural and behavioral properties of IS: REMORA (Rolland and Richard 1982), CIAM (Gustafsson et al. 1982), INFOLOG (Carmo 1985) are examples of methodologies based on **behavior-oriented models**.

An increasing number of **methodologies for OIS** are based on conceptual models. These proved extremely helpful to cope with a relevant feature of the OIS development process, that is, the intensive relationship between different groups of people: analysts, end-users, system experts, organization management experts, and behavioral scientists. The major cooperation problem is the need for a common formalism to express the world each is talking about. Office conceptual models, being abstract and yet precise enough to be understood by the different people involved in the design, contribute to merging of competing viewpoints by providing a common universe of discourse.

OIS conceptual models can be classified in the following main categories:

- data-based models
- process-based models
- object-oriented models
- mixed models

Data-based models group data in forms which are similar to paper forms used in the manual office (as illustrated in Chapter 2).The basic elements are types of data and operations on data (storage, retrieval, manipulation, transmission). Office activities are seen as a series of operations on data.

An office model based on data is used in the *Office Development Methodology* **ODM**, developed in Esprit project n. 59 "Minstrel" (Dunnion et al. 1985). ODM model uses **objects** or **entities** to model "things" (real-world objects) in the office such as workstations, filing cabinets, folders and documents, and in the enterprise such as departments, employees and customers. These entities are organized in types.

Process-based models can be distinguished in network models and goal-based models.

Network models describe sequences of activities performed in the office. Each procedure is decomposed in steps interconnected to each other. Triggering conditions regulate the transitions from one step to another. The first example of this approach is **SCOOP** (Zisman 1978) that uses Petri Nets augmented by production rules to represent office activities. The office is seen as a system of asynchronous event-driven tasks that may occur concurrently. The production system consists of rules, of a database that stores data of the various system states, and of a rule interpreter. Another example is **Information Control Nets** (ICN) (Ellis 1979) that models office activities in a procedural way and allows for simulation of office work. An important aspect of ICN is that the model has a mathematical nature and therefore is rigorous and useful to prove properties about the modeled system (correctness of task coordination, of information flow, etc.).

In *goal-based models* office procedures are modeled by stating goals; the system will then determine the correct sequence of steps to be undertaken to achieve a goal. Thus, the design activity does not require to consider in advance every particular case that may arise in the system, because it is the system that will adapt its behavior to handle each case on a goal-oriented strategy. This approach is used primarily for systems in the area of Artificial Intelligence and expert systems. Examples of systems based on these concepts are POISE (Croft and Lefkowitz 1984) and OMEGA (Barber 1983).

Object-oriented models have become very popular because of the SMALLTALK (Goldberg 1981) system that showed that a similarity exists between office elements acting in their environment and objects moving, deciding and transforming themselves within an object-world. A similar approach has been followed by OPAL (Ahlsen et al. 1984) that uses the central concept of packet, which contains data and actions (OPAL has evolved in AVANCE, developed at SISU).

Mixed models explicitly consider various types of elements as the basis for system specification, and define the relationships among these elements. The different elements are often grouped into submodels; an example of mixed model is the **Semantic Office System** (SOS) (Bracchi and Pernici Winter 1984) that uses three submodels - a static, a dynamic, and a control and evolution submodel - to represent separately the structure and behavior of OIS elements.

b) Computer-based tools

As a support to both *IS* and *OIS design* methodologies, computer aided development tools are currently becoming more and more popular.

Several commercial proposals exist, such as MAESTRO (Philips), FOUNDATION

(Arthur Andersen & Co), EXCELERATOR (James Martin & Associates), IEW (Arthur Young), GRAPHTALK (Rank Xerox).

Research in automated tools to support IS/OIS design is rapidly making progress and a number of tools have been proposed. RAMATIC (Johansson 1984, Eriksson 1984) is a computer graphics based tool whose main purpose is to support ongoing discussion of a project group in real time. RAMATIC supports system analysis by supplying a specific model to each group involved in the project; the various descriptions are then integrated.

OFFIS (Konsynski et al. 1982) is a system designed to facilitate an interactive and iterative office analysis and design process, providing the planner/designer of the automated office with a flexible method of analyzing system features and constraints. The OFFIS methodology is based on a technical model of the office, and office elements are studied in great detail.

QUINAULT (Nutt and Ricci 1981) is an experimental system for OIS modeling and analysis based on an extended version of ICN. Quinault integrates many features of ICN models in a highly interactive personal computer system.

The **POISE** system (Croft and Lefkowitz 1984) is centered around a data model and can be used both to automate routine tasks and to provide support in more complex office tasks by assisting the pattern of work in progress. The POISE system automates some tasks and keeps an agenda of activities, the status of which can be examined using a natural language interface. POISE can operate in two different modes: interpretation and planning. In the **interpretation mode** the user invokes tools directly and POISE attempts to recognize the user's goals in the context of its procedure library. In the **planning mode**, the user invokes a procedure and POISE executes the procedure as far as possible, based on the procedure's goals.

OMEGA (Barber 1983) describes office work in terms of goals and actions and represents it by using constructs borrowed from Artificial Intelligence. An office worker performs a task by establishing a goal, and the system will then attempt to achieve this goal by decomposing it into a hierarchy of subgoals, where a leaf-node subgoal can be achieved using OMEGA's knowledge about it. OMEGA has been designed as a learning system; it can be easily enriched with new procedures and constraints on a learn-by-example apprenticeship base.

AMS (Activity Manager System) (Tueni et al. 1988) merges OIS and AI techniques. It comprises an abstract mixed-type model used to handle the office static aspects (agents, roles, documents, and operations are the basic entities). Activity Networks, that can be defined by the user, are a kind of augmented Petri Nets: logical operators - analogous to "places" - deal with synchronization; the activities act as "transitions" augmented by rules. AMS represents office knowledge in a declarative way: "packets of abstract knowledge" and "explicit sequences" are the AMS approach to knowledge reusability and to dynamic construction of sequences of activities at different levels of abstraction.

Tools employing AI techniques to specify office procedures have the disadvantage of being inefficient, and also of hiding the flow of activities in the organization. For office procedures a mixed approach, where some of the procedures are specified procedurally and others are specified through their goals, provides a more effective representation. Some of the tools that were prompted by this observation are the following.

OICSI (Rolland and Proix 1986) is an expert design tool for supporting the RE-

MORA methodology. The goal of OICSI is to progressively generate the REMORA conceptual schema from a description of the application domain expressed in a subset of the French natural language. Starting with the analysis of an initial description, the system provides syntactic trees as internal representations of the natural language sentences of the description. Then, using a set of interpretation rules and structuring rules, the system constructs a semantic network corresponding to the IS conceptual schema. The whole design is interactively done with the designer.

DAIDA (Jarke and DAIDA Team 1988) is a knowledge base management system for controlling IS development and maintenance. It is based on a semantic representation of the relationships between design objects (i.e. any software object or document in the world/system modeling process), design tools and design decisions. A decision class links two object classes through a tool that realizes the transformation of one object into the other. This semantic description is given using the Conceptual Modeling Language (CML), based on an object-oriented model with generalized instantiation hierarchies and embedded time calculus. The description is stored into a layered knowledge base whose levels correspond to the knowledge of objects, decisions and tools. Such description enables DAIDA to support the following tasks: selection by the user of tasks and tools to be applied; backtracking of decisions and revision support by re-playing the decisions in case of modifications; documentation facilities on the relationship between development objects and the tools that created and managed these objects.

2 The TODOS Conceptual Model and Specification Language

The TODOS conceptual design phase is based on the TODOS Conceptual Model (TCM) and the TODOS Specification Language (TSL) which are presented in this section.

We first give in Sect. 2.1 a general overview of TCM. Then, in Sect. 2.2, we present in more detail each TCM modeling concept and the corresponding TSL constructs.

2.1 A TCM Survey

In conceiving a model for conceptual design of an OIS, our goal was to meet the requirements that we have presented in Sect. 1.2. Accordingly, we derived a model whose main features conform to those requirements and that are outlined in this section.

Six predefined concepts (entity types) and three structuring mechanisms have been introduced in TCM.

a) The predefined concepts

They are: **document, object, message, agent, action** and **event.**

In defining TCM, a special effort has been devoted to define concepts for representing both static and dynamic aspects of the office world. IN TCM there are predefined static and dynamic concepts. Predefined static concepts that model **static aspects** of the OIS are the document, object, message and agent entities. Predefined dynamic concepts that model **dynamic aspects**, i.e., the behavior, of the OIS are the action and event entities. All the model entities derive from these predefined types.

Each concept in the model refers to a **type**. For example, the LETTER type of document describes the documents that have a sender, a receiver, a date, a signature, and a text portion, and that are usually sent in/out of the office for communications. A class is a set of tokens having the same (or at least similar) properties.

A class of elements like CLIENT (Fig. 1b) belonging to one of the six predefined types of the model is called **TCM entity**.

Entities can be static or dynamic entities. For example, ORDER is a static entity. TCM **static entities** represent either physical elements of the office (TCM documents, messages, agents) or more abstract concepts like office information (TCM objects). In particular:

- **document** entities model office documents such as letters, forms, dossiers.
- **message** entities model temporary office communications like phone calls or electronic mail messages.
- an **agent** entity defines the role played by individual office workers or groups of them, when they interact with the OIS. Examples are senders and receivers of messages.
- **object** entities represent office information needed by agents or by the OIS to perform the office work. Examples are loans, states of progress of letters, meetings.

TCM **dynamic entities** allow to describe in the OIS conceptual schema the rules according to which static entities behave over time (when they change and how). In particular:

- **action** entities represent activities that are performed in the office and that modify the status of static entities. Examples are copying a document, modifying an address.
- **event** entities ascertain state changes of static entities that trigger actions. Example are message arrivals, end of the week, address modification.

In summary, actions allow to describe "how" static entities change, while events describe "when" actions must be performed. Since the execution of actions can in turn modify the state of static entities, new state changes can occur upon actions that is ascertained by other events, an so on. These sequences globally describe the **office behavior** over time.

Any TCM entity is characterized by its **properties**. For example, the entity CLIENT is described by the properties name, address, phone, status and client_no (see Fig. 1b); the event NEW_CONTRACT is characterized by the event date and the client who signed the contract; the action CONTRACT_CREATION has the property contract_no which is the number of the new contract.

The six predefined TCM concepts are mutually orthogonal (their intersection is empty). According to what discussed in Sect. 1.2, the orthogonality issue helps to guarantee minimal conceptual schemas, since a concept is guided to be described through one specific type of entity.

b) The structuring mechanisms

They are **generalization, reference** and **structure**. They can be applied to each of the six predefined TCM entities.

Generalization corresponds to the concept used in AI which is called "is-a". It allows to model hierarchical relationships between entity types. An entity E_j may be defined as a specialization of a more general entity E_i (E_j is-a E_i): E_j inherits all the properties defined for E_i. Multiple supertypes are also allowed. Moreover, subtyping may be restricted to instances with particular property values through a condition.

Fig. 3a shows a generalization hierarchy for agents, documents, and events are derived from static entities and dynamic entities. Generalization can be used to describe relationships between the predefined types of TCM itself. More generally, all the modeling concepts used for the description of office systems have been used to describe TCM itself.

The **reference** mechanism is used to express logical relationships between two entities. It is a binary relationship mechanism. A reference between two entities E_i and E_j has a reference name which is considered as a property of E_i (named the source entity).

Fig. 3b shows an example of reference between the document entity CONTRACT and the BANK agent. This reference links the instances of CONTRACT with the instances of BANK.

Four types of reference relationships are defined in TCM:

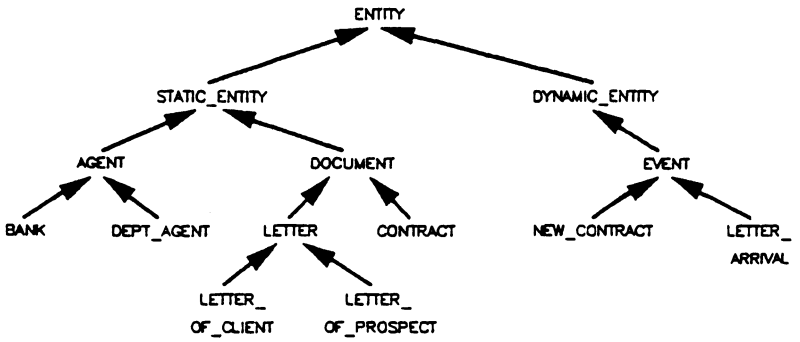
- a **"ref-to"** reference denotes a pointer to a referred entity, giving to the source entity the visibility of the referred entity.
- a **"copy-of"** reference indicates that the source entity contains a part of the structure of the referred to entity. Updates to values of the source entity are not propagated to the copy.
- a **"same-as"** reference is analogous to the "copy-of" reference, but determines a referential integrity constraint on the referred entity.
- a **"view-of"** reference is analogous to the "copy-of" reference; however, updates to values of the instances of the source entity are propagated to the instances of the "view-of" entity.

The **structure** mechanism defines the properties of office elements. The structure concept has two predefined subtypes: **association** and **aggregation.**

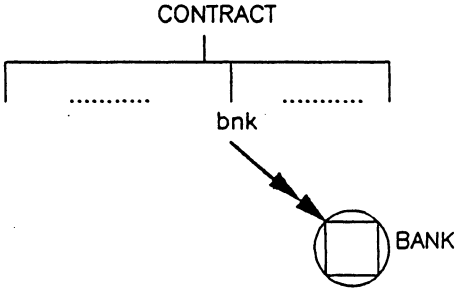
The **association** mechanism specifies a property that includes a repeating group of properties. In Fig.3c, for example, Names is defined as an association of Name. The association mechanism is similar to the abstraction mechanism also named association added to SHM (Smith and Smith 1977) by Brodie in the ACM/PCM methodology (Brodie and Silva 1982); in our case, grouping is limited to properties of entities.

The **aggregation** mechanism defines a collection of different entity properties. Fig. 1b shows an example of entity aggregation: CLIENT is an aggregation of five properties (client_no, phone, name, status and address). Fig. 3d is a more detailed picture of the CONTRACT entity shown in Fig. 3b. Here, CONTRACT is an aggregation of:

- characteristics of the contents of the contract: header and textual_part;

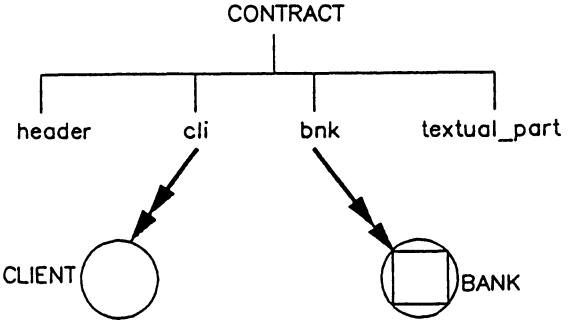


a) the generalization mechanism



b) the reference mechanism

c) the association mechanism



d) structure of the CONTRACT document

Figure 3: Examples of TCM mechanisms

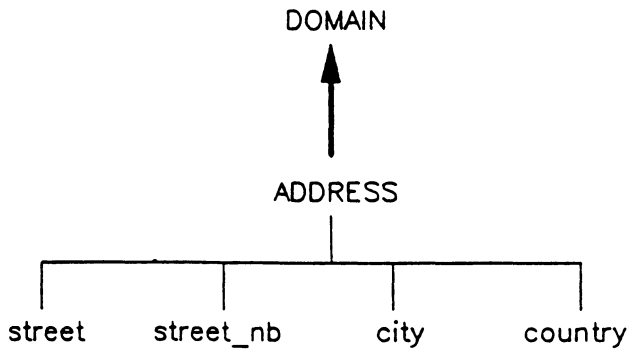


Figure 4: The ADDRESS domain

- two references to the CLIENT and the BANK entities which are mentioned in the contract: "cli" and "bnk".

Compared to the aggregation abstraction form introduced by Smith in SHM, the TCM aggregation mechanism is restricted in that it applies only to the upper level of structure of entities and on properties of entities.

TCM entities are defined over **domains**. Predefined domains are for example integer, string, text, date, image. User-defined domains built through TCM mechanisms (except the reference mechanism) are also allowed. In Fig. 4, the user-defined ADDRESS domain is an aggregation of the street, street_nb, city, and country properties. Usually, domains are not depicted in TCM schemas for reasons of readability.

2.2 Presentation of TCM Concepts and TSL Constructs

Now we present the six predefined TCM entities in more detail. Their specification in TSL and their graphical description are used.

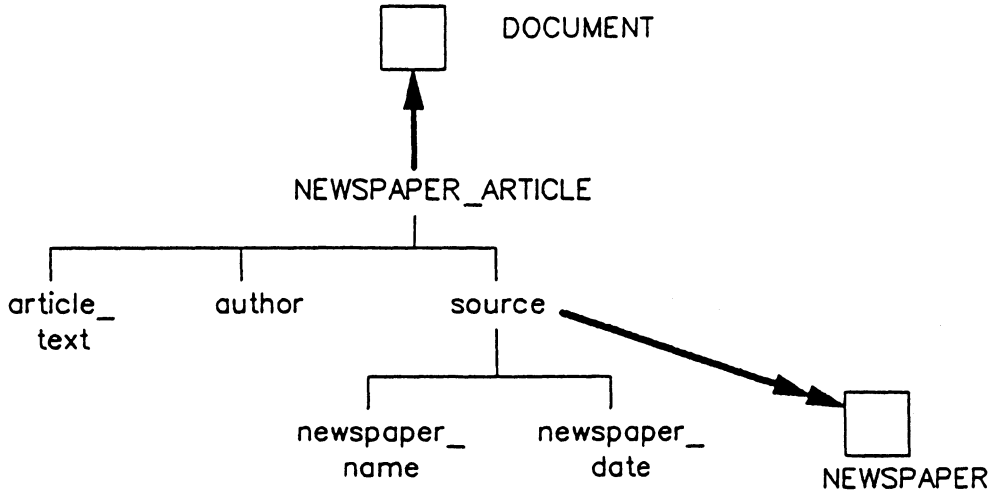
a) The document concept

Definition: A document is a static predefined entity whose instances represent concrete documents (i.e. groups of data which are meaningful as a whole) that exist in the office real world and that will be stored in the OIS.

Examples: Letters, memos, dossiers are typical examples of the document entity.

Fig. 5 shows both the graphical representation (Fig. 5a) and TSL specification (Fig. 5b) of the NEWSPAPER_ARTICLE document, which is an aggregation of a text, an author, and a source.

Fig. 5 shows one type of the reference mechanisms previously introduced. The "view-of" reference shows that "source" shares some information with the NEWSPAPER entity. Moreover, the "with" clause indicates that the source property can be considered as an aggregate of the two properties newspaper_name and newspaper_date. Thus, if the



a) graphical representation

```
<NEWSPAPER_ARTICLE> is-a DOCUMENT;
  { aggregation-of { article_text: image;
                     author: string (20);
                     source: view_of NEWSPAPER
                             with { newspaper_name,
                                    newspaper_date}
                   }
  }
```

b) TSL representation

Figure 5: Specifications of the NEWSPAPER_ARTICLE document

values of the name or the date of a NEWSPAPER instance change, the modifications are reported onto the source property of the corresponding NEWSPAPER_ARTICLE instance.

This mechanism allows the designer to reduce the inherent redundancy of documents. In fact, in the above definition of NEWSPAPER_ARTICLE, we avoid to re-define two properties, newspaper_name and newspaper_date, that are defined elsewhere. However, the semantics of each document is preserved. In our example, for each entity, "newspaper_name" and "newspaper_date" cannot be interpreted separately from the other properties: a NEWSPAPER_ARTICLE is composed of a text, an author *and* a newspaper_name and a newspaper_date.

Aggregation and association can be used iteratively to deal with complex structures of documents: Fig. 6a shows the TSL specification of the SPREADSHEET document, which is composed of a title and cells; cells is a two-dimensional list of individual cells specified using the vector mechanism. The **vector** mechanism expresses an infinite list of ordered elements (in the association mechanism no order is defined on elements). Cells are aggregates of a formula and a value.

In addition, TCM includes three more mechanisms: **identical values**, **parameters** and **cases**.

One **identical value** can be taken by one property for each instance of an entity. For example, the logo of all individual invoices is always the same. The identification of the company (address, phone number, etc.) is the same for any instance of any document produced by the company. Identical values are denoted by the keyword "values" in TSL (see Fig. 6b).

A **parameter** of a textual property is a property which belongs to the text, and which must be distinguished from the text. For example, Fig. 6c shows the ADVERTIZEMENT_LETTER document, where the property "cli" is contained in and is a parameter of the contents of the document (the "cont" property).

At each level of the document hierarchical structure, one property can be decomposed into several exclusive ways, called **cases**. For example, in a curriculum vitae, the professional profile property of an applicant can be described as a list of jobs or as textual sentences.

All these mechanisms can be combined to describe more complex documents, as in Fig. 6b. The LETTER_OF_REFUSAL document contains a property name and contents. "contents" is a text that includes a variable part (parameter-of): the value of the parameter is the client name (denoted by a "$" character).

b) The object concept

Definition: An object is a static predefined entity whose instances represent abstract information stored in the OIS and needed by office agents or by the OIS itself to perform office work.

Examples: Loans, entity states (of progress of projects, for example), marks, keywords can be represented as objects. Objects can also be used to represent information which is useful to manipulate documents (for example for retrieving them) but which is not contained in the documents.

Figures 7a and 7b show the graphical representation and the TSL specifications of the MEETING object. This entity represents a repository which contains information

```
<SPREADSHEET> is-a DOCUMENT;
  { aggregation-of
     { title : string (15);
       cells : vecor [*,*] of cell : aggregation-of
         { formula : text;
           value : string (*)
         }
     }
  }
```

a) using aggregation and vector

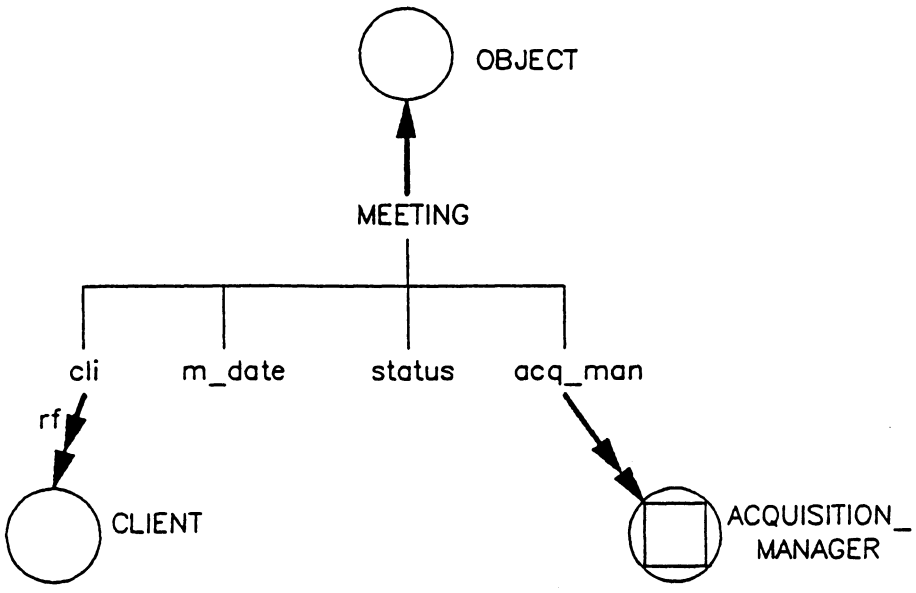
```
<LETTER_OF_REFUSAL> is-a DOCUMENT;
  { aggregation-of
      { content : text;
        parameter-of : aggregation-of
          { content : aggregation-of
              { cli : view-of CLIENT with {name}}
          };
        values : aggregation-of
          { content : { "To client", $cli.name,
                        " : your credit request has not been
                          approved"}
          }
      }
  }
```

b) combining aggregation, parameters, reference and values mechanisms

```
<ADVERTIZEMENT_LETTER> is-a DOCUMENT;
  { aggregation-of
      { cont : text;
        parameter-of : aggregation-of
          { cont : aggregation-of
              { cli : view-of CLIENT with { name }}
          }
      }
  }
```

c) the parameter mechanism

Figure 6: Using different mechanisms for structuring documents



a) Graphical representation

```
<MEETING> is-a OBJECT;
  { aggregation-of
      { cli : ref-to CLIENT;
        m_date : date;
        status : string (*);
        acq_man : ref-to ACQUISITION_MANAGER
      }
  }
```

b) TSL representation

Figure 7: Graphical and TSL representations of MEETING

about meetings between a client and the acquisition manager. It illustrates that all the structuring mechanisms introduced for document entities can be used for object entities.

It is possible to use a "view-of" and a "ref-to" reference for objects, as for example in Fig. 7.

By describing the MEETING object, we are interested in modeling the relationship between clients and acquisition managers. Clients can meet several managers, and acquisition managers can meet several clients (MEETING represents a "n-m" relationship between CLIENT and ACQUISITION_MANAGER).

The "same-as" and "copy-of" mechanisms can also be used in the specification of objects.

c) The agent concept

Definition: Agent entities represent real office agents (individual persons, groups of persons, computers, machines) which interact with the OIS by initiating office activities or by receiving communications from the external world, and responsible for a set of office tasks. Descriptions of agent instances are stored in the OIS.

Examples: Examples of agents are receivers or senders of messages, owners of documents, the secretarial staff, the office manager.

An example of agent specification in TSL is given in Fig. 8. The SECRETARY agent is a specialization of the CREDIT_DEPT_AGENT entity and is defined by the type of task (typing or phone operator) and by the office number.

All the structuring mechanisms and all the reference constructs apply to agent specifications.

Generalization: Fig. 8 shows that any SECRETARY agent is also a CREDIT_DEPT_AGENT agent, and is thus defined by a name, an address and a hiring status.

It is also possible to restrict the generalization mechanism. By using the following specification:

<SECRETARY> is-a CREDIT_DEPT_AGENT where @status="hired"@;

we limit the SECRETARY entity to represent only the permanently hired secretaries.

Multiple subtyping can be expressed as follows:

<SECRETARY> is-a
CREDIT_DEPT_AGENT where @status="hired" union EMPLOYEE;

Here, we define a secretary as a permanently hired agent who is also an employee. In this case, SECRETARY inherits properties from both CREDIT_DEPT_AGENT and EMPLOYEE, and is therefore characterized by a name, an address, a status, a task, an office number, and by the EMPLOYEE properties (salary, phone number, etc.). Multiple subtyping must meet a set of consistency rules, like: "EMPLOYEE and CREDIT_DEPT_AGENT must not have the same property name at the same level of their structure"; if this rule is not satisfied, two identical property names exist in the definition of SECRETARY, which leads to an ambiguity. Other checking rules exist on the model structures.

In TCM, predefined roles are assigned to agents, in particular the "message sender", "message receiver" and "activity responsible" are defined. Agents' roles are known by the OIS; this associates personal data and duties to agents, thus being able to treat

```
<CREDIT_DEPT_AGENT> is-a AGENT;
  { aggregation-of
      { name : string (20);
        address : string (250);
        status : ("hired", "temporary")
      }
  }


<SECRETARY> is-a CREDIT_DEPT_AGENT;
   { aggregation-of
       { task : ("writing", "typing", "operator");
         office_nb : integer
       }
   }
```

Figure 8: Specification of agents

situations like the absence of an agent (messages directed to him are to be automatically forwarded to a substitute agent).

The notion of which agent interacts with the OIS and how, is particularly meaningful to the TODOS architecture selection phase because this affects the software and hardware endowment of each access point to the OIS.

d) The message concept

Definition: Message entities model office communications from and to the external world; they describe the OIS interface to the other functions of the enterprise which the OIS is being designed for.

A message instance is valid only during its transmission and its acquisition by the receiver agent. If the contents of the message has to be kept, the designer has to explicitly model an action of copying the message contents into a document or an object.

Examples: Phone calls, oral messages, electronic mail messages are typically represented as messages. In Fig. 9, the TSL specification of the ARRANGE_MEETING message is represented.

Structure of messages: TCM messages are composed of two parts:

- the **envelope** contains default transmission data: the sender, and the receiver.
- **contents** property defines the information contents carried by the message. The contents property has a structure which is analogous to the document structure.

In Fig. 9, each message instance is sent by a secretary to the OIS (represented by the SYSTEM agent). Messages are automatically forwarded by the OIS to every receiver agent (in Fig. 9, the clerical worker).

"same-as" or a "copy-of" can be used in message specifications. For example, if the contents of a message includes the following property definition:

applicant: same-as PERSON with { name };

```
<ARRANGE_MEETING> is-a message;
  { aggregation-of
      { envelope : aggregation-of
          { from : SYSTEM;
            to : CLERICAL_WORKER
          };
        contents : aggregation-of
          { cont : text;
            parameter-of : aggregation-of
              { cont : aggregation-of
                  { client : }
              };
            values : aggregation-of
              { cont : {"A meeting has to be arranged for
                        client $cli.name, who has shown
                        interest in the credit offer"}
              }
          }
      }
  }
```

Figure 9: TSL specification of a message

every message instance will be taken into account by the OIS in the case where the name of the applicant in the message already exists as a PERSON name in the OIS.

By using a "copy-of" reference, the designer specifies that two entities share the same structure (and not the same values as for the other types of references).

Generalization: Message generalization is identical to other static entities generalizations.

e) The action concept

Definition: An action is a predefined dynamic entity that represents office activities to be automated. Actions modify the states of static entities and are triggered by events under some conditions.

Examples: Creation, modification or deletion of static entity instances, such as "create" the CLIENT object, "copy" the LETTER document, "send" the M1 message, are examples of actions.

Basic assumption: TCM actions model elementary activities of the office: an action entity is a set of action instances which act upon **one** entity and change the state of the instances of this entity.

This assumption, by forcing the designer to describe in detail the effect of the office procedures on static entities, leads to several advantages in the TODOS conceptual design method:

- *minimizing redundancy*: two office activities which act upon the same entity in a similar way are represented by one action in the OIS.
- *enhancing consistency*: due to the level of detail of actions, synchronization and correct concurrency are better ensured because errors such as action pairs that modify the same entity simultaneously are easily detected.

```
<CLIENT_CREATION> is-a action;
   { aggregation-of
      { values : aggregation-of
          { comments : "creation of the CLIENT object";
            act-entity : CLIENT;
            steps : RT ($c, " searching for the last created
                              client");
                    CR ($cl, "creation of the new client",
                        $cl.client_no = $c.client_no + 1;
                        $cl.address = $cont.client.address;
                        $cl.phone = $cont.client.phone;
                        $cl.name = $cont.client.name;
                        $cl.status = "interested")
          };
        in : aggregation-of { cont : ref-to CLIENT };
        var : aggregation-of { c : ref-to CLIENT };
        out : aggregation-of { cl : ref-to CLIENT };
        date_cr : date;
        cli : ref-to CLIENT
      }
   }
```

Figure 10: Specification of an action

- *guiding* the conceptual design: the designer models static entities and, for each of them, checks how the various properties are modified by associated actions.

Structure of actions: Actions contain **structural properties**, which are specific to the action concept, and **descriptive** properties. We illustrate these two kinds of properties using the example of Fig. 10 that shows the TSL specification of the CLIENT_CREATION action (creation of a new instance - denoted by "$cl" - of the CLIENT object).

Structural properties: The action structural properties are specified in the **values** part of an action. They are fixed for all the action instances. The values part contains:

- the **"comments"** property: information useful for the designer;
- the **"act-entity"** property: name of the static entity acted upon (the CLIENT object in Fig. 10);
- the **"steps"** property: it procedurally indicates the execution of actions. The steps are expressed in a programming- language like language with primitives and control operators. The primitives are the following (each acts upon instances):

  CR: creation of static entities

  TX: transmission of messages

  PR: printing of documents

  RM: removal of static entities

  MO: modification of values of static entities

  CP: copy of (parts of) documents

RF: creation of a reference relationship between two instances

RT: retrieval of OIS information

CO: computation of values

ED: interactive editing of documents

Primitives can be combined by using control operators: sequence (denoted by ";"), parallelism ("&"), block (grouping of primitives by means of parenthesis), condition ("IF ... THEN ... ELSE ...") and iteration ("FOR ... DO ...").

Primitives are often associated to free-text comments used to informally describe the behavior of the primitive. For example, when creating an instance by using the "CR" primitive, the designer does not indicate the values taken by the properties of the new instance: these can be expressed through comments which are not interpreted by C-TODOS. Comments are useful also for the other phases of the TODOS method because they indicate useful details such as constraints on the execution of an action. For example, the constraint that printing must occur on a quality printer located near agent A1 is expressed in a comment useful for the architecture selection phase.

In the example of Fig. 10, the steps part indicates that the action retrieves the last instance of client ($c) - to get its client number - then creates the instance of the current client ($cl), and assigns it a progressive number and information about the client (address and phone) which is passed onto the action through the input parameter ("in" part in Fig. 10).

- the **"in"** and **"out"** properties: these properties contain respectively the input and output parameters of the action.

  Input parameters are values needed for the execution of the action. In general, one of the input parameters is a reference to the static entity that changed its status thereby triggering the action (the corresponding value is passed through the triggering event, as we will see later on).

  The output parameters are values produced by the action, and precisely a reference to the static entity acted upon ("act-entity" part).

  In the **"var"** part information which is created and used during the action execution is contained.

  In Fig. 10, the input parameter contains the value of the LETTER_MSG instance received by the OIS; the var part contains the variable "c" used for retrieval; the output part is the new instance of CLIENT.

Descriptive properties: These are optional properties used to keep track of the execution of an action.

In the example, the "date_cr" and "cli" descriptive properties denote respectively the date of the action execution and the CLIENT referred to in the contract.

f) The event concept

Definition: The event is the predefined dynamic concept that represents types of state changes in the office that trigger activities of the same type.

In the definition of event we make the assumption that event instances of the same type recognize state changes of instances of the *same* static entity.

Examples: The end of the week is modeled as an event which, for example, triggers message sendings. Another event is the arrival of letters which triggers the action of copying its text into a TCM document.

These examples show that there are different types of events in the office. Coherently, TCM has three kinds of events, or **event subtypes**:

- **temporal events** describe time-related facts, either an absolute reference (e.g., November 25, 1989), or a periodic reference (e.g., the 13^{th} day of each month), or a reference to another event (e.g., 3 days after the arrival of a new contract).
- **external events** describe the arrival of messages from the world to the OIS (e.g. a letter arrival, a new contract arrival).
- **internal events** describe state changes of the OIS, and are caused by an action execution. We classify them as follows: creation event (which is induced by a creation action), modification event (e.g. induced by an address modification), removal event, completion event (which recognizes that a document instance is complete according to its structure).

Specification of events: Analogously to actions, the event specifications contain structural and descriptive properties.

Structural properties: They are:

- the name of the **static entity** associated to the event (whose state changes are recognized as the event instances).

  Fig. 11 shows the TSL specification of the external event EV2 which ascertains the arrival of the LETTER_MSG message.

- the name of a **predicate** expressing the formal conditions of the state change (for example, in a temporal periodic event, it indicates the period of occurrence of the event instances). In the example, upon the message arrival, if P1 is true (i.e., if the message contains the needed information about the client), EV2 triggers the CLIENT_CREATION and MO_CLI_L actions.
- a **triggers property** specifying the names of the actions to be triggered when the predicate is true. Actions can be associated to **conditions** and/or **triggering factors** when they are triggered conditionally and/or iteratively by events. The triggering condition is the pre-condition that must hold for the action to be executed. The triggering factor computes a set of values; the associated action is executed for each value of the computed set (i.e. as many times as the number of different values in the set). In the triggers part, all these features can be combined by using the primitives and control operators previously illustrated.

Predicates, conditions and triggering factors are specified as separate entities in TSL. When no predicate and no condition is specified, the event can occur without restrictions; when no triggering factor is associated with an action, the action is executed once.

In the example, EV2 triggers in parallel:

```
<EV2> is-a arrival_event;
   { aggregation-of
       { values : aggregation-of
           { comments : "Arrival of a letter sent by an
                        interested client";
             ev_entity : LETTER_MSG;
             predicate : P1;
             triggers : if C1_F then MO_CLI_L &
                        if not C1_F then CLIENT_CREATION
           };
         date_ev : date;
         cli : ref_to CLIENT
       }
   }
```

Figure 11: TSL specification of an event

- the conditional execution of the CLIENT_CREATION action (it is executed if the client does not yet exist in the OIS);
- the conditional execution of the MO_CLI_L action (modification of the CLIENT action).

Descriptive properties: These are optional properties which keep track of event recognitions. The event specification of the example contains two descriptive properties (the date of the event and the client).

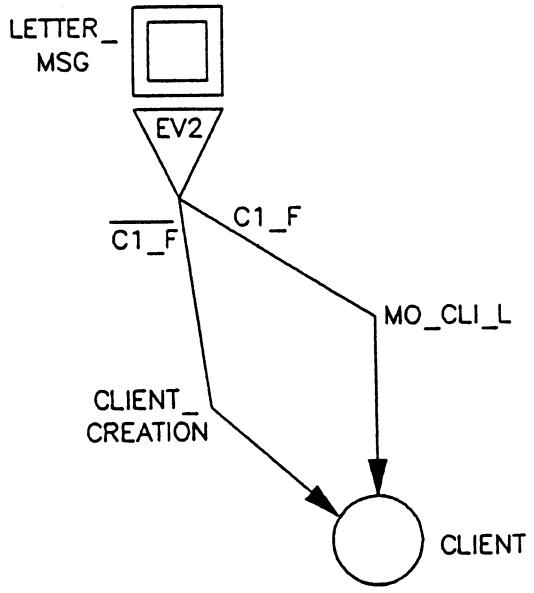
g) The notion of dynamic transition

As we have shown, the behavior of the OIS is modeled through sets of events and actions. The notion of "dynamic transition" combines one event and the actions it triggers in order to give a comprehensive view of the cause-effect relationships that determine the OIS behavior. It is not a basic TCM concept, rather a useful notion derived from TCM concepts.

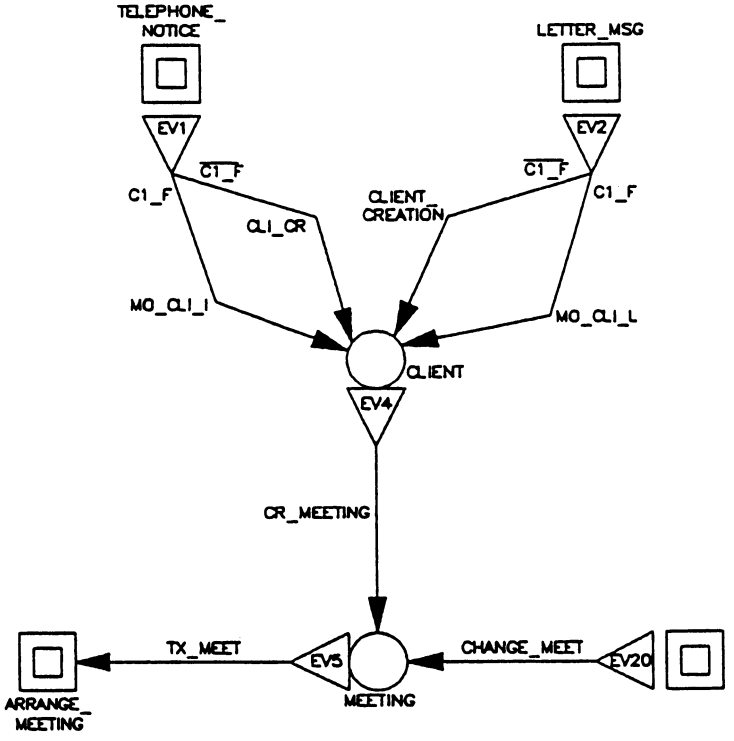
A **dynamic transition** contains one event, its associated static entity, all the actions triggered by the event, and, for each action, the optional condition and triggering factor, and the static entity acted upon.

Fig. 12a illustrates the dynamic transition corresponding to the EV2 event. Notice that the TSL specification of an event corresponds to the description of the dynamic transition (see Fig. 11).

The concept of dynamic transition turns out to be a very relevant one, since it represents the evolution of the OIS from one state to another. The dynamic transition is the OIS **consistency unit**: if the OIS is consistent before a state change is recognized as an event, it should remain consistent after the execution of the dynamic transition associated to the event. Dynamic transitions are executed one at a time.



a) a dynamic transition



b) sequences of dynamic transitions

Figure 12: Dynamic transitions

Dynamic transitions can cause new dynamic transitions to occur. For example, Fig. 12b shows that the dynamic transition associated to the EV4 event (modification of the status of a client, which becomes "interested") follows the transition associated to the EV1 event (telephone call). It can be noticed that such sequences of dynamic transitions are equivalent to sequences of events. In Fig. 12b, for instance, EV4 can follow EV1 or EV2. EV5 (there is a new date of meeting) can follow EV4 or EV20 (modification of a meeting date).

Modeling sequences of dynamic transitions is thus equivalent to *globally modeling the OIS behavior.* The portion of the conceptual schema that contains all the OIS dynamic transitions is called **TCM dynamic schema.** An example of dynamic schema is given in Fig. 12b.

Fig. 12b shows that there are two ways for acting on the CLIENT object (EV1 and EV2). Upon one event, there is a creation or a modification of a client object. The event EV4 is recognized on client creations or modifications; it allows to create meeting information. Meeting changes are modeled through the EV20 dynamic transition relating to event EV20. EV5 models message sendings corresponding to new meetings. Another graph which is frequently used is the event precedence graph (see Fig. 26b) that represents the sequences of events. For instance, in Fig. 26b EV4 follows EV1 or EV2, and EV5 follows EV4 or EV20.

3 Modeling with the C-TODOS Support Tool

C-TODOS is the OIS conceptual modeling tool that assists the office designer in producing a correct and complete **OIS conceptual schema.**

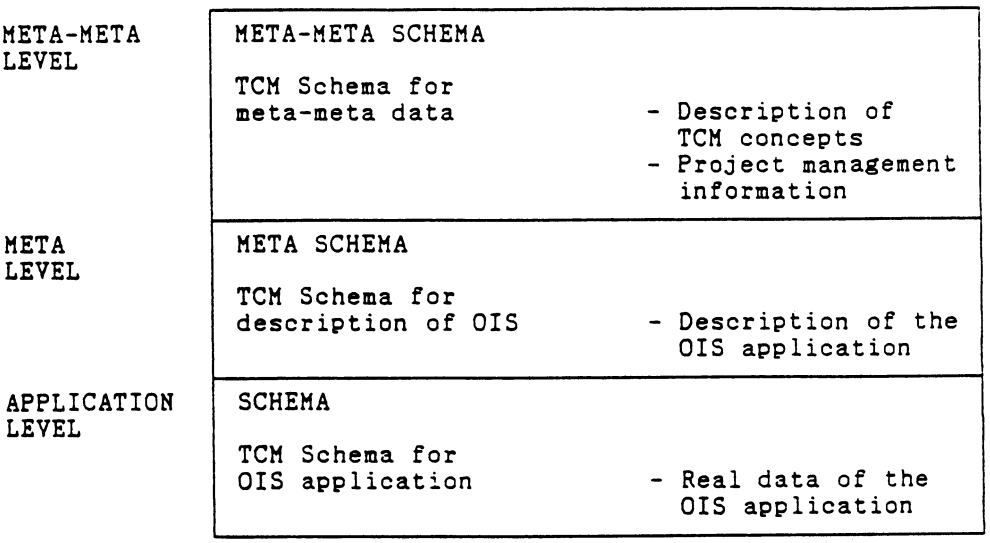
C-TODOS is centered around a **Specification Database (SDB)** that stores the conceptual schema of the OIS application. The conceptual schema, based on the TODOS Conceptual Model presented in Sect. 2, is incrementally produced by the office designer who enters the specifications of the OIS application using the formal TODOS Specification Language of the model.

TCM entities specified by the designer to be part of the schema are automatically mapped by C-TODOS into elements of the database. Support upon **insertion** is given in that the entered specifications are checked by C-TODOS for consistency against TCM concepts and against the already existing specifications.

Other tasks of the conceptual design, such as **modification** and **analysis** of the schema to detect semantic inconsistencies or design errors and to test the quality of the design are supported by C-TODOS by automatically maintaining the consistency of the SDB upon modifications, and by providing the analyst with a query language to analyze the conceptual schema stored in the SDB.

Semantic information about the conceptual model and about the design steps is available to the tool in the form of self- description of the SDB. Using such information, C-TODOS actively assists the designer. For example, the designer can insert specifications into the SDB without any precise precedence order (practically, some rules apply for example to inheritance). Active assistance is provided through a mechanism of **incomplete specifications** handling, that consists of the automatic insertion in the schema of system-defined entities when entities not yet defined are referred by the designer.

Design checks are automatically performed by C-TODOS to enforce the consis-



a) levels of the SDB schema

Figure 13: The Specification Database

tency among entities defined at various moments of the schema production process, with particular attention to the system-defined entities.

C-TODOS provides also support to **project management** functions by maintaining project information while the schema is produced.

In this section we illustrate the SDB (Sect. 3.1) and the management functions performed against the SDB by the modules of C-TODOS (Sect. 3.2). Then, we present some implementation issues (Sect. 3.3).

3.1 The Specification Database

The SDB of C-TODOS is a self-describing database (Mark and Roussopoulos 1986) whose purpose is to store the OIS conceptual schema. The SDB is self-describing in the sense that it incorporates the description of the conceptual model; such description is used by C-TODOS to actively participate in the conceptual design process by supporting insertion of correct specifications and by treating incomplete specifications.

The logical architecture of the SDB comprises three levels, as shown in Fig. 13a; Fig. 13b shows the semantic network of the SDB information. Such network contains both TCM concepts and C-TODOS concepts.

The **meta level** of the SDB contains the conceptual schema of the OIS application. The data in this level are the *description* given by the designer of the real data of the OIS application using TCM, and are called **metadata**. Metadata are stored in the SDB according to TCM concepts (metaschema).

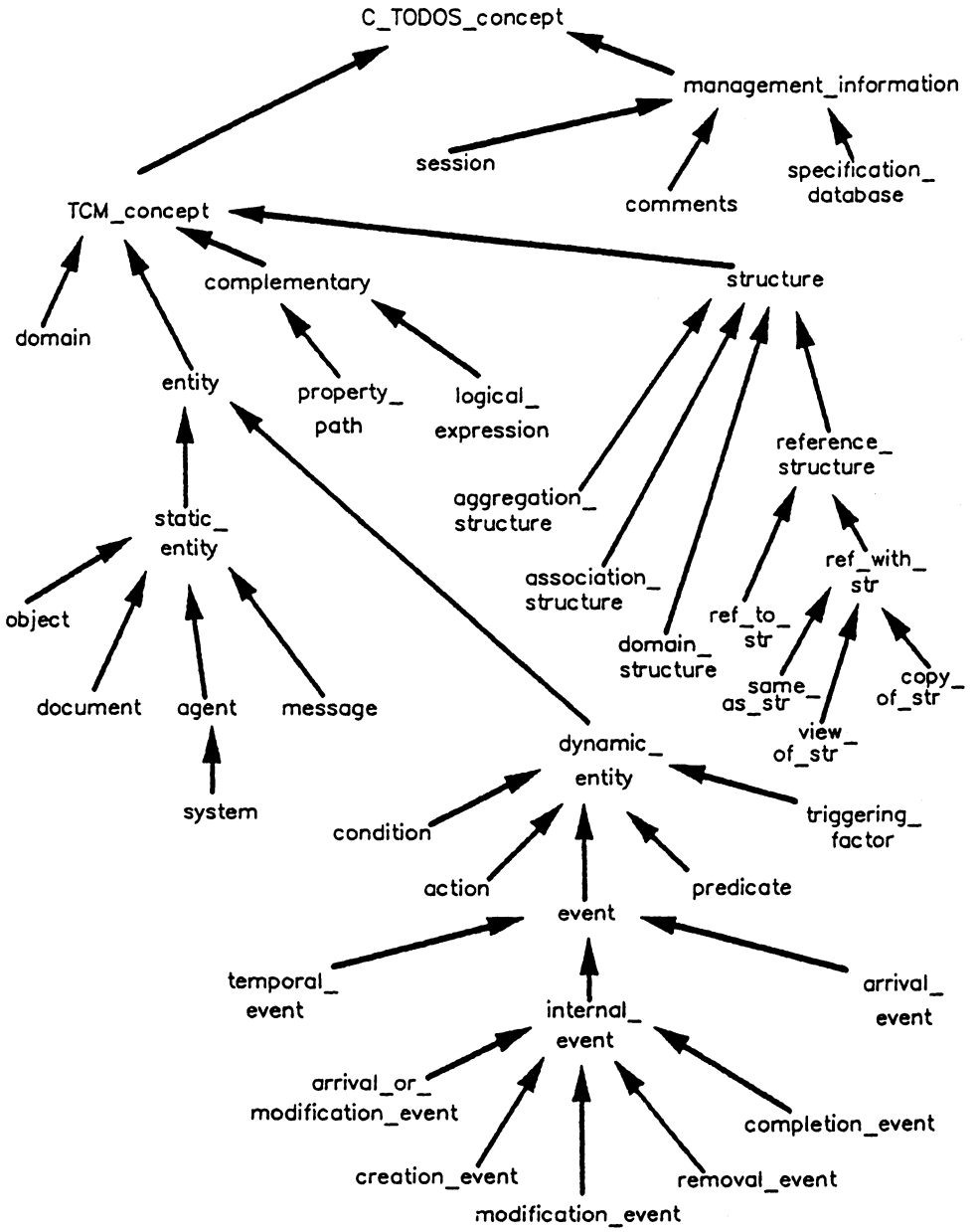


Figure 13 b) semantic network of the SDB information

The **application level** should contain the real data of the OIS application, i.e., the instances of the entities specified in the meta level. Since the SDB is a project database used to store the *description* of the application, this level is empty.

The **meta-meta level** of the SDB provides the database self-description issue. It contains the description of the TCM concepts used to model the application. The meta-meta data describe TCM and use as a reference schema the semantic modelization concepts of TCM (meta-meta schema).

The purpose of meta-meta data is to support correct insertion and consistency maintenance of meta-data. For example, as described in Sect. 2, a TCM entity has a name, an id, a type, a specification text, belongs to an is-a hierarchy and has a collection of properties. This structure is described using TCM as depicted in Fig. 14: an entity has a name, a type, a specification text, can be defined or undefined, has an ancestor in the is-a hierarchy, has children entities and so on. The properties of an entity belong to a property-list which has a structure composed as shown in Fig. 14; in the figure, also an example of the structure of the "document" static entity is depicted (an aggregation of author, text and type).

Such description is stored in the meta-meta level of the SDB. When the designer inserts the specification of an entity, C-TODOS checks that the entity structure conforms to the structure described in the meta-meta level.

Additionally, as shown in Fig. 13, meta-meta data comprise information that is not part of the model but is included in C-TODOS, for example the <session> concept. This is **project management information**, such as the duration of a design session, the author of a specification, the date of insertion/deletion of elements, the relational schema of the SDB implementation. These data are used by C-TODOS to provide the designer with automatic design documentation functions.

The SDB is initialized to contain the following information:

- at the meta-meta level, the self-description of the TCM entities;
- at the meta level, the TCM predefined entities, such as document, object, action. These entities are "predefined" in the sense that C-TODOS knows their structure and they can be referenced to by the designer.

3.2 Management of the SDB

The SDB management functions performed by C-TODOS against the SDB are the following:

1. insertion of specifications
2. deletion of specifications
3. consistency checking
4. query execution.

These functions are performed by the modules of C-TODOS shown in Fig. 15 against the SDB metadata using meta-meta data. In the following, we examine in detail these four functions.

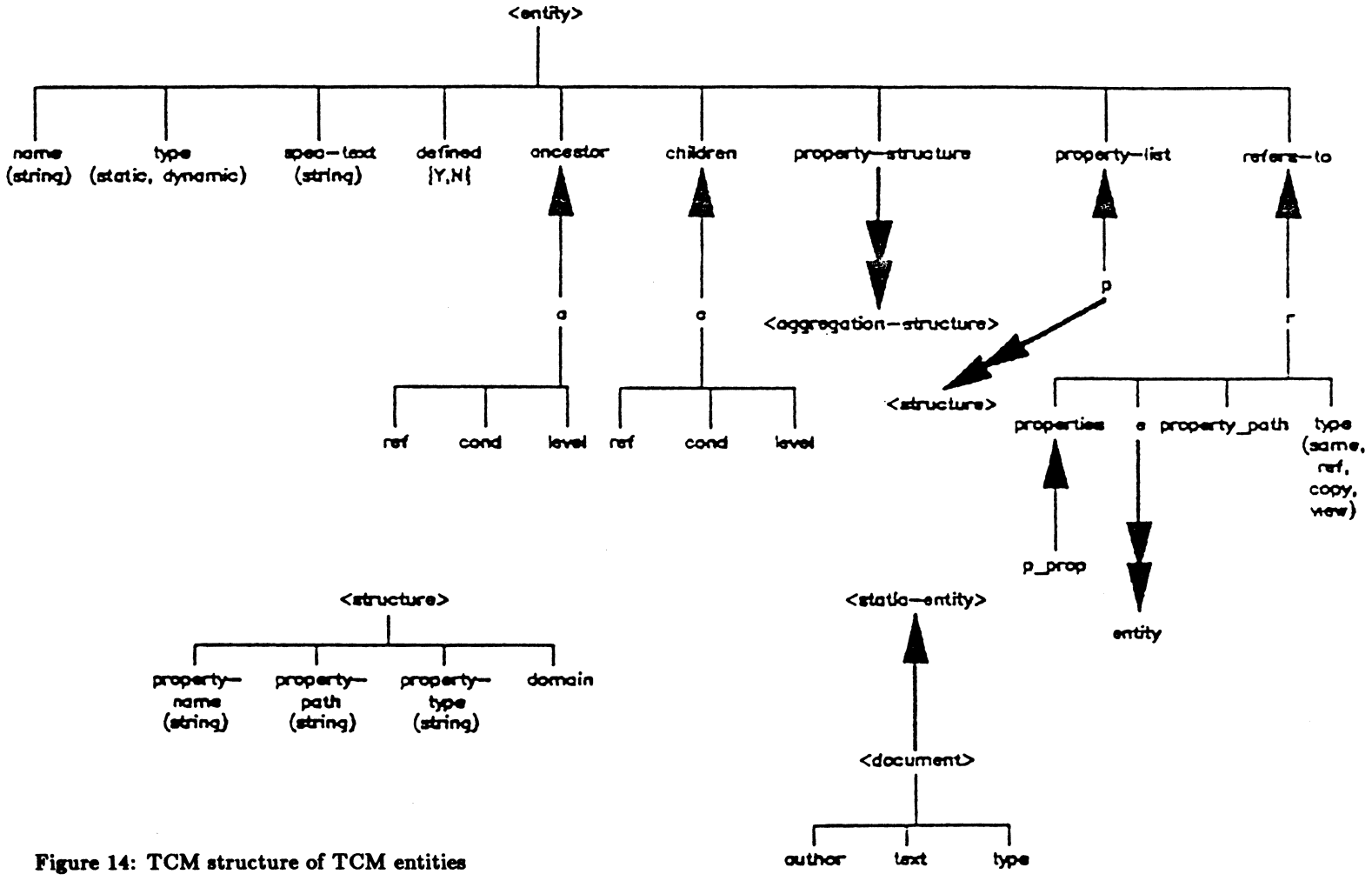


Figure 14: TCM structure of TCM entities

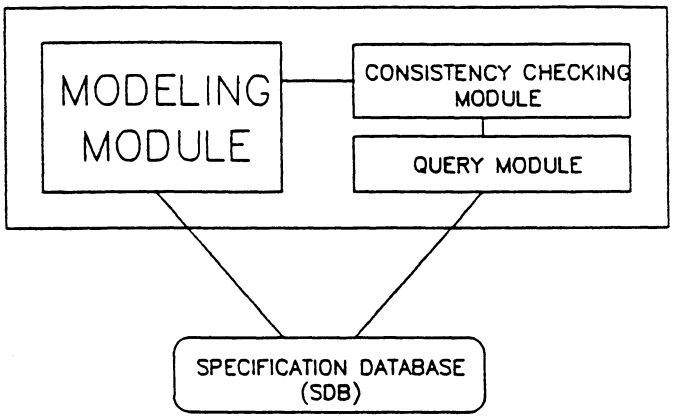


Figure 15: Architecture of C-TODOS

a) Insertion and deletion of metadata: the Modeling Module

The insertion and deletion of specifications into and from the SDB are processed by the **Modeling Module** of C-TODOS that maps the specifications into metadata. Modifications of stored specifications are supported as sequences of insertions/deletions.

Interaction with the Modeling Module occurs on the basis of TSL that allows the designer to specify TCM entities in TSL. Based on TSL, different interfaces to the Modeling Module are provided to the user, allowing him to operate in batch mode, via menus, or graphically (the interaction modes are described in detail in Sect. 4).

Before accepting new specifications, the **insertion support** component of the Modeling Module applies syntactic checks and invokes the *Consistency Checking Module* for semantic checks. Afterwards, it generates the appropriate metadata and stores these into the SDB. Syntactic checks executed by the Modeling Module ensure the correctness of TSL specifications according to the TSL grammar, and ensure that some constraints, such as the uniqueness of entity identifiers, are met.

Upon deletion of specifications, the **deletion support** component of the Modeling Module invokes the Consistency Checking Module to check the deletion dependencies involved in the deletion, and then performs the necessary deletions of metadata from the SDB.

The Modeling Module handles **incomplete specifications**, according to the TODOS design method that the OIS can be incrementally specified and refined by the designer. Incomplete specification handling is performed through calls to the Consistency Checking Module and works as follows.

The ancestor of the entity that is currently being defined must be defined in the schema. For the reference mechanisms, C-TODOS allows the referenced entity to be still undefined at the moment of insertion of the entity where the reference is made.

In order to illustrate the basic operations performed by C-TODOS, we consider a fairly complex example; we consider the Product Order office of a company. Let us

suppose that the designer enter the TSL specification of the PRODUCT_ORDER_-LETTER entity shown in Fig. 16. This entity models the answer letter that a product agent of a company (PRODUCT_AGENT) sends back to a client who ordered a product through a letter (CLIENT_LETTER).

C-TODOS requires the ancestor type LETTER to already exist in the SDB as an entity explicitly defined by the designer (for example as a specialization of the predefined TCM DOCUMENT entity).

Assuming that the referenced entities PRODUCT_AGENT and CLIENT have not yet been inserted, C-TODOS them inserts automatically into the SDB the metadata corresponding to two *undefined entities* PRODUCT_AGENT and CLIENT. The TCM structure of these entities is shown in Fig. 16b.

The self-description of TCM stored in SDB the meta-meta level allows C-TODOS to derive the information that PRODUCT_AGENT and CLIENT are static entities. The properties pa-name, pa-address and c-name, c-address are derived from the specifications contained in PRODUCT_ORDER_LETTER and are automatically inserted by C-TODOS; their types are unknown.

The two undefined entities must be defined explicitly by the designer during the design process; upon definition, C-TODOS will enforce consistency between the properties specified by the analyst and those of the undefined entities.

Upon the user actions shown in Tab. 1a, C-TODOS performs the actions summarized in Tab. 1b. The designer first inserts the specification of <product-order- letter> (action 1 in Tab. 1a). The actions of C-TODOS upon this insertion are existence checks (actions 1.1 and 1.2 in Tab. 1b) and insertion of two undefined entities. Finally, C-TODOS records that a check must be executed upon completion of the current design session (action 1.3 in Tab. 1b). Then we suppose that the designer inserts the following TSL specification of <client> (action 2 in Tab. 1a):

```
<client> is-a <person>
    {aggregation-of
        {name: string (20);
         add: aggregation-of
             {street: string (10);
              number: integer;
              city: string (10)
             };
         code: integer 1..100
        }
    }
```

Consequently, C-TODOS performs the actions 2.1 through 2.4 of Tab. 1b; since the consistency check of step 2.3.a is not successful because the properties of <client> do not match, the designer is required to re-enter the specification of <client>.

b) Analysis of conceptual schemas: the Consistency Checking and Query Modules

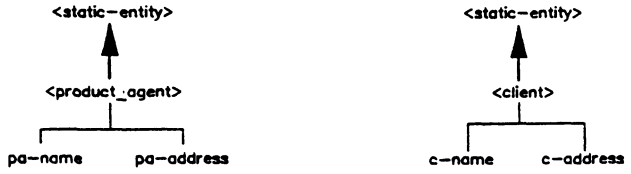
These two modules of C-TODOS allow the designer to validate the OIS conceptual schema stored in the SDB before using it as a basis for the prototyping and architecture

```
<product-order-letter> is-a <letter>;
   { aggregation-of
       { logo : image;
         header : text;
         contents : text;
         parameter-of : aggregation-of
           { head : aggregation-of
               { sender : view-of product_agent
                            with { pa-name, pa-address };
                 receiver : view-of client
                              with { c-name, c-address };
                 current-date : ref-to office-calendar
               };
             contents : aggregation-of
               { d : view-of client-letter
                       with { letter-date};
                 signature : view-of product_agent
                               with { name }
               }
           };
         values aggregation-of
           { header : { "$sender";
                        "$receiver";
                        "$current-date"
                      };
             contents : "Dear Sirs,

                         Thanks for your "$d" letter. We inform
                         you  that  the quantity of product you
                         order  is available from today. Please
                         execute the money transfer.
                         Yours sincerely,
                                              "$signature"

           }
       }
   }
```

a) specification entered by the analyst



b) undefined entities inserted by C-TODOS

Figure 16: TSL specification of an entity referring to entities not yet inserted in the SDB

USER ACTION	ENTITY	SUPERTYPE	REFERENCED ENTITIES	PROPERTIES
1. insert	<product-order -letter>	<letter>	<product_agent> <client>	logo,sender, receiver, d, day,signatur contents
2. insert	<client>	<person>	---	name,add, code

Tab. 1a)

Table 1: User actions and C-TODOS actions upon insertion of metadata

selection phases. They support the analysis of the characteristics of a schema aimed to checking both design errors and the **quality** of the design. For example, the analyst may want to examine the correctness of a document flow in the system, i.e., whether the flow paths designed for the document are correct, do not contain deadlocks, flow between proper agent pairs, and bring the correct information content. C-TODOS supports this analysis by enabling to abstract from one schema all the elements (actions, events, predicates, trigger conditions, agents, etc.) that are involved in the flow of that document.

Much of the quality analysis on the conceptual schema can be performed by the designer using appropriate queries, both predefined and user-defined, using the Query Module. This module is illustrated in more detail in Sect. 4.

The Consistency Checking Module is in charge of executing semantic checks on the conceptual schema using correctness, completeness, and accuracy rules.

- **Correctness rules** ensure that the elements of the schema meet semantic constraints bound to the model structure. For example, TCM dynamic transitions should contain at least one action; the is-a network of the entities defined by the designer must be loop-free.
- **Completeness rules** allow to detect entities missing from the schema. Checks executed by the module to enforce these rules include:
  - *entity existence checks* on ancestors of inserted entities, and on the referred entities according to the mechanism shown in Sect. 3.2.a: if it does not exist, the referred to entity is inserted by C-TODOS as undefined, and must be subsequently explicitly specified;
  - *property existence checks* when a property P_i of entity E_i is referenced by an entity E_j as follows:

C-TODOS ACTIONS ON THE SDB

```
1.1 check existence of supertype:
      <letter>  (must exist as DEFINED)
         if not defined ----> reject specification

1.2 check existence of referenced entities:
     <product_agent>
       <client>

  - if do not exist ----> generate meta-data for referenced
                          entities as UNDEFINED entities:

                        <product_agent> UNDEFINED static-entity
                          properties: pa-name (type unknow)
                                      pa-address (type unknow)

  - if exist (UNDEFINED or defined)
                    ----> check consistency of properties

1.4 add new UNDEFINED entities to list of UNDEFINED to be
    specified by user
             ---o---   ----o---   ---o---   ---o---
CHECKS ON UNDEFINED ENTITY

2.1 check existence of supertype:
      <person> (must exist as DEFINED)
         if not defined ----> reject specification

2.2 check existence of referenced entities:
       NONE

2.3 check entities referring to it:
             <product-order-letter>
                      --> a. check consistency of properties
                              client.c-name ¦ client.name
                              client.c-address ¦ client.add

                              NO NAME COMPATIBILITY-->
                              ERROR TO USER

                          b. if check a. OK:
                             ---> set status of <client> entity
                             to DEFINED

2.4 remove current entity from UNDEFINED
```

Tab. 1b)

$< E_j >$ is-a ...;
{aggregation-of
{...
P_j: view-of E_i with $\{P_i\}$
}
}

P_i must exists.
Moreover, properties defined explicitly must be consistent with those of the undefined entity.

- **Accuracy rules** are the basis for active design support functions provided by C-TODOS. In fact, their application detects design flaws that do not correspond to errors, but that might be inconsistencies, or that might represent undesired features of the target application (for example, because they influence parameters such as the performance of the target system).

  For example, communication paths or document flows can be optimized if accuracy checks show that there exist elements of the schema that are bottlenecks for the activities performed in the application (for example, an object from which too many transitions depend). An other example of accuracy check regards the similarity of entities. The structural equivalence of entities may suggest a redundancy in the schema: the designer can decide to merge two similar specifications, or decide that the redundancy is useful and keep it.

Consistency checks are executed at various moments of the design:

- Upon insertion and deletion of specifications. These checks are executed automatically.
- Upon completion of one design session. These permit to ensure the global consistency of one schema before passing it on to the prototyping phase and to the architecture selection phase (for example, no entity can remain undefined). Typically, accuracy rules are applied upon completion of a session.
- On portions of the schema, or on groups of specifications. These checks can be initiated by the designer at any moment.

Consistency checks are implemented through the TODOS Query Language as queries on the conceptual schema; therefore, the Consistency Checking Module invokes the Query Module (Fig. 15).

3.3 Implementation Issues

C-TODOS has been implemented on a relational database system using the ORACLE DBMS[1]. The system is running on IBM AT-Class computers in the 640 Kbyte RAM/ 4Mbyte Fixed Disk configuration, under MS-DOS 3.11[2].

The relational implementation of the SDB consists of the following basic relations:

[1]ORACLE is a trademark of Oracle Corp.
[2]MS-DOS is a trademark of Microsoft Corp.

1. In the *meta level* of the SDB the ENTITY, the IS-A and STRUCTURE relations store information about the entities of the schema.

   For example, the tuples generated by C-TODOS in the ENTITY relation from the specification of PRODUCT ORDER_LETTER of Sect. 3.2 are shown in Tab. 2a.

   The tuples that are generated in the IS-A relation are shown in Tab. 2b and the tuples generated in the STRUCTURE relation are shown in Tab. 2c.

2. In the *meta-meta level* of the SDB, the relations are divided into three groups: project management relations (such as ENTITY- HISTORY, SESSION-HISTORY, and SESSION-CHECKS-TO-BE-PERFORMED), relations that store the TCM concepts (META-ENTITY, META-IS-A, META-STRUCTURE), and relations that describe the relational schema (RELATIONS, ATTRIBUTES, DOMAINS, DOMAIN-VALUES).

   For example, the tuples generated in the ENTITY-HISTORY relation when PRODUCT_ORDER_LETTER is defined are shown in Tab. 3a (we suppose the current design session is the 11th). The tuples generated in the ENTITY-INFORMATION relation are shown in Tab. 3b.

4 Interfaces of C-TODOS

In this section, we describe the interfaces that C-TODOS provides to its users for inserting, deleting, and inspecting the specifications contained in the SDB.

The insertion of specifications can be made in three different ways (see Fig. 15):

- by using the **TSL batch interface**, the designer can enter specifications written in the TSL language. These specifications are first written using a text editor external to C-TODOS. They are then "compiled" by the TSL batch interface.
- the **TSL menu based interface** allows the designer to insert specifications in a menu-driven fashion. This interface is particularly useful for inexperienced designers who are not familiar with the syntax of TSL.
- the **graphical interface** is an easy-to-use interface which allows the designer to insert specifications by using the graphical notations presented in Sect. 2.

All the specifications inserted in the SDB are checked by the Consistency Checking module. At any moment in time, both the graphical and the TSL notations of a TCM entity are stored in C- TODOS in the Graphical Data Base (GDB) and the SDB respectively (see Fig. 17). This means, for instance, that the TSL representation of an entity is automatically produced by C-TODOS when the graphical representation is inserted.

The deletion of specifications is performed through the deletion module. Modification is supported through deletion and creation of the entity.

Inspecting specifications allows the designer to check the specifications stored in the SDB and to analyze them. This operation is performed through the **query interface**.

In the following, we focus on the graphical interface called G- TODOS (Graphical TODOS): it is illustrated in Sect. 4.1. Then we present in Sect. 4.2 the query interface provided by the Query Module.

ENTITY

entity-id	subschema	entity-name	specification-text
e1	static	document	!TSL-source-doc
e105	static	letter	!TSL-1
e106	static	product-order-letter	!TSL2
e107	static	product_agent	UD
e108	static	client	UD
e109	static	office-calendar	!TSL-source-cal
e110	static	client-letter	!TSL-source_clett

Tab. 2a)

IS-A

child	ancestor	level	condition
e106	e105	1	-
e106	e1	2	-
e105	e1	1	-

Tab. 2b)

prop-id	entity-id	lev0	lev1	prop-name	type	domain	value
p400	e106			logo	simple	image	
p401	e106			header	simple	text	
p402	e106			parameter	aggr		
p403	e106	parameter		header	aggr		
p404	e106	parameter	header	sender	view		e107
p405	e106	parameter	header	receiver	view		e108
p406	e106	parameter	header	current -date	ref		e109
p407	e106			contents	simple	text	
p408	e106			parameter	aggr		
p409	e106	parameter	contents	d	ref		e110
p410	e106	parameter	contents	signature	view		e107

Tab. 2c)

Table 2: SDB relations in the meta level

ENTITY-HISTORY

entity-name	entity-id	insertion-session	deletion-session
product-order -letter	e106	11	
product_agent	e107	11	
client	e108	11	

3a)

ENTITY-INFORMATION

entity-id	definition-state	author	comments
e106	YES	analyst-id	"answer to client"
e107	NO	analyst-id	"..."
e108	NO	analyst-id	"client will be a <person> object"

3b)

Table 3: SDB relations in the meta-meta level

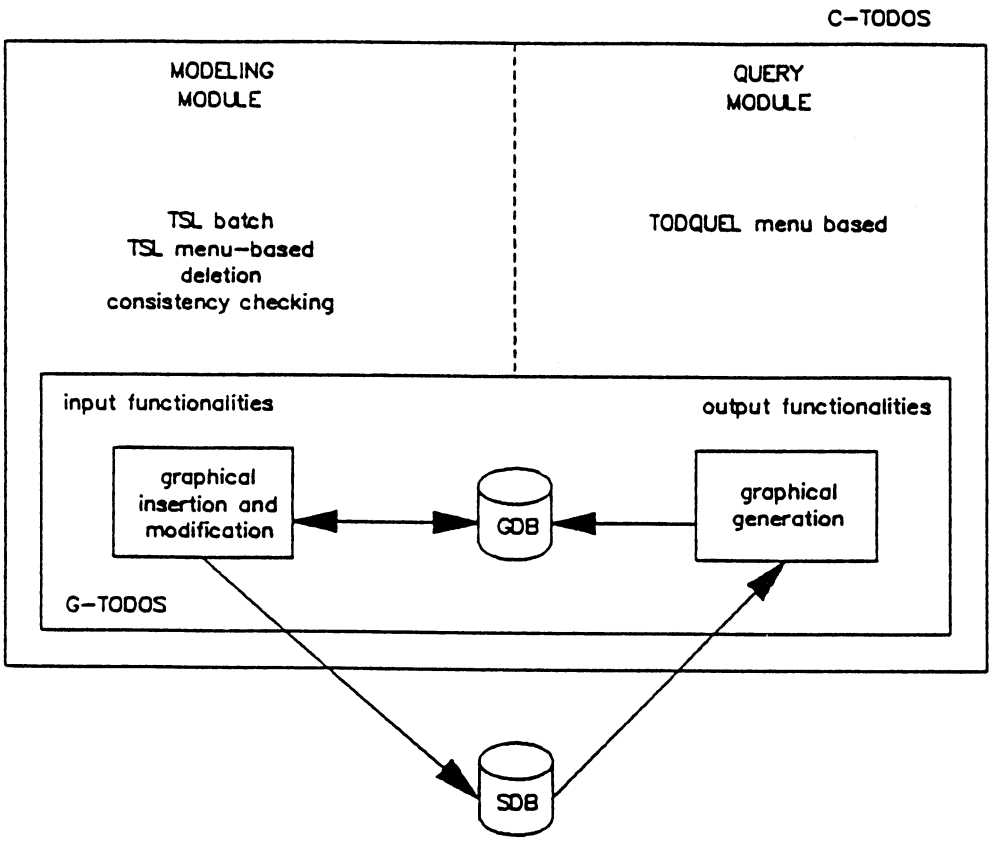


Figure 17: G-TODOS within C-TODOS architecture

4.1 G-TODOS: the C-TODOS Graphical Interface

G-TODOS is the graphical component of C-TODOS. It helps the designer in manipulating the office conceptual schema by providing an easy-to-use user interface based on graphical representations. These graphical representations give a global, synthetic view of all the entities of the conceptual schema; G- TODOS helps the designer in drawing the OIS schema and illustrating parts of it.

In the following of this section, we present the main features of G-TODOS and its functionalities for building and showing OIS schemas. G-TODOS has been developed on an IBM PC-AT. For a detailed presentation see (TODOS TR 2.4.1).

a) G-TODOS main features

The location of G-TODOS within C-TODOS is shown in Fig. 17. G-TODOS provides input and output functionalities. More precisely:

- the *input functionalities* concern the way the designer can graphically insert new entities in the SDB and modify existing ones through their structural representation. The G-TODOS submodule that handles the input included in (and is consistent with) the C-TODOS Modeling Module.
- the *output functionalities* concern two aspects:
  - G-TODOS can automatically produce three entity diagrams which correspond to synthetical information which is deduced from the content of the SDB.
  - G-TODOS allows the designer to inquiry the SDB using the C-TODOS Query Module, and to display the pictorial representation of the result on the screen. The corresponding G-TODOS submodule belongs to the C-TODOS Query Module.

Any diagram which is used in G-TODOS (either in input or in output) is stored by G-TODOS in a repository which is called the **Graphical Data Base** (GDB). This feature allows one to retrieve quickly any diagram in the state where it was created by the designer or produced by the tool. This improves the capability of the designer to analyze the OIS conceptual schema.

For all these functionalities, G-TODOS provides a homogeneous graphical interface for all manipulations. This user-friendly interface is based on windows and icons for displaying information, and on pop-up menus, keyboard and mouse for entering data. It provides graphical facilities for changing the drawings of graphical representations. G-TODOS also allows the designer to simultaneously see on the screen several views of the contents of the GDB by using a multi-windowing technique.

In the following, we analyze the input and output functionalities of G-TODOS.

b) Graphical insertion

The goal of the G-TODOS insertion (and modification) module is to allow the designer to interactively build the graphical representation of an entity (or to modify it). In addition to the general G-TODOS interface, several facilities are provided by the graphical editor for achieving this goal:

- the designer is not bound to a too-strict sequence of manipulations for drawing the diagram,
- the editor uses its knowledge of the TCM model semantics for executing automatic manipulations and avoiding errors,
- G-TODOS ensures consistency between the GDB and the SDB.

These three facilities are now considered separately.

A flexible interface: Such an interface is characteristic of graphic interactive tools and is basic to enhancing the man- machine communications of G-TODOS. For example, using the mouse, the designer can create the new entity MEETING by selecting and executing menu lines of the "CREATE" pop-up menu (see Fig. 19a).

Then, in order to build the diagram pictured on Fig. 18b, the designer can start drawing either the 'is-a' or the reference link.

The designer is not limited by the size of the screen: he can use the scroll arrows to examine the whole diagram.

Knowledge of TCM semantics: This knowledge enables G-TODOS to perform automatic manipulations of the specifications. For example, upon movement of the symbol of CLIENT shown in Fig. 19a, G-TODOS rearranges the link between CLIENT and the 'cli' property symbol (see Fig. 19b).

This is performed because 'cli' and CLIENT are related and allowing CLIENT to be linked with no properties would be an error.

The knowledge of TCM also avoids design errors. For example, G- TODOS does not permit the designer to add an 'is-a' link over the 'cli' property: it is forbidden in TCM.

Another effect of the TCM knowledge availability is the use of the **multi-windowing facility**. Suppose the designer wants to see the structure of the CLIENT entity. G-TODOS allows him to display the corresponding diagram through a new window (Fig. 20).

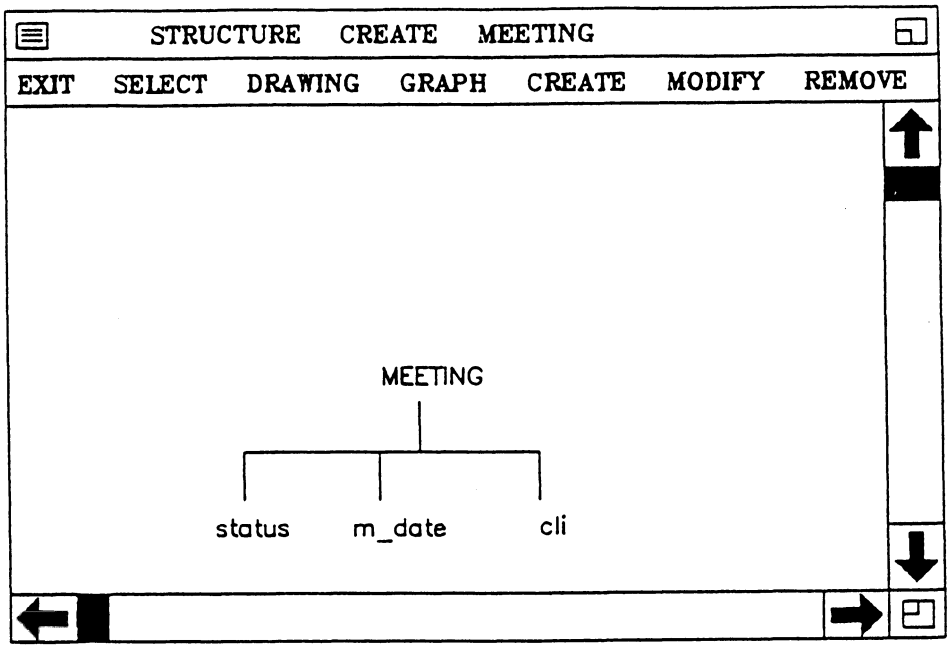
This is feasible because G-TODOS knows that CLIENT is an entity and can therefore manage the correspondence with its graphical structure.

Fig. 20 also shows the is-a hierarchy on the screen. Some other windows (the reference network and the dynamic schema which have been automatically produced) have been opened by the designer but are not displayed on the screen, probably because of lack of place on the screen. It is possible to display each of them at any moment, thus displaying the window in the state where it was before being hidden.

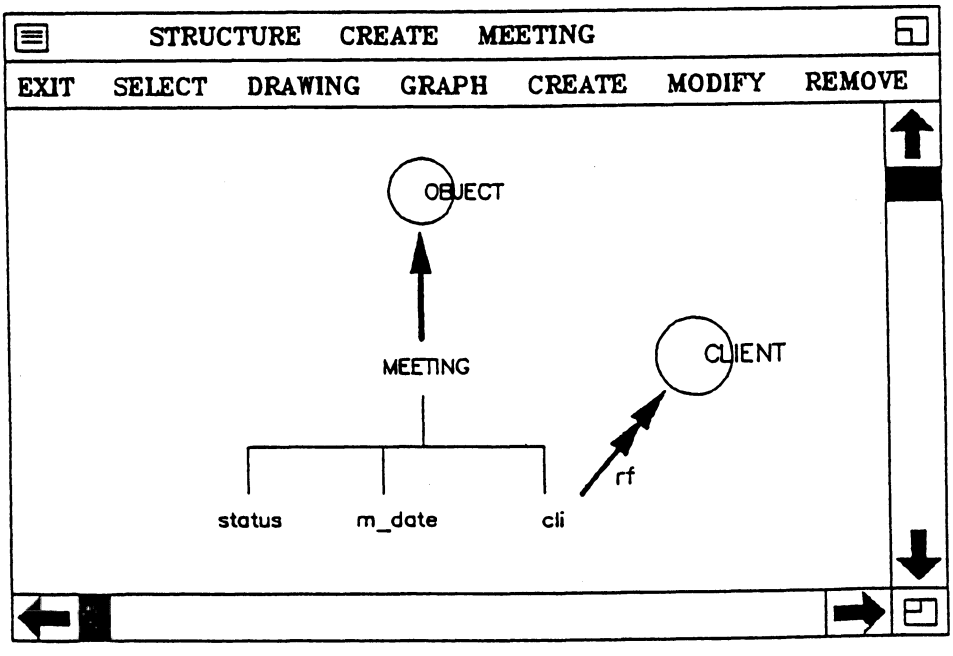
Consistency: G-TODOS is globally coherent with C-TODOS. It performs the same consistency checks that the other input modules perform against the SDB.

Moreover, in order to make the GDB and the SDB consistent, we have chosen to make both the TSL textual and the graphical representations of any entity available at any moment.

This means that the TSL description of an entity is automatically produced by G-TODOS when the designer graphically inserts this entity. The graphical representation is generated when the designer uses the TSL batch or the menu based modules for entering an entity. The graphical representations of entities are not strictly equivalent to

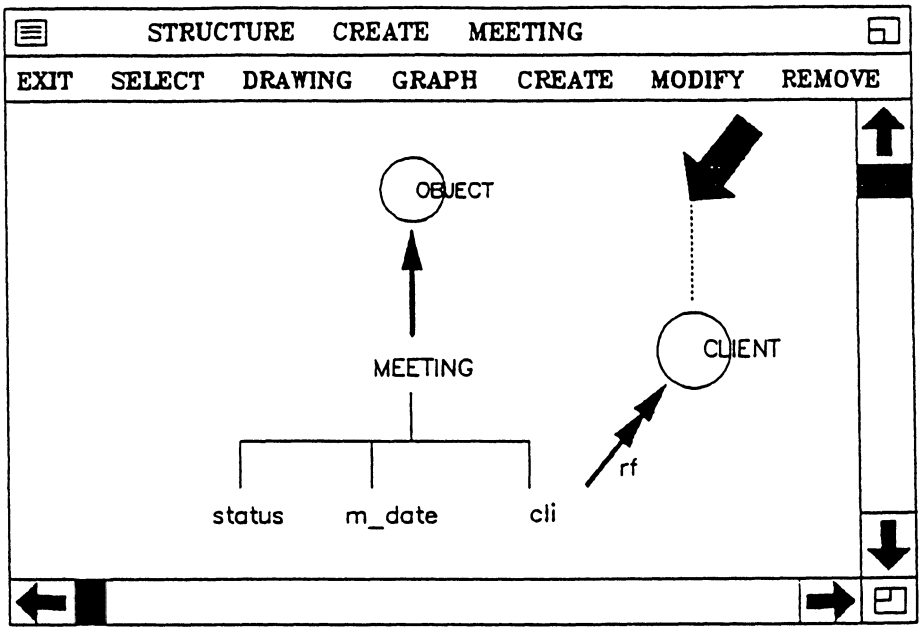


a)

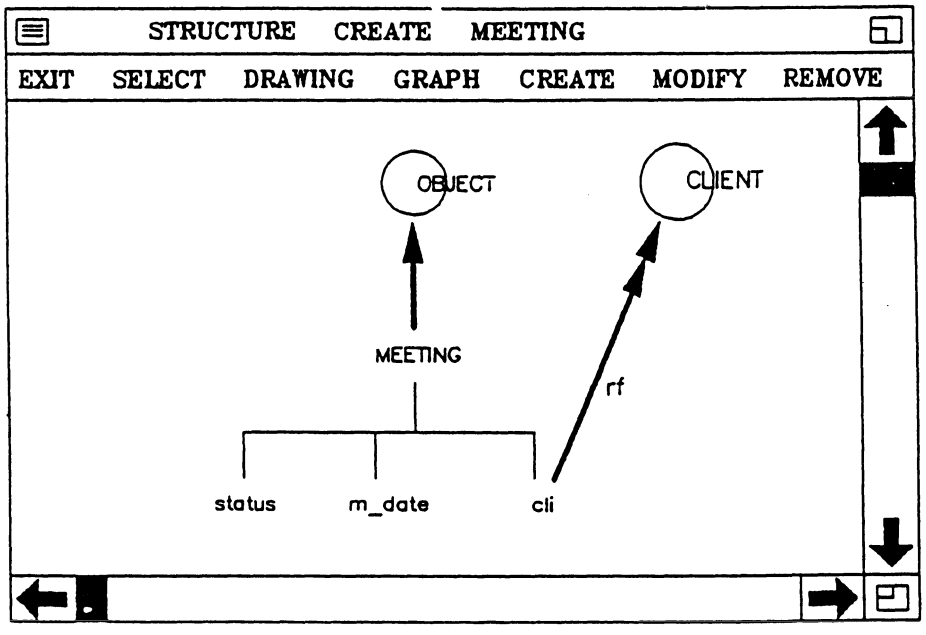


b)

Figure 18: Sequences of drawings in the "CREATE" pop-up menu



a) before the movement



b) after the movement

Figure 19: Rearrangement of links between entities

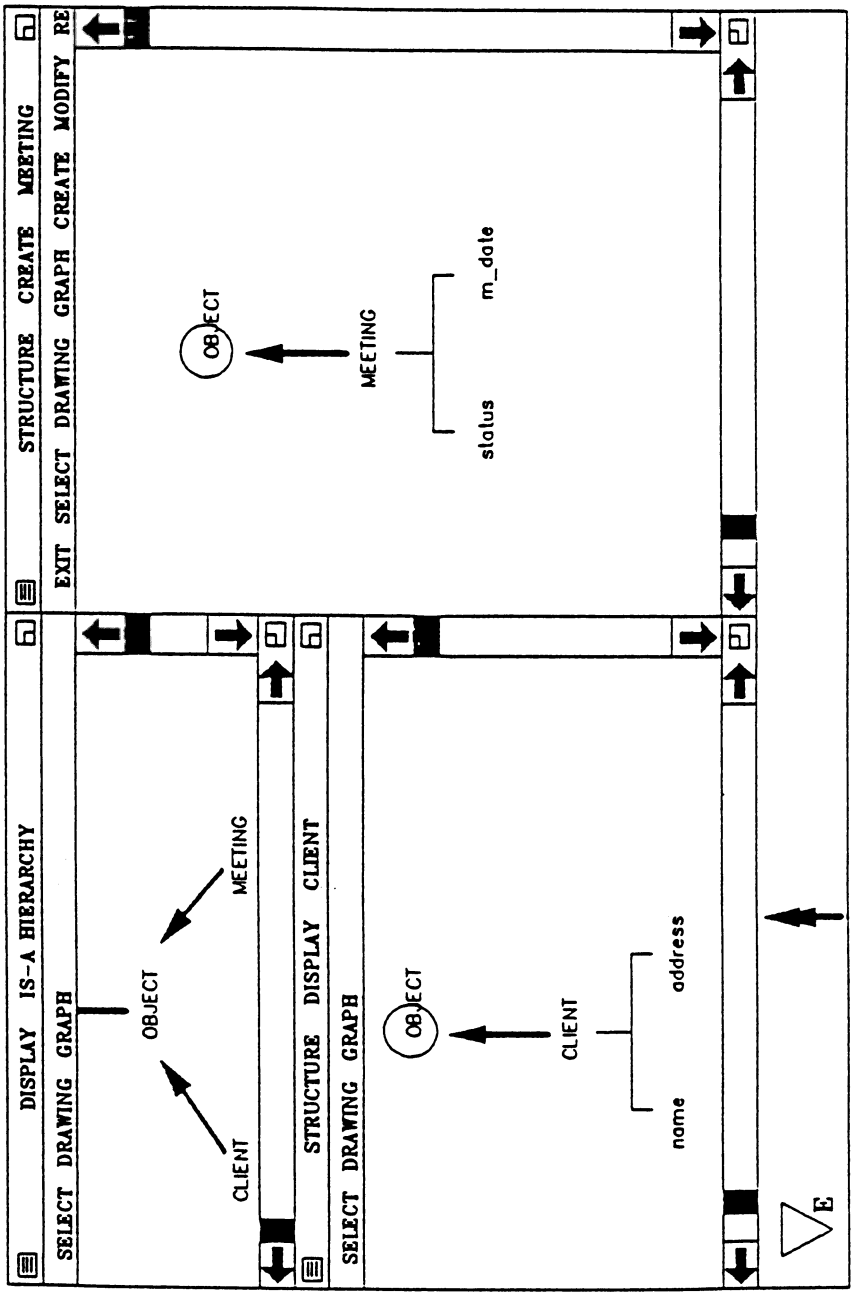


Figure 20: The multi-windowing facilities

the TSL specifications, because, as we mentioned in the previous sections, the graphical representation does not contain detailed information about domains. For example, in Fig. 21, the domains of the properties appear only in the TSL specification. Consequently, G-TODOS forces the designer to add the missing parts of the specifications. Upon termination of the insertion process, both the graphical representation and the missing parts are stored in the GDB.

c) Using G-TODOS for graphical analysis

G-TODOS performs automatic production of graphical representations: G-TODOS produces three entity diagrams which are derived from the actual contents of the SDB:

- the Static-Reference Network (SRN: see Fig. 25), which represents all the entities which are connected by a reference link;
- the is-a Hierarchy (IH: see Fig. 13b), which depicts all the entities and their generalization relationships,
- the Dynamic Schema (DYN: see Fig. 12b), as presented in Sect. 2.

The production of graphical representations creates a version of a diagram from the state of the SDB at any moment when the designer activates this process. The process is not interactive and is highly time consuming.

The generation algorithms place the graphic symbols at some appropriate locations, and draw arcs between them. Some aesthetic criteria are taken into account. For instance, in the Dynamic Schema, two layout criteria are the following:

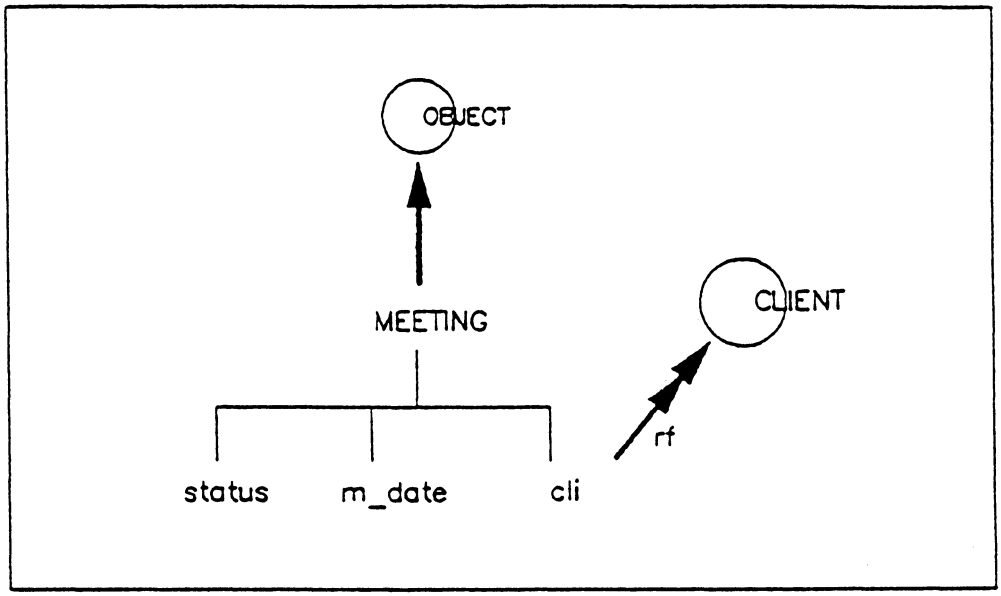
- the message symbols must be placed on the border of the picture, without any line which is drawn "behind" them.
- the lines of actions should traverse one another only when necessary.

G-TODOS allows the designer to modify the layout of the produced diagram, in order to let him place and arrange the graphical symbols to his taste. The produced diagrams can also be displayed on the screen at any moment. For this facility, the designer can use all the functionalities of G-TODOS interface such as multi- windowing and symbol movement. In particular, each diagram can be printed at any moment, which allows the designer to maintain the documentation of the conceptual design phase.

4.2 Querying the SDB

Queries to retrieve information about the OIS conceptual schema stored in the SDB can be performed at several points during a design session, as an aid for the designer to retrieve already defined specifications and to check the completeness and consistency of the OIS schema being created: when elaborating new specifications, inserting specifications, at the end of a design session.

While defining elements of the OIS schema, the designer may be interested in knowing which element types have already been defined. It may be interesting to aggregate office conceptual elements according to different points of view: list actions performed




```
<MEETING> is-a object;
   { aggregation-of
      { status : text;
        m_date : date;
        cli : ref-to CLIENT
      }
   }
```

Figure 21: Correspondence between graphical and TSL specifications

by a given agent, list documents with non textual parts, list subtypes of a given document, and so on. A first support to extract such information from the SDB is provided by the graphical interface, through the entity graphs, the is-a graph, the dynamic graph, and the static-reference and the event precedence graphs shown in Fig. 25. However, such graphs provide summary information about the contents of the OIS conceptual schema. The designer, while defining and checking the schema, needs to examine the specifications in detail, focusing on particular aspects.

To allow for retrieval of specific information from the SDB, we use the Query Module of the C-TODOS tool shown in Fig. 15 of Sect. 3.

When defining the Query Module, we examined several design alternatives for its implementation. First, we considered the opportunity of using SQL as a query language, as the Specification Database is realized using a relational DBMS. This possibility was discarded as the queries would be too implementation dependent, and the benefit of having a semantic data model for describing office elements in the conceptual schema would be partially lost. Therefore, we decided to develop a specific querying module for the TODOS conceptual design environment. The problem to be solved was that of being able to extract information useful for the designer in a flexible and easy way. Two solutions have been proposed during the project:

- a semantic query language
- a menu-based interface allowing to ask queries relevant for OIS conceptual design.

As a basis for both solutions, we studied the classes of queries that are important for supporting OIS design and for delivering the results of the conceptual design phase to other design phases in TODOS.

TODQuel (TODOS Query Language) (Pernici 1988) is a query language for semantic models. In TODQuel, the queries are constructed using a semantic description of the TCM model. Each modeling concept in TCM is described using the same concepts of aggregation, association, and classification on which TCM is based. Queries are formulated using this description as a basis. For instance, in Fig. 22, we show a partial description of the semantic schema for entity "entity" in TCM. "entity" has a name and an association of properties, each with a name and a type. If we want to know which entities in the OIS schema have multimedia (image) properties, we use the semantic schema for "entity" in the following way:

- relevant entities are selected, traversing the schema of entity to find at least one property type equal to image;
- the names of the selected entities are retrieved.

TODQuel is based on first-order predicate calculus. We refer the interested reader to (Pernici 1988) for details on TODQuel.

The disadvantage of TODQuel is that query formulation is rather technical and results in a difficult querying interface for conceptual designers, which should be considered as non technical users of the conceptual design tool.

Therefore, we decided to apply the theoretical basis gained with the study of TODQue in the realization of a menu based interface that enables the designer to formulate queries easily and according to possible types of analysis to be done on the conceptual OIS schema.

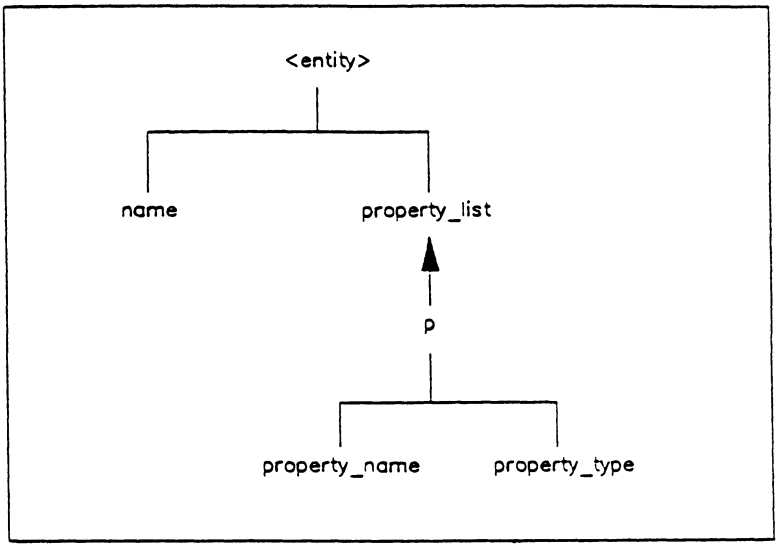


Figure 22: TCM entity structure

In the following, first we present and discuss the types of queries necessary for supporting conceptual design, then the interface realized for the query module.

a) Classification of queries

Queries needed to support conceptual design are of four types:

- structural and static queries
- dynamic queries
- reports
- checks.

In the following, we discuss separately each type of query.

Structural and static queries: With this type of queries, it is possible to retrieve entities specified in the OIS conceptual schema with given characteristics. For instance, entities with a given name, with a given property name, with a given type, and so on. For a careful design, it is necessary to distinguish between inherited and non inherited properties of an entity, for instance for evaluating the impact of a given change in the specification of an entity. This type of queries is principally applied to retrieve characteristics of static entities, such as documents, objects, and messages. It is also useful for retrieving structural properties of dynamic entities: e.g., retrieving the name of the entity associated with a given event type.

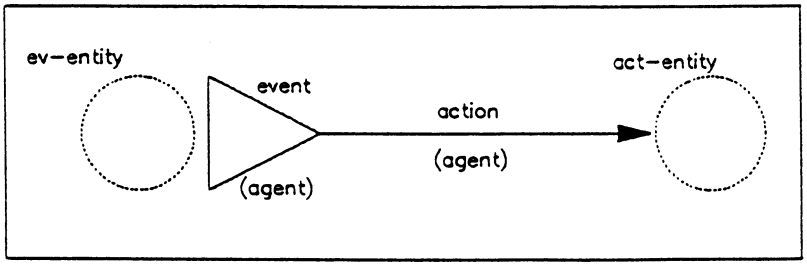


Figure 23: Dynamic transition

Dynamic queries: This class of queries allows to extract information contained in the dynamic subschema, i.e., about the procedures performed in the OIS. Dynamic queries are based on the concept of dynamic transition presented in Fig. 23.

The designer is enabled to retrieve particular information about all the possible relationships between the elements defined in the dynamic schema, following the description of Fig. 23.

For example, he may ask to retrieve all events originated from a given entity (ev-entity). Similarly, all pairs of elements in Fig. 23 can be considered, originating queries such as: retrieve all actions acting on a given entity, all actions triggered by a given event, and so on. It is also possible to extract information about responsible agents for entities and for actions.

Reports: Reports allow to generate summary information. The principal types of reports used in C-TODOS are those used for delivery of the results of conceptual design to the other design phases in TODOS. The **inserted entities** report lists all the specifications of new entities, inserted after last delivered session. Similarly, the **modified entities** report lists all modifications and deletions. These files are used to communicate incremental modifications of the conceptual schema to the prototyping team, which is thus able to modify the office prototype incrementally. Other types of reports are used by the conceptual designer himself. **Session** reports provide information about the history of insertion, modification, and deletion of entities, about the date of sessions and the inserted and modified entities in each session. **Lists** of defined entities can be created by type (e.g., all documents, all messages, all agents, and so on). It is also useful to retrieve **lists of undefined entities.**

- *Checks.* Some checks are performed routinely by the designer to control the consistency and quality of specifications. As discussed in Sect. 3, some consistency checks are automatically performed by the insertion module at design time. The checks presented here are those which cannot be performed automatically, as their result has to be evaluated by the designer. Three types of checks are predefined. Other checks can be performed directly by the designer using the other classes of queries presented above. The predefined checks are the following:

- *static entities used in no action*: those entities that are defined in the specifications, but never accessed by any action are listed; it is the task of the designer to examine these situations and evaluate if they correspond to a design error or to acceptable situations in the system.

- *static entities with no associated event*: this case is similar to the one presented above; a static entity usually has one or more associated events denoting it is used in the office procedures. The lack of associated events, however, does not imply a design error, since some entities may be necessary to store information for other office procedures which are not part of the OIS.

- *overloaded entities*: entities causing more than five events are considered as overloaded, and the designer has to evaluate whether to split the entity in different entities, or if this overloading is correct within the logic of the application.

b) Menu based query module interface

The formulation of queries is based on a menu-guided interface. The classes of queries presented above constitute the top level of the menu. For each class of queries, a sequence of menus is defined. Queries are parametric, thus enabling to define the query parameters using the menu.

In Fig. 24b, we show an example of a simple menu sequence, used to retrieve the list of properties defined for the NEWSPAPER_ARTICLE entity defined in Fig. 24a.

Results of queries can be saved and utilized to combine query results (using the set operations of union and difference) in order to be able to retrieve specifications based on complex conditions.

5 The Modeling and Analysis Method for Conceptual Design

In this section, we describe the method that is at the basis of the TODOS conceptual design phase.

The purpose of this method is to structure the production of the formal TSL specifications in a set of design steps. Partitioning of the design task is a must when large projects are considered (Olle et al. 1982, Ceri 1983).

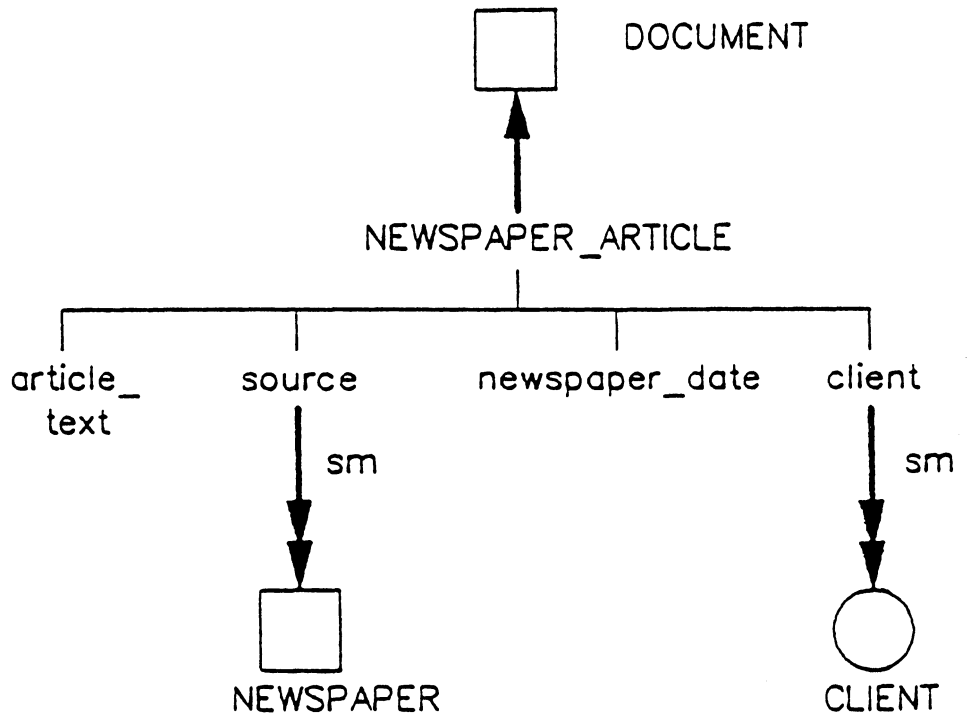
In defining a method, we have to establish:

- which *steps* compose the method;

- which *inputs and outputs* are expected from each step and *when* they should be delivered;

- the *facilities* to support the execution of the steps.

For the TODOS conceptual design phase, the *inputs* are the *functional requirements* collected and analyzed during the first phase of the TODOS design process (see Chapter 2).




```
<newspaper_article> is-a document;
   { aggregation-of
       { article_text : image;
         source : same-as newspaper with ( name};
         newspaper_date : date;
         client : same-as client { with }
       }
   }
```

a) entity graph and TSL specification of NEWSPAPER_ARTICLE

Figure 24: Query on the NEWSPAPER_ARTICLE

Example of query formulation.

"Retrieve all properties of entity <NEWSPAPER_ARTICLE>"

Screen1.

```
------------------------------------------------------------
QUERY START
------------------------------------------------------------

Select with arrows:

list   control  static dynamic

------------------------------------------------------------
PF1   PF2        PF3          PF9
help  visualize  query files  quit
------------------------------------------------------------
```

Screen 2.

```
------------------------------------------------------------
STATIC START
------------------------------------------------------------

Select with arrows:
- properties of entity
- domain of property
- entities with given property
- subtypes of entity
- referred to entities form an entity
- similar entities

------------------------------------------------------------
PF1   PF2        PF9
help  visualize  quit
------------------------------------------------------------
```

Sreen 3.

```
------------------------------------------------------------

Select with arrows:

retrieve  all                  properties
          inherited only
          non inherited only

of an entity.
------------------------------------------------------------
```

Screen 4.

```
------------------------------------------------------------

Retrieve not inherited properties of entity newspaper_article.

article_test
source
newspaper_date
client
------------------------------------------------------------
PF1   PF2        PF3   PF4        PF5          PF6
help  visualize  save  scroll up  scroll down  combine queries
PF9
quit
------------------------------------------------------------
```

b) interaction with the query module

Such requirements contain information about the main documents in the office, the goals of the office and the main activities that are performed in the office to achieve these goals.

The *output of conceptual design* is a conceptual description of the office system, that is, the OIS conceptual schema describing the automated system functions that are able to support the office activities described in the requirements. The envisioned OIS should provide improved ways of producing documents and of executing critical office steps, while treating exceptions and anomalies in office work.

In this section, we describe the methodological steps outlined for the TODOS conceptual design phase (Sect. 5.1). A discussion about the method concludes this section (Sect. 5.2).

5.1 The Method for Conceptual Design

a) Mapping the Descriptive Model into TCM

The specification of an OIS conceptual schema in TODOS proceeds from *weakly-structured requirements* coming from the requirement collection and analysis phase described in Chapter 2, and produces the *formal definition* of the conceptual schema.

The requirement collection and analysis phase models the office activity in such a way that a simplified view of the document flow and the sequence of phases of work performed by human resources is given.

The representation of this view is provided according to the **Descriptive Model** (described in Chapter 2). The concepts of "documents" and "human resources" in this model are compatible with those of documents and agents in TCM. The "phase" concept in the Descriptive Model describes a set of actions and its connection with other phases. These connections can be directly mapped into TCM events by the designer, thus obtaining a first version of a TCM schema.

Inputs from the requirement collection and analysis phase to the conceptual design phase are contained in the Office Data Dictionary, including, among others, details on the structure of the organization, layout of documents, structure of archives and description of sequences of office activities.

b) Conceptual design steps

In TODOS conceptual modeling, we perform the following design steps:

1. specification of the structure of the most important documents
2. introduction of objects
3. first informal TCM schema of the OIS dynamic behavior (*dynamic schema*)
4. revision of the dynamic schema
5. detailed specification of the dynamic and static schemas
6. completion of the design session through specification of missing elements
7. delivery of the schema to the rapid prototyping phase

8. revision of the schema according to suggestions for modification by the end-user.

These eight steps are iterated until the end-user approves the prototype obtained from the functional specifications. This is the *functional design cycle* of Fig. 2.

Each step is performed in a number of **sessions**, where each session is defined as a sequence of one or more operations of definition/modification of specifications. Sessions are initiated and terminated upon the designer's decision.

Let us now examine the conceptual design steps in more detail.

1 - Specification of the structure of the most important documents

The first problem for the designer is where to start the conceptual modeling task. We propose to start from the most precisely defined elements in the office. The most relevant documents are usually clearly defined at this stage of the design process, since they are seen by the user as the output and goal of the office activity. For example, in our test case, it is clear that the office should produce a contract, and the format of the contract is well defined. On the other hand, the procedure for preparation of the contract is ill-defined. Thus, we leave its definition to a subsequent design step. In this phase, the information in the Office Data Dictionary is used to describe the structure of the main office documents.

2 - Introduction of objects

A global analysis of documents specified in step 1 would suggest the designer to introduce objects. Objects avoid redundancies in documents; they illustrate the set of information shared by different agents, different documents and different activities.

3 - First informal TCM schema of the OIS dynamic behavior

The third step consists in the activity of modeling the dynamics of the OIS. A first draft of the TCM schema is set up by making an intense use of the graphical representation of the dynamic schema. The obtained schema is used as a basis for verification of the correct understanding of OIS requirements by the designer. Unclear issues are submitted to the attention of the design team who prepared the descriptive schema of the office. One or more meetings are held with this team and with the end-user in order to clarify these issues.

4 - Revision of the dynamic schema

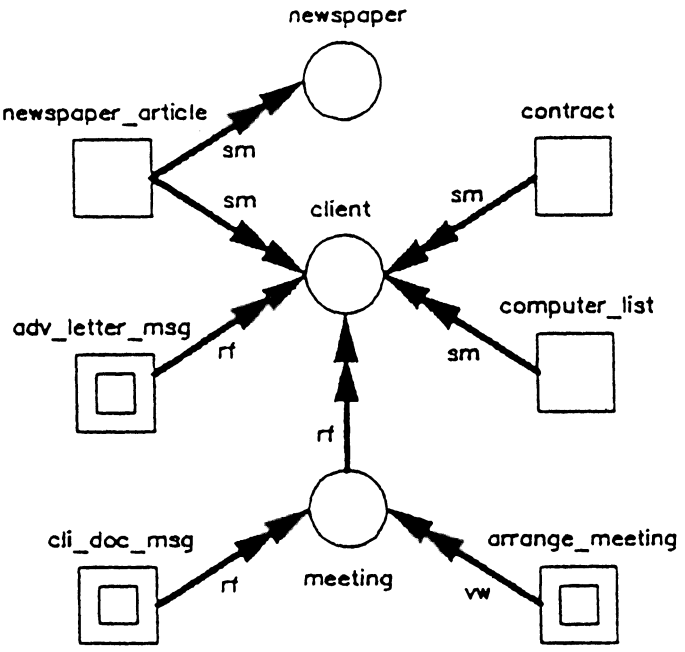
The dynamic graph is revised according to the results of the meeting with the analysis team.

5 - Detailed specification of the dynamic part

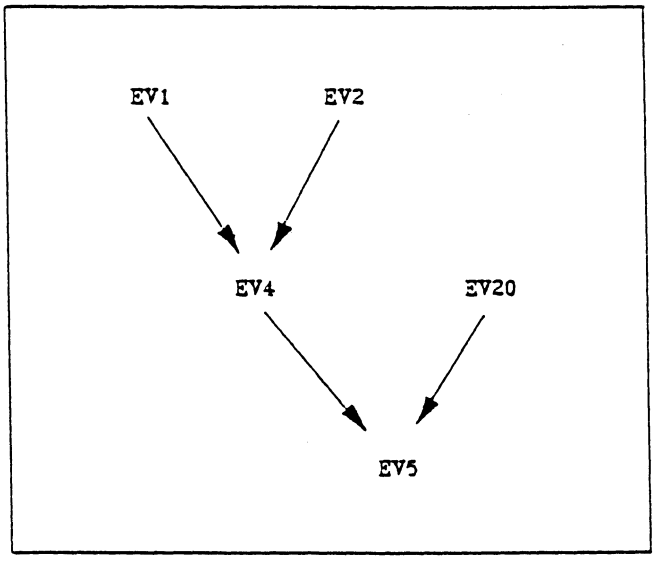
The graphical OIS conceptual schema is refined and formally defined in this step. The designer formally defines the elements of the graphical dynamic schema. In the process, he examines the activities performed in the office, the data utilized by each activity, the performer of each activity and the responsible for each event. Event precedence is carefully considered (see Fig. 25b). Although the final result is the delivery of a complete and correct specification of the dynamic schema, the other graphical representations of the specifications are utilized during this step as tools for defining and checking the specifications.

6 - Completion of the design session through specification of missing elements

When the dynamic schema is formally defined, the designer checks its completeness.



a) static-reference network (portion)



b) event-precedence graph

Figure 25: Examples of the static-reference and of event-precedence graph

In general, several intermediate static elements are defined in the conceptual static schema, in addition to the main static elements already defined in the first step. In this step, the designer formally defines all undefined office elements, until the conceptual schema is found consistent and complete.

7 - Delivery of the schema to the rapid prototyping phase

When the designer terminates the definition of the conceptual office schema, he delivers it to the rapid prototyping team, which, with the help of the prototyping tool, sets up a prototype of the OIS. This will be operated by the end-user to check whether the system responds to the requirements.

In the first iteration of the functional design cycle, all the specifications are delivered; in subsequent iterations of the design cycle, only incremental modifications of the formal specifications are passed on to the prototyping team. The schema is delivered by the designer at the end of one design session, and the delivered schema must be complete, that is, it has to contain no undefined element.

8 - Revision of the schema according to suggestions for modification by the end-user

The prototype is examined by the end-user and modifications and additions are suggested. The modifications are examined by the requirement collection and analysis team, and appropriate updates are performed to the Office Data Dictionary. Updates are notified to the designer who in turn incrementally modifies the OIS conceptual schema going through the necessary design steps.

6 C-TODOS as a Support to Cooperative Work

This section deals with aspects of cooperative work that may arise in the conceptual design of an OIS and that can be handled by C-TODOS.

Cooperative work is receiving much attention in areas such as group decision making, collaborative production of documents, meeting organization, etc. (Greif and Ellis 1987).

Design of applications is also an area where cooperative work is relevant, especially when designing large systems: for example, environments for the development of large software systems can be regarded as a set of tools aimed at supporting a group of people that cooperatively work towards a common goal. Computer-based support in these environments, as well as in IS and DB development, is extremely helpful to coordinating the different development tasks executed by different people.

In the conceptual design of an OIS application, cooperative work issues arise because the whole design task can be split into sub- tasks leading to separate design of different aspects of the application. For example, differently skilled analysts can work separately on different **functional** aspects of the application: the structure of documents, the office procedures and their synchronization, the communication aspects, etc. Alternatively, if the OIS application affects various organization areas, such as secretarial support, management, administration, staff support, the design can take place by **organization** units.

In both cases, each team of analysts produces a **conceptual subschema** of the OIS application. An **integration phase** will then merge the subschemas into the **global conceptual schema of the application.**

Automatic support to OIS cooperative conceptual design is useful both in the stage of designing the subschemas and in the integration phase. C-TODOS can be enhanced to provide support to cooperative design by providing further facilities for information sharing and for communications within the design team. Automated assistance to the cooperation aspects involved in the cooperative design helps to manage design information dictionaries, to maintain the whole schema consistency and non- redundancy, and to support cooperative work sessions (Greif and Ellis 1987) of the design team.

In (Fugini and Pozzi 1989), the enhancement of C-TODOS to support cooperative conceptual design has been investigated. The result consists of a **method** for cooperative design and new **functions of C-TODOS** needed to design the subschemas and to support their integration.

In the following, we illustrate the case of cooperative design performed for different organization units: first the cooperative design method is illustrated (Sect. 6.1), then the new functions of C-TODOS are discussed (Sect. 6.2).

6.1 The Method

The first step in the cooperative conceptual design of an OIS application is to decide which issues of the application can be designed separately. Subsequently, the various design tasks can be performed independently by the analysts. Finally, the products of these activities are merged into the global conceptual schema.

The OIS application is viewed as a set of organization units that perform internal activities (such as document handling, information filing, etc.) and that *communicate* to each other through message passing. A **design unit** is the task of producing a subschema for each organization unit.

The cooperative design method consists of the following stages.

a) Design of a raw global schema and identification of the design units

Through informal or computer-assisted conversations (for example based on WYSIWIS (What You See Is What I See) tools, as presented in (Stefik et al. 1987)) among the participants in the design, the overall structure and functions of the OIS are identified and a raw global conceptual schema of the application is produced.

The purpose of this step is to help the whole design team to get a basic understanding of the application, and to identify the design units.

The raw schema is a TCM *dynamic schema* where the transitions represent the internal communications of the application.

Design units are the objects of the raw schema that are connected by transitions representing communications within the OIS application. Each design unit has a **communication interface** through which it sends and receives messages from the other units.

b) Detailed design of units

The design units are assigned to the analysts who work asynchronously on their subschemas. The units share a **design dictionary** where the specifications of entities of

common interest for all the units are contained. For example, the <person> object can be defined as a public entity of the dictionary as follows:

```
<person> is-a object, public;
    {aggregation-of
        {name: string (20);
         address: address-domain;
         phone-#: ph-num-domain
        }
    }
```

Then, the design units can *import* the definition of <person> and specialize it according to their needs. For example, within one unit the <client> can be defined as a <person> having, additionally, some properties describing the orders of the client:

```
<client> is-a <person>
    {aggregation-of
        {product-orders: ref-to order;
         payment-mode:
             association-of modes: payment-mode-domain
        }
    }
```

A design dictionary provides the basic facilities for importing and exporting entities from the shared area to the design units and vice versa.

Design units can export specifications from this work area into a dictionary buffer; the dictionary administrator is responsible for consolidating the dictionary permanent storage after the examination of the buffer's contents. Fig. 26 shows the structure of the dictionary and a sample export operation. The *internal-message* is a new TCM concept (see Fig. 27) that models communications within the application and that is the basic entity at the level of the *interfaces* of the design units. An internal-message can be sent from a unit (output-message) or received by a design unit (input-message). The semantic structure of the internal-message entity is shown in Fig. 27. Upon its definition, an internal message is transferred automatically by the support tool, as uninterpreted text, from the defining unit into the interface of the addressed unit. The analyst of the addressed unit checks the consistency of the message with the requirements of his unit. If the analysts of two units need to cooperate on the definition of a message, a **cooperative work session** is performed, supported by the cooperative design tool using conventional cooperative work means, such as message passing (Malone et al. 1987) or electronic mail (Kaye and Karam 1987).

c) Integration of the subschemas

In order to obtain the **global schema**, the integration task consists of the following steps:

1. integration of the units' interfaces

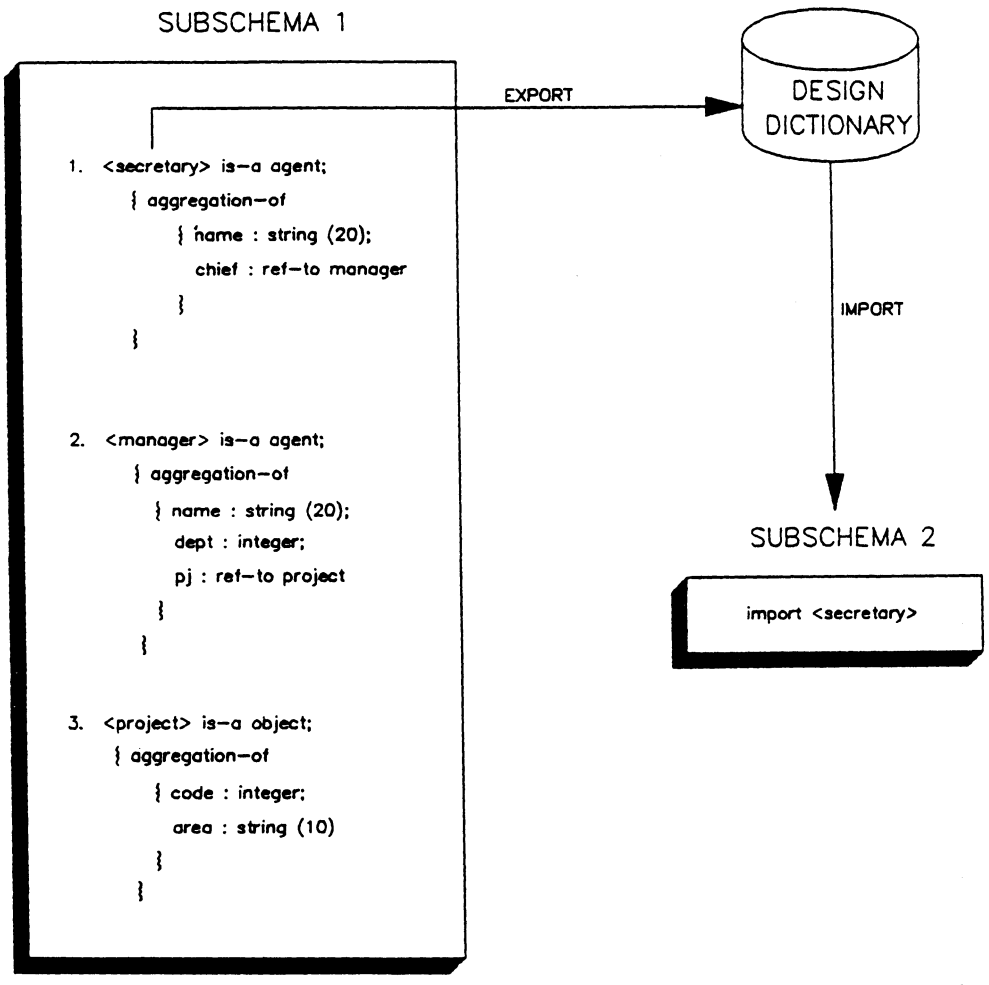


Figure 26: The design dictionary

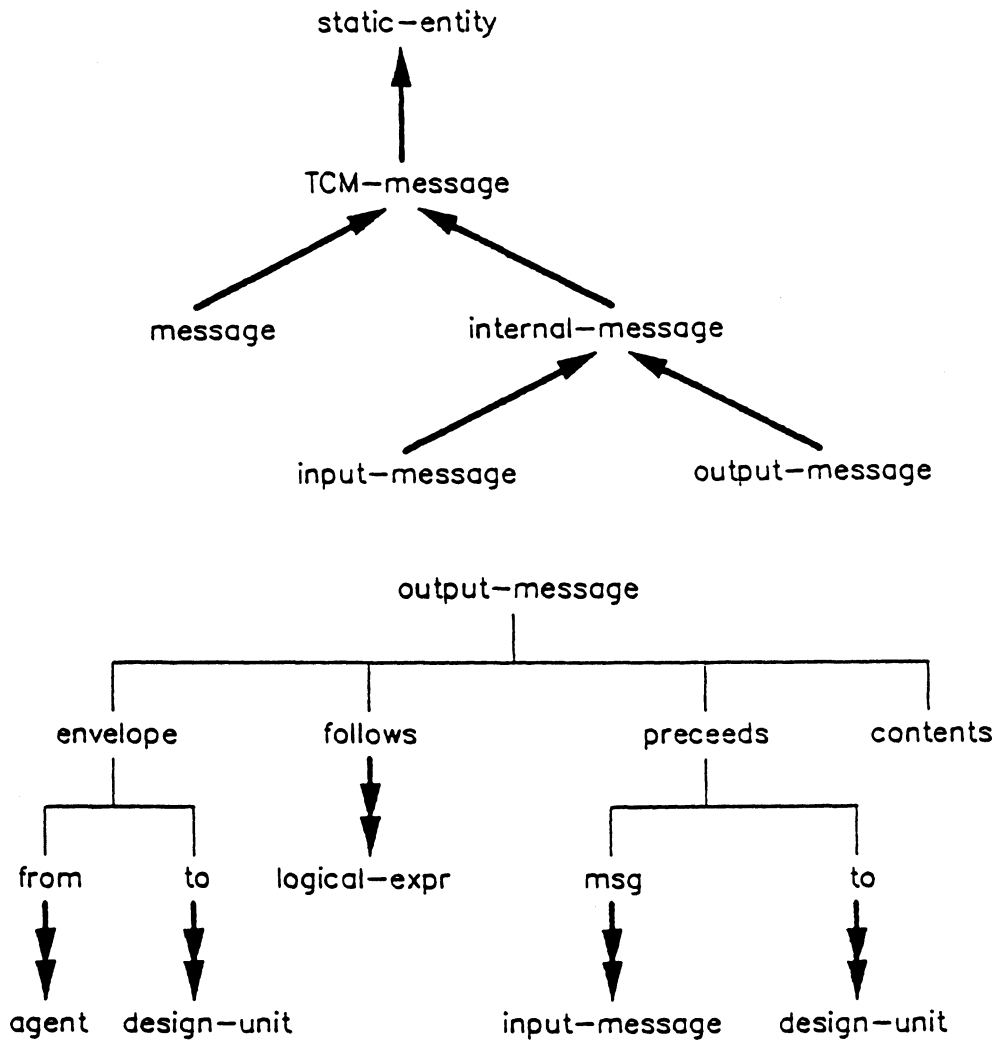


Figure 27: TCM <internal-message> entity

2. merging of redundant specifications.

The first step consists of recognizing pairs of related messages and in designing the object or document that corresponds to the pair. In fact, these messages cannot be modeled as TCM messages because they are internal to the application.

The second step consists of identifying the specifications that belong to different subschemas but have the same semantics. This allows one to produce a redundancy-free global schema. For example, a unit may define the <client> object as here above (see step b.), and another unit may define client as follows:

```
import <person>;

<client> is-a person;
    {aggregation-of
        {personal-data: aggregation-of
            {birth-date: date;
             birth-place: string(10);
             soc-sec: integer;
             confidential-info: ref-to confidential-dossier
            }
        }
    }
```

In the global schema, the analyst may want to have just one definition of <client> and integrate the two definitions into one entity whose properties are the set union of the properties of the two entities.

6.2 C-TODOS Features for Cooperative Design Support

The cooperative C-TODOS (CC-TODOS) tool is the enhancement of C-TODOS proposed to support both subschema design and subschema integration.

CC-TODOS comprises two new modules:

- the unit design module that supports subschema design
- the global design module that supports subschema integration.

The **unit design module** handles the interface of the design units and information exchange between the unit and the dictionary. One basic feature is external reference handling.

The **global design module** supports the functions of merging the units interfaces. In these interfaces, internal messages are defined. These have to be transformed into objects or documents, because they are internal to the OIS application. This task is simplified because of the early consistency mechanism that transfers a message immediately from the defining unit to the addressed unit (see Sect. 6.1).

A complex task to be supported by the global design module of CC-TODOS is merging of similar entities among the subschemas. Also this task is simplified through the dictionary: when entities are considered of common interest for the whole design,

they are exported to the dictionary and made available to the whole design environment. We can suppose that the analyst who is in charge of the integration, or the project leader examines the dictionary on a regular basis in order to eliminate the redundancies. However, entity merging remains a hard task and must be supported by ad-hoc queries directed to all the subschemas, or distributed queries.

7 Discussion about the Conceptual Design Method

The method and tool described in this chapter for conceptual design in TODOS show implicitly the benefits deriving from the use of a design support tool, in particular for bookkeeping operations in incremental design, and for checking the consistency, completeness and accuracy of the conceptual schema.

We conclude this chapter with a brief discussion about some problems concerning the **quality** of the result of the TODOS conceptual design phase. In fact, a good method, and consequently a tool supporting this method, should provide support not only for the production of a schema, but also for helping in the production of a *good-quality* schema. Although this is still a research issue, some points that have been observed while testing the method and tool on some small-size examples, and on the TODOS test case. These observations are also hints to overall enhancements of computer-based OIS design tools in general.

The first point concerns the **difference** between the design of a computer-based OIS and an office system in general. According to the method, we assume we have a description of the activities performed in the office, and from this description we extract the OIS conceptual schema. In the described process, much is left to the designer in terms of what to automate and how. The principal problem is the **risk of automating existing procedures**, including inefficiencies and paths that could be avoided using a computer-based support environment in the office. While the goal of some office systems is to have a paperless office, a design that automates all existing office activities will produce the same amount of paper as in the non automated office, or even more. The analysis of document flows and information exchange in the office are still a research issue (Pernici et al. 1989); some work has been done within the TODOS project concerning the analysis of message exchanges in the office (Cazzola et al. 1990), in order to verify the correctness of message exchange in the office, according to given agent protocols, described through finite state automata. However, more work is needed on this issue, in order to take full advantage of the fact of having formal specifications.

Another issue, related to the previous one, is that of considering **exceptions in the office activities**. In general, requirements specifications describe normal activities and document flows. Major exceptions are considered, if they are important for the correct functioning of the office. Minor exceptions, in a non computer-based office, are handled by office employees according to common-sense reasoning. However, when specifying a computer-based support for these activities, it is necessary to consider also minor exceptions. Not considering them could cause the realization of an inflexible system; inflexibility of OIS is one of the major reasons for their failure. On the other hand, considering all possible exceptions is a very heavy task in design. Therefore, a *suggestion* coming from our work is that of providing default modules for general exception handling in the system. These modules should be a predefined part of the office schema, and automatically invocated when an exception arises. For example, if the

agent responsible for a given action is missing, his substitute should be automatically considered for performing the action in case the action cannot be delayed. Future research in this field should cover the definition of such modules, to be integrated in TCM as predefined concepts.

Further work is required also on the extension of C-TODOS to support OIS **cooperative design**; the target is the construction of a design dictionary where **reusable specifications** are stored which are accessible to the OIS analysis team. Mechanisms for dictionary bookkeeping are required to support *application engineers* in the task of classifying into the dictionary structures the OIS specifications produced during the designs of various OIS, and in the task of maintaining the level of redundancy under control. Mechanisms are the required to support the OIS analysts in performing the integration of the OIS subschemas in a consistent and non redundant way. Knowledge-based mechanisms for representing the OIS design activities and elements both in-the-small and in-the-large, a powerful query mechanism based on the semantic queries illustrated in Sect. 4.2, and a friendly interface supporting navigation and browsing along semantic networks of design concepts are the directions that we are currently exploring.

Chapter 4

Office Rapid Prototyping

Antoinette Kieback and Jochen Mader

1 Introduction

This chapter introduces the rapid prototyping phase in the TODOS environment. Rapid prototyping can be used as a process in system development to support several phases in the system life cycle, which lets expect better quality and higher reliability during the development by early use of experiments in preliminary system versions. This phase aims at constructing an OIS prototype, supporting system designers and end users in validating the conceptual design (described in Chapter 3).

The Rapid Prototyping Tool in TODOS is the tool supporting the rapid prototyping phase. It builds an OIS prototype based on the specifications of the conceptual design phase and additional requirements of the analysis phase. Since the specifications of the conceptual design phase are mainly independent of implementation details, the purpose of the prototyping tool is to interpret those designed specifications. In order to visualize the functional description of the OIS, its representation on the screen is the second purpose of the TODOS Rapid Prototyping Tool. These screens represent the user interfaces to the OIS prototype, possibly different for each involved end user.

The prototyping tool is transforming the formal specification into an executable OIS prototype, as far as possible automatically, to facilitate the handling by the designer and to avoid to repeat work, already done in the conceptual design. Since the conceptual design phase provides consistency and other checks, the given formal specification is consistent, syntactically correct and complete in terms of TCM and is used as the basis for the OIS prototype.

In the following the outline of this chapter is represented. In the first section we give a general view on prototyping, how it is seen in literature. There we first show why it is good to use rapid prototyping (Sect. 1.1). Then we present different models in system development (Sect. 1.2) and the principles of evolutionary system development (Sect. 1.3). After an excursion on the term prototyping (Sect. 1.4) we discuss prototypes in the software development (Sect. 1.5). Next, we define prototyping methods (Sect. 1.6), the terms specification, prototype, and target system (Sect. 1.7), and introduce the three most important prototyping techniques (Sect. 1.8). Finally we give an overview of software tools supporting prototyping (Sect. 1.9).

In Sect. 2 we present the prototyping philosophy in the TODOS project. The basic TODOS prototyping concepts are illustrated in Sect. 4, with the static concept (Sect. 3.1) and the dynamic submodel (Sect. 3.2).

The prototyping tool architecture is presented in Sect. 4 describing each module of the prototyping tool (Sections 4.1 - 4.6) and displaying the use of the OIS prototype

(Sect. 4.7). How the prototyping tool is realized in hardware and software, and how the user interaction is established is presented in Sect. 5.

Sect. 6 concludes this chapter with some remarks.

1.1 Why rapid prototyping

> "The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver a throwaway to the customers. Seen this way, the answer is much clearer. Delivering that throwaway to the customer buys time, but it does so only at the cost of agony for the user, distraction for the builders while they do the redesign, and a bad reputation for the product that the best redesign will find hard to live down. Hence plan to throw one away, you will anyhow". (Brooks 1975)

This citation shows a view agreed on by many system developers. Costly mistakes made in the field of system development due to misunderstandings of user requirements and application fields led to new concepts in developing software systems. Knowledge of the changing context was taken into consideration and feedback of the user requirements was integrated into the developing process. This drives towards the concept of prototyping.

This section shows prototyping in general and afterwards how it is used for the TODOS methodology. Before explaining we would like to start with pointing to basic elements of system development.

1.2 System development

System development basically consists of modeling the information system, construction by coding, testing, integrating and documenting, and implementation into the office environment. At this point the adequacy of the system will be either proved - or not. This kind of top-down-strategy is executed by the following conventional methods (Balzert 1982):

1. *Life Cycle Models*:
   The elements of the system development are principally organized in the way that the different activities and results lead straight from design to product, and feedback is minimized.

2. *Phase-Concepts*:
   The procedure and mode of running a software system ought to be normatively laid down. The main task is hierarchically separated into several subtasks which are refined to the executable software solution. The procedure is separated into phases, the results of one phase serves as specification for the next phase.

3. *Software Engineering*:
   Requirements of a new product are vague, unstructured, incomplete, and incorrect. They should be structured in a consistent and complete document, modules

should be constructed according to the principle of information hiding. Specification takes place before realization, and programs should have a structure. This view stresses the idea of software development but leaves aside the activities that lay in front and behind.

These methods lay emphasis on the result of software development - the product itself. The software is seen as independent product detached of user and basis machine. The representatives of these methods view the application context as fixed. Therefore they provide division of labor very early and use requirement documents, once agreed on, until the system is implemented. But in using these methods problems usually arise.

1.2.1 Problems of conventional development strategies

Due to complexity and change of requirements a valid description cannot be of durability. Change of requirements opposes a strict fixation of phases whose results serve as specifications for the next ones. Formal specifications in the software engineering process lack understandability for the user and often even for the developer. Important decisions for activities are made without consulting the user. Thus, "maintenance" is often misused for adaptation of the system to the application context. It is said that conventional development strategies support project control. But lack of information can cause inadequate decisions at the beginning of the project which are first discovered later on in a product's development when revision is very expensive.

On the other hand it cannot be denied that there are a lot of successes based on conventional strategies. Why?

1.2.2 Reasons for successes of conventional development strategies

Feedback is often misinterpreted. It may be called elimination of errors. Developers often have experiences in feasible technical and organizational solutions. A lot of systems are made by developers for themselves; then feedback is inherent. Many developed systems are later transformed for other application fields. Last but not least one must not forget that guidelines generally agreed on may be either circumvented or ignored. To remedy grievances we propose to look at software in the context of human processes like learning, working and communicating. The elements of system development should be set into relation and supplied by further ideas. During development the process of developing is more important than the product. According to this concept the expression "system development" is extended by the characterization "evolutionary".

1.3 Evolutionary system development

In contrast to conventional system development evolutionary system development accepts a dynamic context, and the alteration of requirements. There is no strict division between specification and implementation but relation and supplement. Continuously, during the time of the project, communication and cooperation between users and developers take place. System development is understood as learning process of all the persons involved. This requires a high level of discipline on developer's and on user's sides, as well as dropping the developer's position as the one knowing best. To secure the adequacy and correctness of the construct, prototypes should be designed and evaluated providing a basis for productive discussions. Evolutionary system development

encloses strategies like incremental system development or slowly growing systems, and installation of pilot systems. Incremental system development uses existing concepts as a basis and substitutes elements by new computer-supported components. Complex problems are solved step by step, extending the system while always bearing the main work task in mind. The incremental development strategy can be used for prototypes as well as for the target system.

1.4 Excursion on the term "prototype"

In other technical engineering branches a prototype is a model of a product which is produced in advance. It shows all the characteristics of the anticipated product and all of its functions can be tested. When the results of the tests are satisfactory mass-production will be started. Prototyping in this context is a well defined phase of the mass-production process.

1.5 Prototypes in software development

Prototyping in software development as described in (Floyd 1984) is part of a process that usually leads to only one product poorly defined at the beginning. It is a technical method within system development. A prototype means an *executable model of the target system.* It may be a part of the requirement analysis and of the specification, as well as a model at further stages of development. Each prototype can be a predecessor for the next one or, in the end, for the target system. It always shows essential aspects of the target system and therefore serves as an expressive basis for communication and cooperation between developer, user, and manager. At the practical demonstration of important parts of the target system the user's requirements are checked and, as occasion demands, reviewed. Prototyping gives place to a change of context.

Another term often used in American literature is according to (Balzert 1982) "rapid prototyping". This refers to the speed of prototyping development and mainly consists of similar concepts.

It is wrong to declare prototyping as strategy giving up requirement analysis and overall system design for a trial-and-error method. In this case the term "prototyping" would be a euphemism for the so called "hacker-style". On the other hand, a product is sometimes named "prototype" later on when customers complain about errors. Then they are either left alone to look for the errors and correct them on their own, or expensive redesign is necessary.

Prototyping can be divided into four steps (Floyd 1984):

1. *Functional Selection*:
   At this time the appearance of the prototype must be defined.
   If it has all the functions of the final product but not in detail we speak of **horizontal prototyping.** In this sense the prototype gives the user a notion of the working appearance of the system. It defines the user interface.
   If the prototype shows selected parts of the target system, completely implemented, we speak of **vertical prototyping.** In this sense the prototype is a limited but detailed sample of the target system.

2. *Construction*:
   The less effort - the better! By using existing tools, techniques, and high level

languages the prototype is constructed in a very short time, certainly at the expense of robustness, efficiency, and reliability, but the emphasis should be on the intended evaluation.

3. *Evaluation*:
   Evaluation is the essential step in prototyping. Developer, user, and manager discuss the appearance of the running prototype, requirements are modified or refined. Feedback takes place.

4. *Further Use*:
   Further use depend on the purpose the prototype is constructed for. It may either serve as medium for learning and then be *thrown away* or *used as a component of the target system.*
   We can distinguish between three kinds of prototypes, each with a special designation:

   - Prototype in a narrow sense:
     The list of requirements is supplied by a provisional but executable software system.
   - Lab Model:
     Basis for developers' discussion as well as for questions of technical realization.
   - Pilot System:
     The prototype is used as part of the final system for the application.

We want to use the term "prototype" in its narrow sense. In the following we discuss the classification of methods of prototyping in three categories proposed by (Floyd 1984).

1.6 Methods of prototyping

1. *Explorative Prototyping*:
   This kind of prototyping is used at the beginning of system development. There, problems are obscure and a prototype is constructed to clarify nature and size of the desired system as well as requirements given by the application context. This is done by intense discussion with the user and the management. The prototype is used to settle the desired functionality of the system.

2. *Experimental Prototyping*:
   This method is used to control the adequacy and the feasibility of the system functions. It emphasizes their technical realization. Developers and users discuss questions of software ergonomics. The users go on in refining their requirements.

3. *Evolutionary Prototyping (Versioning)*:
   This method enables the developers to adapt the data processing system to changing conditions of a dynamic environment. The product is not developed in a straight line but in several versions. Each version is evaluated and thus serves as a prototype for the next one. System software is constructed in development cycles and therefore drops the conception of separation. The distinction between prototype and target system is given up. Basis for this is, within an overall strategy,

a functional specification of the system defining user hardware including interface and target machine.

This classification relates to the different stages in the development life cycle, where prototypes are used. In each stage, those aspects should be prototyped that are of special interest at that given stage in system development.

1.7 Specification, prototype and target system

Before relating these terms to each other we would like to give our interpretation of "specification".

Specification

In (ANSI/IEEE 1987) standard no. 729-1983 the term specification is defined as:

- a document that prescribes, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or system component;
- the process of developing a specification;
- a concise statement of a set of requirements to be satisfied by a product, a material or process indication, whenever appropriate, the procedure by means of which it may be determined whether the requirements given are satisfied. (ANSI N45.2.10-1973).

The term *specification language* is defined as "a language, often a machine-processable combination of natural and formal language, used to specify the requirements, design, behavior, or other characteristics of a system or system component ..."

(Hussmann 1986) is pointing out, that therefore the language must have operational semantics. But a specification is also a vehicle for human beings; and for that purpose it should be abstract and understandable, e.g. independent of machine models.

In the past specifications were not executable. Only few were supported by machines. Classical, machine oriented languages were used to write prototypes. By now the situation is different. High and very high level languages having the characteristics of specification languages are available, and interpreters for special specification languages have been developed. Due to these developments the fences between specification and prototype within software system development are removed.

In the context of information systems the state of the art and of the development have not quite reached that far. Tools for prototyping specifications, such as for instance the RUBIS project (Rolland 1982), show the execution process by listing executed actions and happened events. The combination of this execution flow with the interface design is still missing. However, much effort has been spent on writing specifications, bearing in mind that they have to be made executable later on.

Specification and prototype

Both the specification and the prototype have the same methodical goal as (Hussmann 1986) pointed out: "to give a logical, formal description of the anticipated product and

to explore it before starting the large process of implementation". Such formal specifications may easily be transformed into a program code for the prototype. Therefore they are suited for prototyping in a special way. Not only do they serve for the exploration of the specified functions but also for the explanation of more common questions. Within requirements engineering prototypes generated from specifications are of special use. The translation of informal requirements into the formal specification may be checked while the prototype is running.

Prototype and target system

Prototypes can be constructed at several stages of system development. According to (Zuellighoven 1987), the intention behind and the application of prototyping is usually in one of the following two classes:

1. The prototype serves as specification for the next stage and as a learning instrument but is not integrated in the target system. The final system is constructed anew to guarantee a clean software solution.

2. Step by step the prototype is developed and transformed until it may be integrated into the final system. This method is possible since high level languages, efficient interpreters, and compilers are used. Cheap hardware, input/output-components and elements for handling errors are available and do their part to establish this method.

According to the goal prototypes are constructed for different techniques may be used to reach this. We would like to point to three of them.

1.8 Techniques for prototyping

(Floyd 1984) described in her paper the three most important techniques relevant in connection with prototyping:

1. *Modular Design*:
   Modular design supports the integration of the prototype into the target system. It is helpful for the method of incremental development. If necessary, modules may be exchanged. This characteristic supports the replacement of prototype modules with components of the target system. In combination with an extended vertical prototyping, showing the overall structure of the system, modular design enables the communication between modules through interfaces. Modular design includes human interface simulation.

2. *Dialogue Design*:
   Dialogue design is applied to user interfaces. It supports the transparency and the flexibility of user interfaces. It should be possible to discuss and change aspects of a dialogue without any problems. Therefore the conversational aspects of the dialogue should be as far as possible separated from the specific processing aspects of the application field. The concept laying behind is that it should be possible to change either of them without too much affecting the other. Dialogue design refers to the complete dialogue structure, to the choice of system commands, and

the layout of screens. The handling of errors and dealing with special cases are additional tasks in dialogue design. Conversational independence is an essential requirement for the flexibility of interactive systems design.

3. *Simulation*:
   The term "simulation" is applied to a wide field of activities, usually signifying the working replica of something in reality which would be too expensive to test. As a technique for prototyping simulation has a different meaning. There it serves to demonstrate important parts of the target software system which will not be shown in their final appearance. For example, during prototyping time, file management and organization would cause too much effort in constructing them in their intended form as a component of the final system. Careful analysis of storage capacity and retrieval needs - very important for the construction of the target system - would override the size and meaning of prototypes. Thus, for prototyping, storage capacity and retrieval needs may be simulated by a trivial in- core data organization allowing several realistic and relevant cases studies.
   Simulation as a technique for prototyping clearly depends on modular design. Otherwise it would not be possible to separate the component to be simulated out of the complete system.

Up to this point we have discussed various aspects of prototyping. Certainly prototyping is neither something without foundation nor bound to special tools, but it is made easier by the existence of comfortable software environments based on modern construction methods and supported by software tools. In the following we illustrate general demands on software tools and present a few of these software tools supporting prototyping.

1.9 Software tools for prototyping

There already exist several amounts of methods and tools useful for prototyping, like (see Floyd 1982):

- very high-level languages,
- database management systems,
- dialogue definition systems,
- interpretive specification languages, and
- symbolic execution systems.

All these tools are general-purpose tools, offering no model of the application area, but supporting rapid development of demonstrable operational system versions.

1. *Relational Databases*:
   Relational databases allow *formal* descriptions of the software system as well as *informal* ones in natural language; there might even be mixed descriptions. In addition, they allow revisions and reconfigurations of the software design without altering the architecture model.
   However they lack some very important requirements of prototyping:

- they do not allow recursive structures;
- they are not suitable to represent hierarchical class/object structures as far as prototyping demands;
- they lack any control/scheduling mechanism to execute transactions on stored objects; and
- they are mainly used for management information systems (MIS), not supporting an application area on a larger extension as required by prototyping.

4^{th} generation languages, e.g. NATURAL (Natural 1986), are more powerful than the conventional relational database languages. They offer the following functionalities:

- intelligent editors for global data elements;
- verification and processing rules, integrated with mask generator definitions;
- interactive test and debugging;
- integrated text and data application support containing text-retrieval-functio
- multiple output options using graphics;
- features for prototyping.

2. *Integrated Tool Kits*:
   An integrated tool, supporting the different types of prototyping (as discussed above), would be the most convenient solution for the construction of prototypes. (Floyd 1984) gives a view of how an effective prototyping tool kit may look like. It should be composed of an editor and a text processing system, a high-level language, a database system, simulation facilities, a statistical package, a window/ screen management system, and a compiler-writing system. In addition a program library as well as tools for configuration, embedment, and integration of existing programs are needed. The tool kit should be integrated to make effective work possible.
   Unfortunately available tools (like ACT/1 described in (Mason 1982)) are mostly related to special tasks. In general (as for ACT/1) they support the requirement collection phase and therefore do not fit into other prototyping types. There exists no tool on the market which supports an object/class hierarchy.

3. *Very High Level Languages*:
   Very High Level Languages (VHLL) are suitable for prototyping because they allow the programmer to concentrate on the presentation of the problem and not only on the coding of the solution. This advantage can be compared to the one of prototyping where the developer in first line is interested in the process of system development and secondly in the final result. Therefore similar concepts underlay VHLL as well as prototyping.
   The effectiveness of VHLL is increased if they are used within an environment that simplifies organizational duties, providing a coordinated set of tools. A VHLL should allow the conversion between its data structures and its programs and therefore guarantee reflexivity. Of importance is the possibility to combine it with other languages and software tools. VHLL should also be interpreted. An

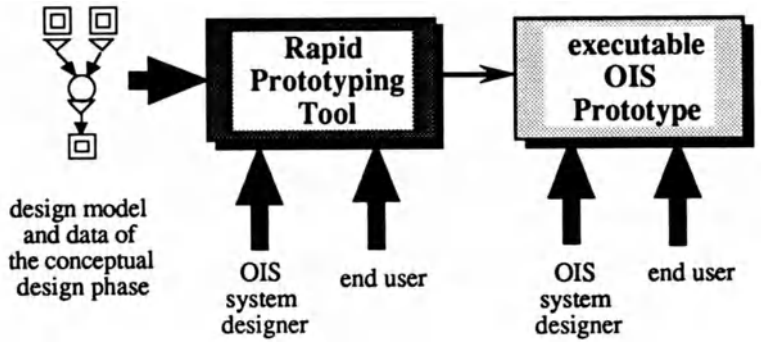


Figure 1: TODOS Rapid Prototyping Tool

incremental compiler within the VHLL is favorable for quick prototyping in cycles, as well as a structure of small units easy to translate. To increase the speed of the development- and debug-cycle, incomplete programs written in VHLL should be executable.

2 Prototyping in TODOS

In the previous section we have presented the overall ideas of rapid prototyping. Now we show how prototyping has been realized in the TODOS project.

In the TODOS methodology, prototyping takes its place after the conceptual modeling of the office system, described in Chapter 3. The modeled specifications are the basis for prototyping, they are used to formally describe the static and dynamic aspects of a future office system, previously defined by the requirements collection methods (see Chapter 1). The prototyping phase transforms the formal specification into an executable prototype, representing the conceptual model and the semantics behind it. As the specifications become executable, the office worker who has defined his requirements to a specific office system may validate that prototype, whether it fulfills his expectations. The presentation of the functionality via a prototype is one aspect of prototyping in TODOS.

During the conceptual modeling phase, the user interaction with the future office system does not play a major role. The modeling of a user interface to the office system prototypes is a second aspect of prototyping in TODOS.

Following from this, the objectives of the prototyping phase in the TODOS project where to develop a software tool, based on the conceptual model TCM and on requirements (like layouts, and instances), building executable prototypes as models for office systems, and to provide a user-friendly interface connected to the office system prototypes. Closing the functional design loop, presented in Chapter 1, a user of the office

system will test the developed office prototype on the aspects of functional flow and user interfaces according to his own requirements.

There might evolve two possibilities: The user is either satisfied with the results or he will propose some alterations. These alterations will be delivered to the requirements collection phase, and the cycle is thus being closed. This loop will be iterated until the office prototype is acceptable for the customer.

Using rapid prototyping in TODOS is based on the following considerations:

- the characteristics of the final office system are usually not well defined at specification time;
- during the prototyping cycles the designers and/or users can make constructive use of experiences gained during previous cycles;
- prototypes are means of communication and feedback between designer and office worker;

Within TODOS office systems design the prototyping is in essence a mean of expressing the user's functional requirements and a way to demonstrate their implications. When working with the prototype, the user and the office system designer may refine the given requirements or gain new ones.

According to these objectives we have to automate the process of transforming the specifications into programming code for the prototype and creating user interfaces to the prototype.

With respect to the characterizations of prototyping presented in the previous section, the prototyping phase in TODOS can be classified as vertical as well as horizontal, experimental, being a "lab model" and being used for dialogue design. As in vertical prototyping the TODOS methodology takes some aspects of the whole system and prototypes the functional flow of them in detail and in horizontal prototyping it takes the whole system and prototypes the functional dependencies on a higher level. Whether prototyping is vertical or horizontal depends on the demands of the user requirements. The fact that the produced prototype is used as a basis for discussions and answering questions on realization makes the TODOS prototype a "lab model". This "lab model" is used to control the adequacy and the feasibility of OIS functions and emphasizes their technical realization, which is called experimental prototyping. Due to the interface generation of the OIS prototype, prototyping in TODOS is also seen as dialogue design, individually to the office worker, whose requirements are applied.

The prototyping tool in TODOS is used by a designer, who knows TCM and the handling of the tool and a user, who previously expressed requirements in the first phase. The user normally is an office worker, who wants to see how the requirements are interpreted and how the system he wants to interact with in the future, looks like. The prototyping tool is not meant to be used just by the user, without a designer. Mainly the designer is working with the tool and will be supported by the office worker, while modeling the user interfaces.

Functionalities of the prototyping tool in TODOS

Starting the tool the designer is able to create a new prototype for a new application. The data for the prototyping, representing the formal description of the user

requirements, are transferred from the conceptual design phase to the prototyping tool. Furthermore the designer may change or update existing prototypes by inserting new classes, deleting or modifying existing classes in that prototype, if the user requirements for example changed. This must be realized via the interface to the conceptual design phase and will not be done interactively by the prototyping tool, to guarantee consistency between the two phases.

The remaining functions of the prototyping tool are widening the OIS prototype with layouts (representing the user requirements of the interface to the office system) and instances (representing existing objects of the office system like data of agents, predefined documents contents), and help the designer editing office primitives, being simple functions common in all prototypes, such as creating a new document, editing or printing documents.

Interacting with the OIS prototype is the business of the office worker, possibly under supervision of the designer. The user identifies himself to the OIS prototype and sees the interface as well as the messages belonging to him. Sending one message after the other to the prototype, he watches all the consequences directly on the screen. A report on the right side of the screen is documenting the execution sequence of the prototype and is intended for the designer to observe the processes formulated in TCM and the user requirements.

3 Construction of a TCM Based Prototype

In Chapter 3, TCM has been described in detail including the functions to design the applications interactively. In this section, the construction of a TCM based prototype is explained. This model is the basis for the prototype interpreter, which executes the built prototype.

The TCM is composed of a static and a dynamic submodel, which on one hand describe the characteristics of static objects of an office system like

- personal data, grouped to "agents";
- reports, articles, grouped to "documents";
- information, communication, grouped to "messages";
- internal data, grouped to "objects".

and on the other hand describe the dynamic flow in the office system using

- actions, describing the activities;
- events, describing the conditions for a state change and the actions to be triggered subsequently.

These entities of the TCM are not enough for execution. *Instances* are very important, as well as *layouts* for the documents and messages, and *primitives*. As TSL was not intended to be a programming language, some of TCM constructs have to be transformed into code, for example the "condition-part" of the events.

Common to all the classes are the internal constructions, already presented in Chapter 3. They are composed of "aggregation_of", "association_of", "ref_to", "copy_of",

"same_as", and "view_of" (for a detailed description see Chapter 3). For having executable specifications these constructions have to be transformed into machine readable code. The methods doing that transformation are stored in the predefined root-units of the concepts (i.e. "event", "action", "document", ...) and are therefore inherited by all the subclasses.

The processing of the methods depends very much on the kind of entity, and there is quite a difference between what to do with static entities, messages and domains (shortly called statics hereafter) on the one hand and dynamic entities on the other hand.

3.1 Static concept

The model of the static concept of TCM is used as the basis for the data structure of the prototype, the static entities, storing structured data (representing a state) during building and executing the office prototype. These static entities can be handled by dynamic entities producing state changes. Static entities are organized hierarchically and the system is object-oriented, where each entity is represented by an object. For several reasons (most important for instance: supporting object handling, inheritance, and layouts connected to objects), later on described in more detail, the KEE system (Trademark of IntelliCorp) (IntelliCorp 1986) is used to build the TODOS prototyping tool, which generates a prototype upon the specifications created in the conceptual design phase, based on TCM. Structures of TCM are set against KEE structures as in the following table:

TCM-concept x	-	KEE-unit x
property y of x	-	slot y of x
value of property y	-	value of slot y

One can see the close relationship between TCM and the resulting knowledge base, which is due to the object-oriented feature of KEE.

The "is_a" hierarchy is represented by the class-subclass structure of KEE. The different static concepts are discussed in the following according to their structure and use in the generated prototypes.

Agent

All classes under the agent entity represent types of persons in the office interacting with the system. For example, if a secretary wants to interact with the prototype, an instance of the class secretary representing that person is needed. Each agent instance in the prototype has a login name he will use to identify himself to the prototype, because the prototype works on personal views of the office system.

For instance in the case study example (presented in detail in Chapter 6), Mrs. Schneider is a clerical worker and Mr. Schmitz is an Acquisition manager; so two

235	Human Resource	Name	Mrs. Schneider
236		Code	clerical worker
237		Level	High school
238		Cost	DM 60.000
239		Birth Date	2. Dez. 52
240		Duty	Credit administration
241		Duty starting date	1. Mär 73

159	Human Resource	Name	Mr. Schmitz
160		Code	Acquisition manager
161		Level	Dr., University
162		Cost	DM 150.000
163		Birth Date	Acquisition, contacts
164		Duty	6. Jan 80
165		Duty starting date	3. Aug 40
166			Terminal

Figure 2: a) Informal requirements

units, one for each agent, are created as instances of the corresponding classes CLERICAL_WORKER and ACQUISITION_MANAGER respectively, as can be seen in the following figures.

This figure is extracted from the case study requirements list (see also Chapter 6). It contains detailed information on the agents interacting with the office system. These information are used for the design of data structures, for instance those of the CLERICAL_WORKER and the ACQUISITION_MANAGER. Both data structures are generalized in the CREDIT_DEPT_AGENT. Here, the link to TCM can be seen. For more detailed information on the design structures we refer to Chapter 3.

Having the entities of agents designed by the conceptual design phase, the prototyping tool builds a class hierarchy for the prototype, where all the class information are stored.

Messages

Messages are the means for users of the prototype to communicate with the office system (prototype) or with other agents. They are divided into input messages and output messages. The input messages are connected to events, which will occur if a message arrives.

Output messages are transmitted by the system to a special user. Common for all messages is the predefined structure in TCM, as well as in the prototype KB; that is each message has an envelope (consisting of a from and a to part) and contents.

The message envelope may be of several types depending on the classes of the FROM and TO properties.

Due to communication purposes of the users, messages need visual representations, realized by layouts. The layouts are different for each user (instance) and for each message class. Later on we will show the correspondence of the TSL structure and the

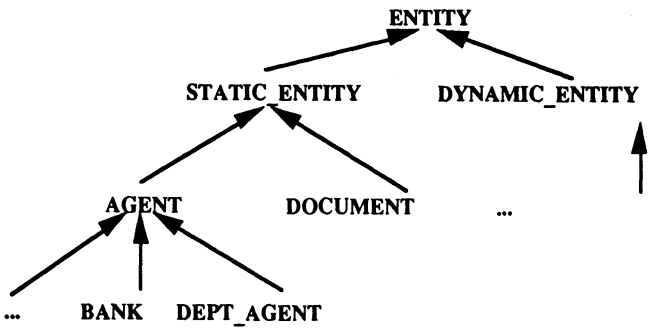


Figure 2: b) Classes represented by TCM

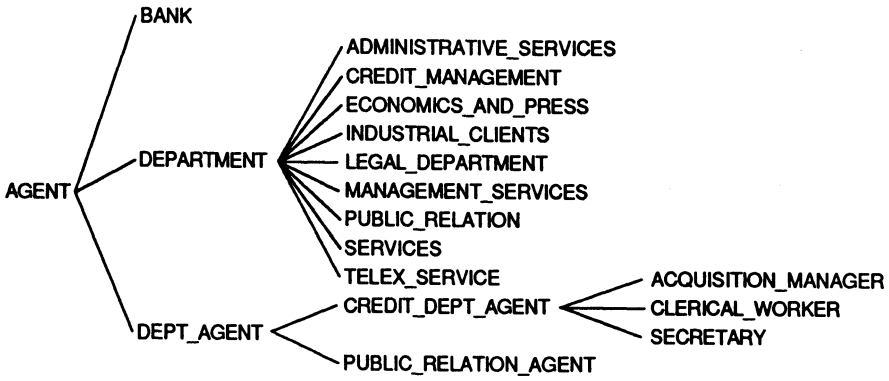


Figure 2: c) Classes created by the Prototyping Tool

layout structure. The behavior of the system is different during the transmission of messages. It depends on the user who is sending the message (the user must belong to the class which is the value of the FROM slot of the member). If an output message is transmitted to a user who is logged in, the message will be displayed on the screen. If the user is not logged in, that means the TO slot does not contain a class, where the user is a member of, this message will be stored in a message KB until an authorized user is running the prototype and will see the message.

Documents

Documents are representing real office reports or articles as well as form sheets, to be filled by the office agents. All documents have visual images allowing users to fill in data. The visual images, which will be prototyped by layouts, are mainly composed of windows.

```
<...> is_a message;
      {aggregation-of
            {envelope: aggregation-of
                  {from: ...
                   to:   ...
                  }
            contents: aggregation-of
                  { ... }
            }
      }
```

Figure 3: a) Data structure of the "message" entity of TCM

	From	To	execution
Input	**agent**	**System**	**arrival-event**
Output	**System**	**agent**	**save message into a separate knowledge base (Message KB)**
Input - Output	**agent**	**agent**	

Figure 3: b) Table of message handling

First the data structure of a typical document used in office systems is shown in the TCL notation. This document is then presented by a layout, showing for each simple and composed data structure in the class letter a corresponding window of one user view, the layout representation of this letter document may be different for an other user.

Objects

The objects in the prototype are used primarily internal in the system to transfer data to an intermediate storing. They normally need no layout presentation, because they are seldom directly modified by the user.

During transformation the main thing to do for statics is to create for every property a corresponding slot, which is going to hold the value of the property in instances of the class, but which also has a number of facets describing the domain of the property and providing access functions and a test predicate to check whether a value is in the slot's domain, for each slot.

```
<letter>   is-a document;

{aggregation-of
  {sender-address  :      aggregation-of
                         {name   : string(20);
                          street : string(*);
                          town   : string(15);
                          state :  string(10);
                         };
  receiver-address: aggregation-of
                         {name   : string(20);
                          street : string(*);
                          town   : string(15);
                          state :  string(10);
                         };
  location    : string(10);
  letter-date : date;
  reference   : text;
  letter-text : text;
  signature   : image;
  }
}
```

Figure 4: a) TSL description of entity "letter"

Slot-specific access functions are used instead of the KEE slot access functions to implement the dynamic behavior of the PARAMETER-OF and VIEW-OF constructs of TSL. For the detailed description of those constructs we refer again to Chapter 3.

Domain checking is also done by special predicates constructed automatically for each slot, as it turned out that the KEE "valueclass" concept is not suited to implement TSL data types. Difficulties occur especially where structured data types (AGGREGATION, ASSOCIATION) are used, as those give rise to a whole cascade of units standing for the value of such a slot.

The information resulting from the transformation of static entities is also used by the interface generator module.

Layouts

Layouts are not handled by TCM, so they have to be created by a user or designer interactively, if for a special entity a layout is desired. The structure of the layout corresponds directly to the structure of the represented entity.

The entity frame is represented in a region on the screen possibly with a title. The user will decide whether the regions have borders and titles. The different kind

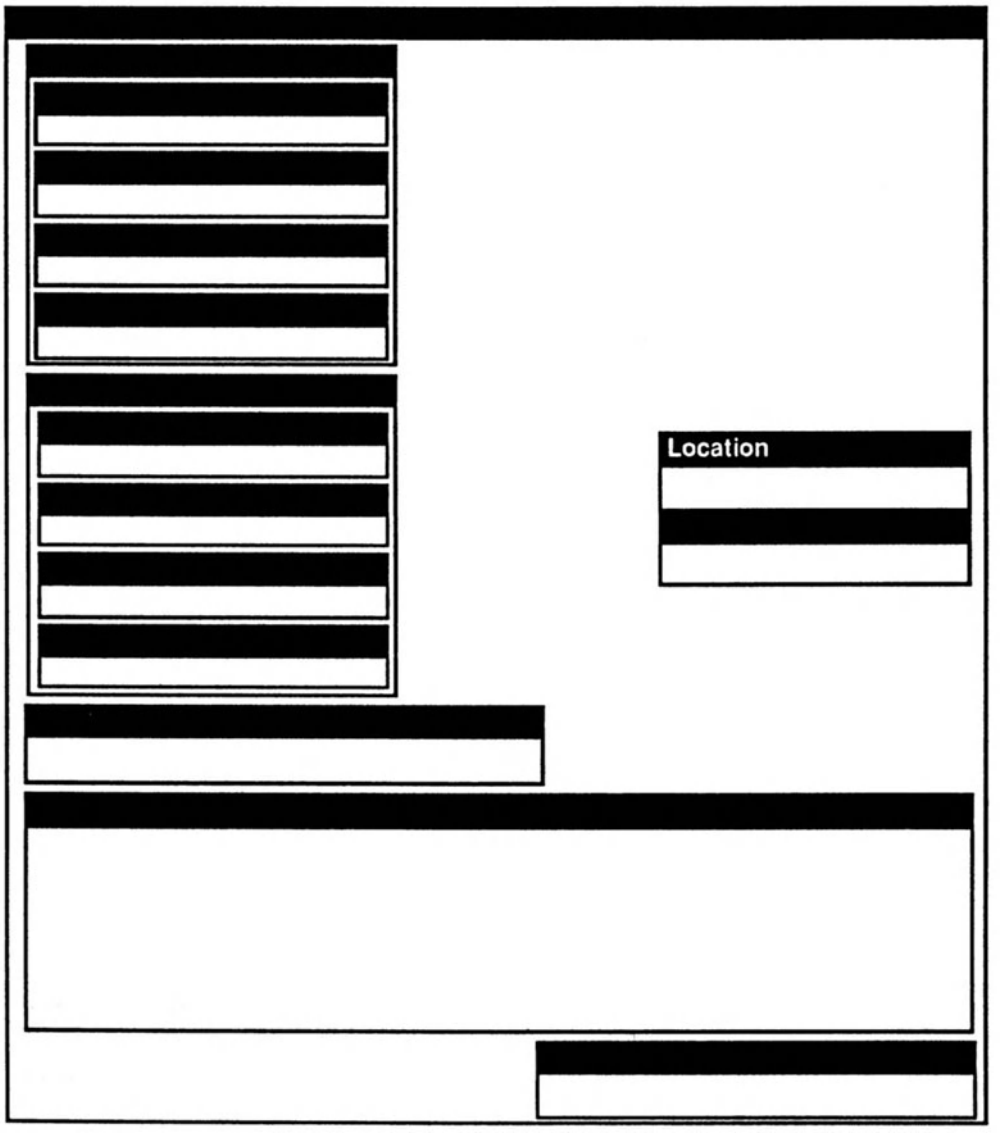


Figure 4: b) Layout corresponding to document entity "letter"

```
<telephone-notice>    is-a input-message;

{aggregation-of
        {contents     : aggregation-of
                        {tel-date : date;
                         call     : aggregation-of
                                    {name      : text;
                                     company   : string(*);
                                     telephone : integer
                                    }
                         subject  : text;
                         recall   : ("yes", "no");
                         notice   : text;
                        }
    }
}
```

Figure 5: a) TSL description of Entity Input-Message

of structures in the entities are represented in different ways, depending on the value types they are connected to. The properties that do not have structured values will be represented by windows which display the text or string.

In the example in Fig. 5, an entity is described first in TSL notation and then with a possible layout for it.

In this example the name telephone-notice is representing the structure of a special message, an input-message, and is displayed by a frame. All the properties are located in the frame, as it is normally seen on a piece of paper. It is not necessary to view all properties, here for example contents is not represented in the layout.

Users may model their layouts according to the contents they are interested in, by leaving out representations of some properties. If for example an office worker for statistical calculations is just interested in the date, the person name, and the company name in the telephone-notice, he will not display the message text as well, to concentrate on the essentials.

Instances

TCM is describing classes of entities but does not deal with instances. For running a prototype instances are necessary to show the behavior using different values in the application. For example TCM describes the classes of agents like secretary, but if a special secretary wants to interact with the system, there must be an instance of secretary representing that special person. Instances have the structure described in the classes they belong to.

To have a good starting point for running the prototype, it may be desirable or even necessary to instantiate some classes before the first run of that prototype.

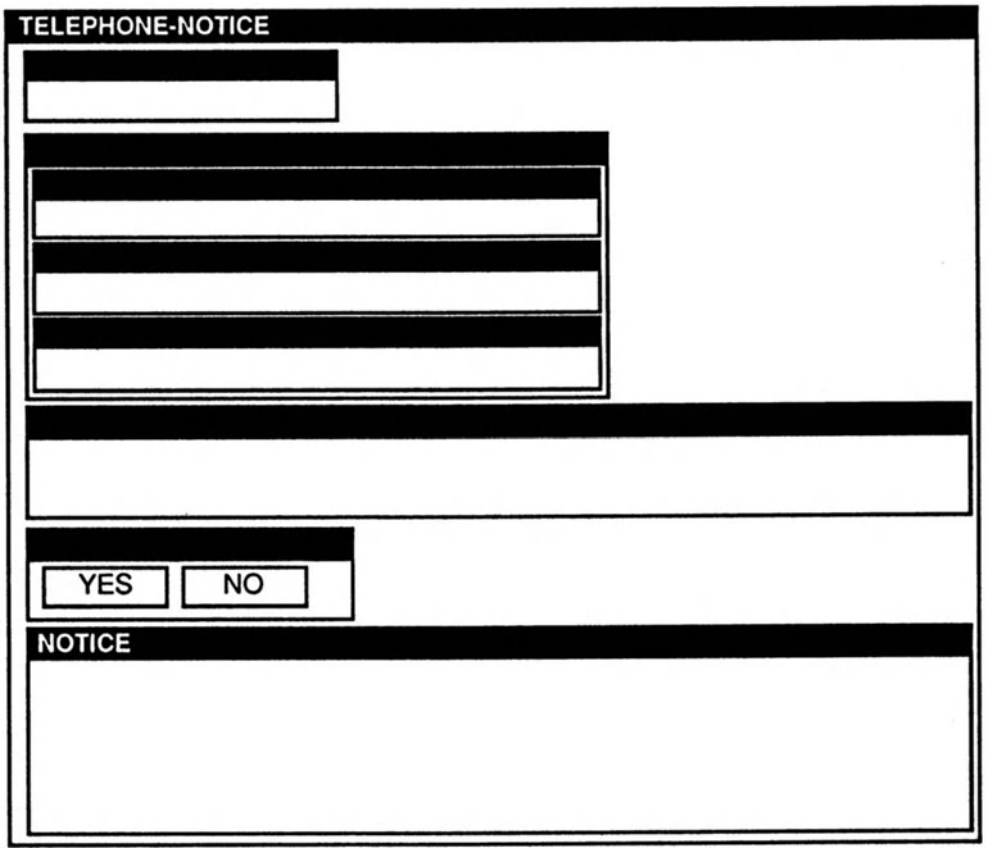


Figure 5: b) Corresponding layout to the entity "input-message"

3.2 Dynamic submodel

The dynamic submodel is composed of actions and events (both have a partly predefined structure). They describe the behavior of the office system, where those events are triggering actions under several conditions, and cause state changes of the static entities (see (Pernici 1986)).

To make this dynamic concept executable the description of TCM is transformed into code. To realize that, different methods (i.e. LISP functions) have to be installed in predefined (and thus inherited) slots, which will then make the prototype running, in cooperation with the scheduler object. For example, events need a method to decide whether the event should fire, actions need a method to perform the operation(s) described in TSL. In addition, actions have a slot whose value is a unit holding information about all the parameters of the action.

Transformation of dynamic entities is an area where designer interaction with the tool will relatively often be necessary. The reason is the possibly widespread use of so-called free text, which is unstructured text that can stand for a condition, describe the outcome of a factor or the expression of a predicate. Primitives which represent basic office functions (see also Chapter 3 and Sect. 5 in this chapter) also take mostly an argument "primitive_comment" which may cause similar problems, as it cannot be used only as a comment, but can also stand for one or more arguments that have then to be specified by the designer.

Events are separated for different applications, where arrival events take a special part. The arrival events are related to input messages which allow the interaction with the system (prototype).

If the user sends an input message, and if there is an event firing on arrival of this special kind of message (which should generally be the case; it does not seem to make too much sense to send a message to a system that, in response, has absolutely no consequences), the prototype begins with a cycle of activities that lasts as long as more events are recognized (which may occur because of actions or because of temporal reasons). Only if no more events are found, the user can send the next message to the system to trigger a new cycle of activities.

The execution flow in the office prototype can be seen as a search through a graph whose nodes are the events defined in TCM. The successors of each node, i.e. the events which should fire next, are then determined dynamically by the scheduler object. For each event that is found to fire, the corresponding actions are triggered, which will generally cause more events to occur and so on.

There is some freedom in the choice of the search strategy. By default, a depth-first strategy is used, i.e. if after execution of the actions triggered by some event several events are found to fire, the consequences of one of those are first completed before another one is taken into account. It seems to us that this strategy lets activities which are logically connected also be executed together. However, other strategies may be wanted.

Breadth-first can be selected simply by changing a parameter. It is also possible to provide every event with a priority and then have a best-first search. If it seems desirable, the designer may even implement any arbitrary search strategy by a relatively simple change in the scheduler object of the office prototype knowledge base.

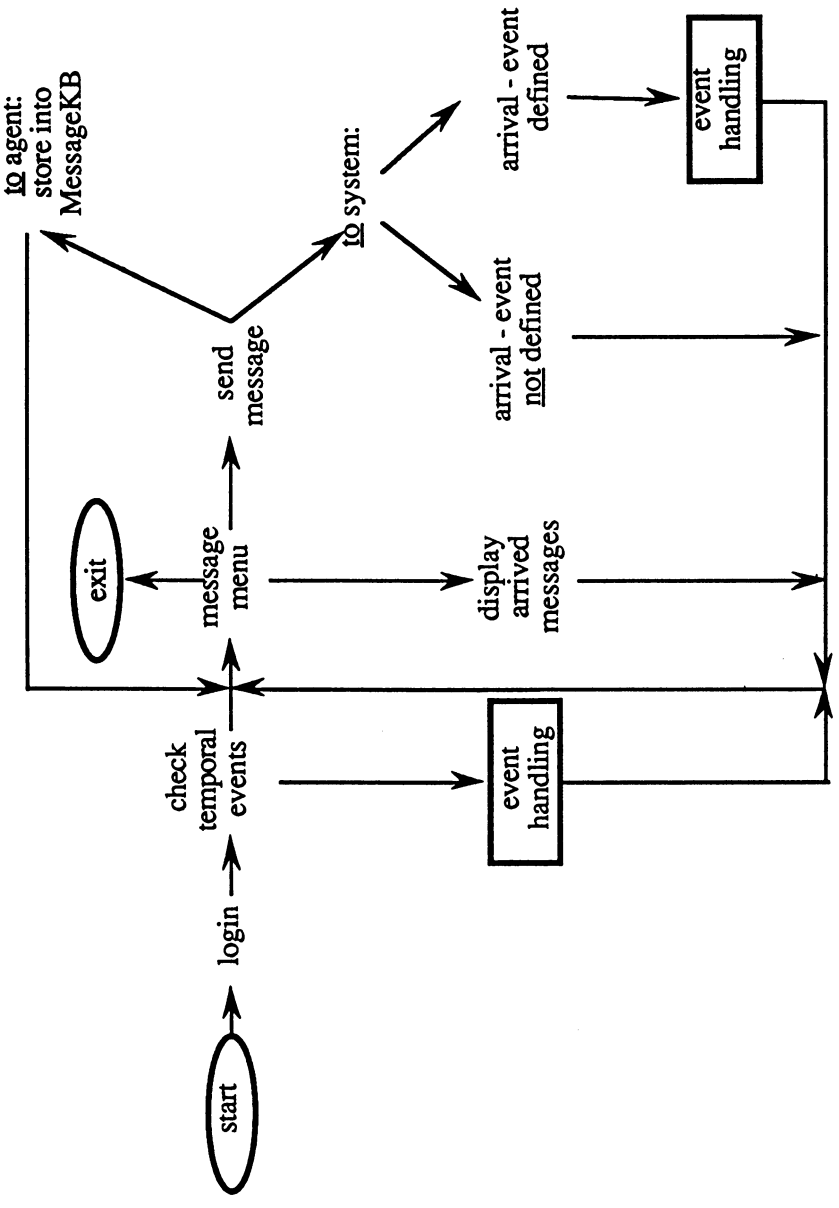


Figure 6: Execution flow of the built prototype

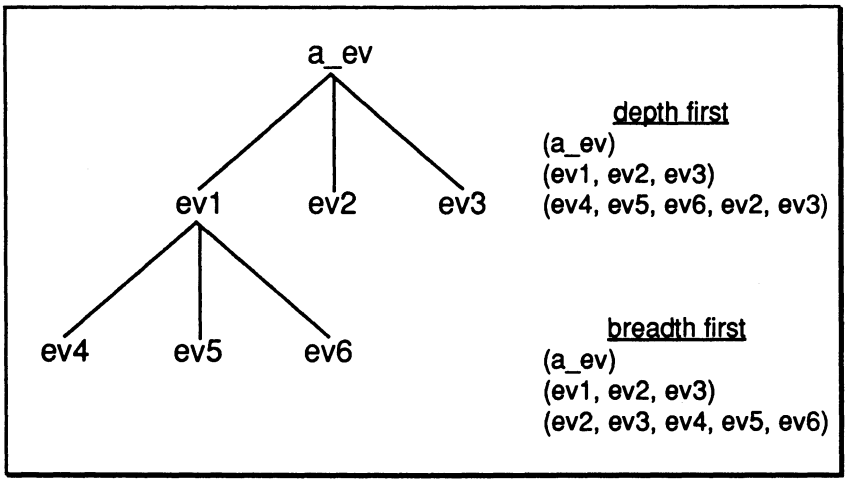


Figure 7: Events execution sequence

4 Prototyping Tool Architecture

4.1 Model treated as a prototype

For prototyping the user requirements, we developed a prototyping tool, based on the TODOS Conceptual Model. This tool supports the methodology of executable specifications and partly automatic prototyping and allows the user of the office system to develop a user interface to his office system prototype.

The more automatic part of the prototyping tool is the transformation of TSL into executable code. As described in Section 3.1, we use Common Lisp and KEE to develop the prototyping tool and the executable version of TCM, the office system prototype.

TCM is reflecting the information of the user requirements, as described in the chapter on specifications. This information will be transferred from the conceptual design phase by a file containing the user requirements in the form of TSL to the prototyping tool, which stores them after transformation in an executable version, in an office prototype knowledge base ("TCM KB" in Figure 8). For changing requirements the tool has to modify this knowledge base. Therefore another file will be transferred, containing the names of the deleted or modified entities.

The non-automatic part contains the primitive editor, an editor for necessary functions, which are not part of the primitive library, and an instance generator, which adds instances to the entities provided by the TCM. Instances are necessary to provide the user a test environment with real data, which will initialize the office prototype.

The tool configuration is such that it first translates or completes the OIS model when a prototype version has been implemented. Functionalities which are used as prototype functions are stored in a library. The tool checks if all functions described in the model are contained in the library and, if necessary, missing functions are introduced by an editor.

In the following we explain the components of the prototyping tool in more detail (see also Figure 9).

4.2 Initialization

The office prototype is based upon the TCM, interpreting the features in the same way as the semantics of TCM denotes. In TCM there is one part which is common to all office information systems which will be designed by this tool. This part contains the predefined descriptions of the static and dynamic submodel. The action, event and message entities have such a predefined structure.

We want to use the prototyping tool for developing several office information systems. To this purpose, the basic part of the TCM with the predefined entities has been stored in a knowledge base. Starting developing a new office information system we will initialize the new office prototype by copying the basic model structure (basic TCM KB) using the "initialization" component in a new knowledge base for a new office system prototype. Upon this basis the designed user requirements of the office information system can be expanded for prototyping purposes and completed with layouts for interfaces.

4.3 TCM Transformer

After each session in the conceptual design phase an insert and a modify file will be transmitted to the prototyping phase. These files will be read by the TCM transformer. The files are containing the description of the user requirements in TSL. The TCM transformer inserts the contents into the application dependent part of the TODOS office prototype (see Figure 9).

The TCM transformer is to create semi-automatically a KEE knowledge base from the TCM delivered by the conceptual design phase. It is important to mention here that designer interaction with the tool during transformation cannot be completely excluded. TSL has some features which are too imprecise to allow for a completely automatic translation. However, TSL was never intended to be a programming language but a specification language.

The transformation takes place in two steps. First, the "is-a" hierarchy of TCM is mapped directly into a hierarchy of KEE- units. In this step, the TSL description of each user-defined concept is put into a slot (called TCM-STRUCTURE) of the corresponding unit.

In the second step, this textual description is processed by a parser based on the TSL grammar to fill the unit with the information which make the prototype knowledge base executable (together with a scheduler module controlling the event/action handling). Each predefined TSL class has its own methods to perform this second step as seen in the previous section.

This component also takes into account deletions and modifications placed in the modified files, but there are just the names of the deleted or modified entities.

4.4 Primitive Editor

Office information systems may be of different types, needing different office functionalities and probably other office primitives are needed for the design of those systems.

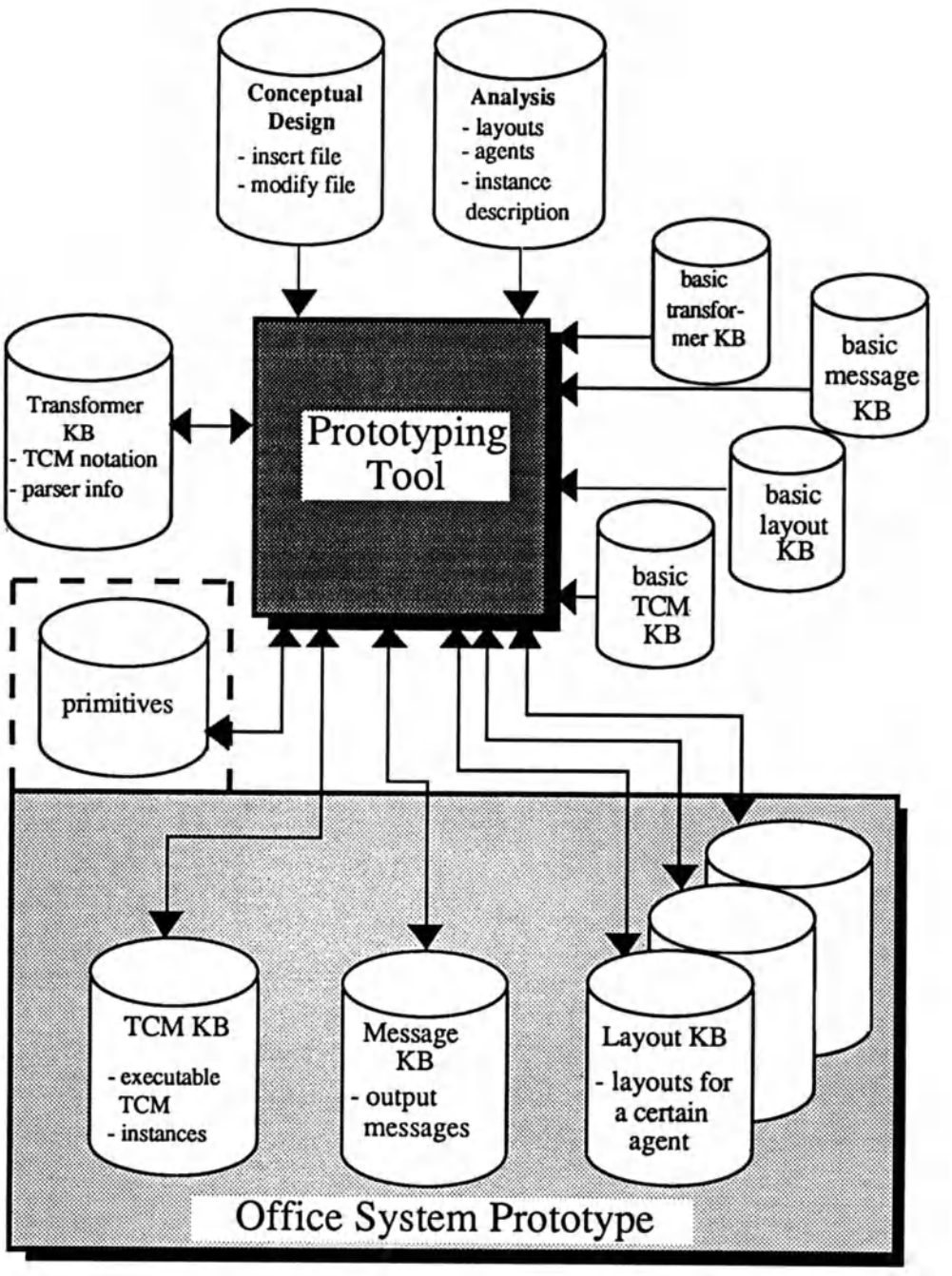


Figure 8: Prototyping Knowledge Bases

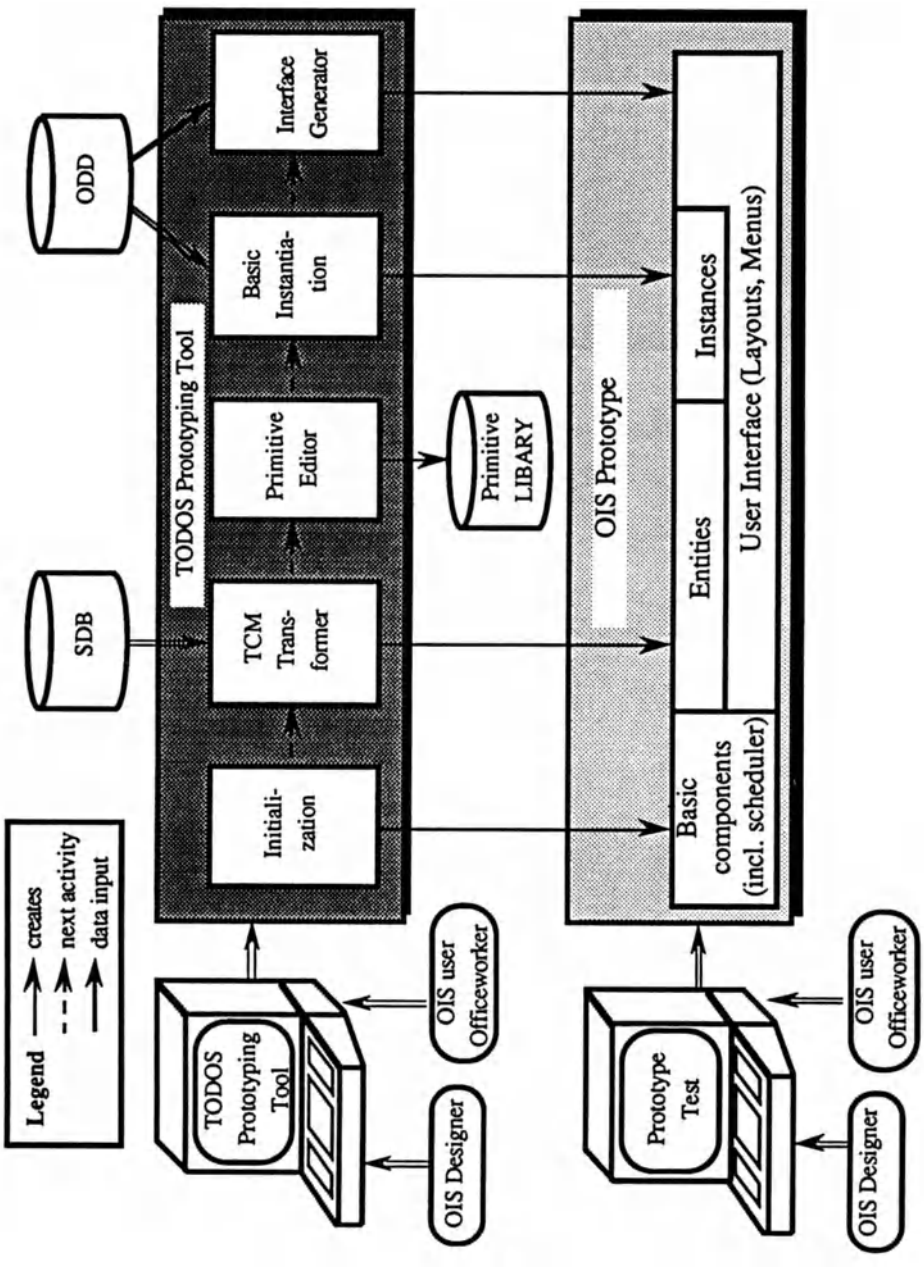


Figure 9: TODOS Prototyping Tool architecture

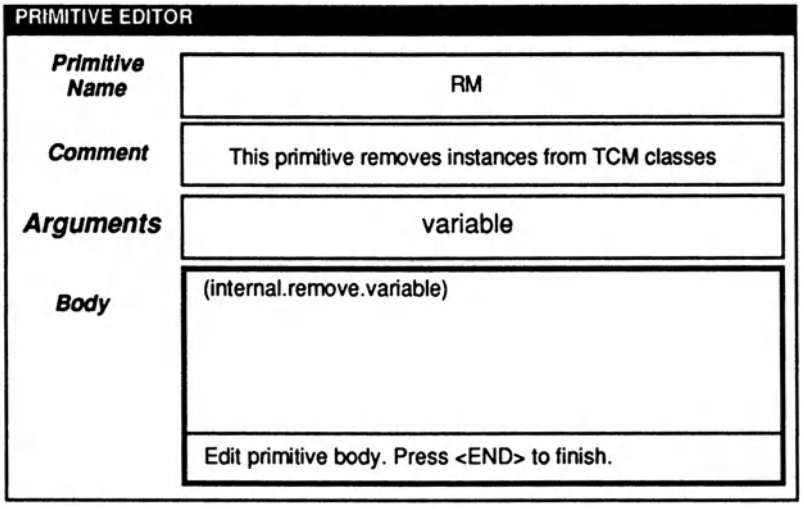


Figure 10: Primitive Editor

It is wrong to think that implementation of "all" office primitives necessary for individual office system design is possible at once. We just have to think about new technologies in the fields of office information systems and the increasing efforts in office automation, which are up to now dominated by manual work.

In making the prototyping tool flexible to build office prototypes, this tool has a component, the primitive editor, to create new primitives, or to modify existing ones, if necessary.

We used this editor ourselves during the tool development to implement our basic office primitives, right from the beginning of the empty primitive library. This library is another KEE knowledge base, which will be loaded when loading the office prototype. It contains the primitives as objects which can receive messages from within the office prototype knowledge base.

The primitive editor supports the following activities:

- creating new office primitives
- allowing editing the program code
- entering code into the primitive library

For transferring the primitives from the conceptual design phase to the prototyping tool, the primitives will be identified by the primitive name and a parameter list.

name (par1, ..., parn)

This figure is showing the layout, the designer is using to edit primitives in the primitive library.

4.5 Interface Generator

This chapter describes the user interface of a TODOS prototype developed using the TODOS prototyping tool. Starting from the prototyping support tool, the user develops his own interface to the OIS prototype. This interface can look different for each user accessing the OIS prototype. In the following we explain in more detail the user interface creation component with the interface generator and the user interface of the prototype.

A part of the TODOS prototyping tool is an interface generator supporting the modeling of the user interface of the prototype.

The user of the prototype should have the possibility to create a user friendly interface to his office prototype. The interface generator supports the user and the designer to build those interfaces. It supports in offering

- choices on different interface techniques
- possibilities to create layouts of static elements, like documents.

This component is the only part the user of the prototype is allowed to interact with. All the other components will be handled by the designer.

4.6 Basic Instantiation

The last component of the prototyping tool is the instantiation component. The TCM is not dealing with instances of the office system types. Those instances are necessary for the prototype to provide the user with test data. For example, via this component, all the users interacting with the office system will be integrated in a subtype of agent.

The instantiation process is related to the static entities. If the entities are already connected to interfaces, they will appear on the screen and one can fill in the data of the instance, delivered by the requirements collection phase. After that the data are stored in the appropriate office system prototype in newly created instances to existing unit classes, representing the entity classes.

There is one exception: the messages. They are handled a little bit differently. Messages are the communication features between the user and the office system prototype. They have a predefined structure with an envelope, containing the sender (From) and the receiver (To), and the contents. We are distinguishing three kinds of messages, as shown before:

- input	From: agent	To: System
- output	From: System	To: agent
- input-output	From: agent	To: agent

The first one is an arrival message and will be created only during execution time. The other two types can be instantiated before running the prototype and will be stored in a message KB.

4.7 Prototype Execution

The office prototype allows the user to test the specifications he made by watching a system running that is directly based on these specifications. Let us see how he interacts with the system and what happens inside it.

To work with the prototype, the user must first identify himself (see also Figure 6), so that the system knows who he is and can fill the FROM property of messages sent by the user and, perhaps more important, can load the appropriate layout knowledge base containing this user's private layouts giving him a personal interface to the office prototype.

He can then either decide to view messages that have been sent to him since he was the last time on the system, or send an input message to cause activities within the office prototype.

We will now take a closer look at what happens in every single search step. Before, it is necessary to say some words about the KEE worlds concept.

KEE worlds are originally designed to allow reasoning with hypotheses by letting any slot have different values in different worlds, i.e. under different assumptions.

The office prototype, however, uses the worlds concept to reach a different goal. Worlds are used to allow a quasi-parallel execution of actions which are defined in TCM to take place in parallel. "Quasi-parallel" here only means that all the actions that shall be executed "in parallel" see the same state of the system, so that, if A1 and A2 are to be parallel, A2 does not see the state changes caused by A1.

Let us now assume that some event E1 has been found to fire and the scheduler decides to take into account this event (there might be others that are firing, too!).

The scheduler then calls the method placed in the triggers slot of E1, which then again calls - possibly conditionally or repeatedly, as specified by conditions or triggering factors - the listed actions and/or primitives with the arguments which are specified (or not) by TCM. If several actions/primitives are to be executed in parallel, they are not called directly, but within a LISP macro that takes care to create for every action/primitive a particular world in which it runs and afterwards collapses all the worlds, i.e. brings into effect the changes that took place in them. The scheduler then looks first whether there are any temporal events to process and puts them into the list of events to look at next (i.e. the list of "successor nodes" of E1). Behind these it places the events-on-event depending on E1. Finally, the other internal events describing state changes caused by the operations which just finished are appended. This completes one search step.

The primitives and the actions consisting of primitives are responsible for any interaction of the office prototype with the user and for all state changes made automatically by the office system. They are worth while being inspected in more detail.

Primitives are the lowest level of system activity that is visible for the user (and also for the designer). They can be called by actions or directly by events. Their arguments must always be specified. A call to a primitive is in fact a message to an object in the primitive library knowledge base, which must therefore always be loaded together with an office prototype. There are primitives which provide system interaction with the user; those make always use of the layouts (of static entities) which are in a separate, user-specific knowledge base. Others simply supply functions changing the internal state of the system by creating or deleting new instances or changing property values.

Actions may or may not be called with their arguments. If not, the action unit has

the problem to collect the necessary arguments. Normally arguments are declared as a ref-to some class. In this case, the default argument is the last instance of the referred class that has either been created afresh or had any change in a property value. The domain of an action argument as well as its present value during the execution of an action are stored in a unit attached to the action unit.

The interface to the prototype was designed to be as simple to handle as possible. Wherever appropriate, the user is supported by menus offering a number of alternatives from which he can choose one. He can also make wide use of the mouse in activities which do not consist simply of a choice, for example, when editing a longer text.

When the user logs out, the changes he made in the prototype are written out to the file system so that he or anybody else can continue where he stopped.

5 Realization of the Prototyping Tool

5.1 Requirements to a development environment

The TODOS prototyping tool is used to demonstrate the effect of a TSL office information specification by showing the functionality of the office system specified. The TSL-specification of the intended office information system has to model not only pieces of software, but real-world entities like clerks too, as far as their functional interaction is necessary to model the office functionality. This is clearly a much more demanding task than only specifying program requirements.

Correspondingly TCM and TSL offer a very powerful tool for specification. So the problem at hand is to transform a TSL office system specification into an executable prototype including the intended system surfaces! In order to tackle this task in an efficient way the target tool or language should meet these TCM-specific criteria:

1. Allow the implementation of an object/class hierarchy
2. Model nested and recursive TCM-DOMAIN constructs
3. Model the TCM EVENT/ACTION - state change mechanism

Most of these criteria directly stem from the object-oriented representation mechanisms chosen by TCM.

Additionally other, more general, programming aspects should be kept in mind, too:

4. The support of user defined data structures
5. The possibility of recursive programming
6. The possibility of data driven programming
7. Tools and facilities for a comfortable system interface construction

These eight selection criteria represent only a subset of desirable features a tool/langu for prototyping should have.

Their fulfillment ensures that the TODOS-prototyping tool can make as much use of existing technology as possible.

5.2 Choosing a tool for TODOS

For deciding which language and tool we could use for implementation, we have analyzed the conditions concerning realizing prototyping and transforming TCM.

As for prototyping, it is important to use languages and tools supporting

- changes, additions and testing of programs interactively, as prototype creation is a dynamic process,
- possibilities for programmatically creating other programs,
- comfortable and flexible system surfaces to provide user friendly interfaces, here for a designer and a test user,
- transparent system behavior.

Concerning TCM, it is useful to have languages and tools which have provisions for:

- predefined hierarchical structure concerning the "IS-A" structure of TCM,
- supports to model nested and recursive constructs with respect to the "aggregation-of" and "association-of" abstractions,
- an easy construction to model state changes provided by event/action concepts of TCM.

As a result of a study on languages and tools, it had been found out that expert system shells, specially the KEE-system, are the best choice for the TODOS prototyping tool according to the selection criteria. This choice is based on the superior programming environment offered. This kind of an environment increases software productivity substantially (factors between 2 to 5 are discussed in the AI-community).

The main drawback of these tools are their costs and the need for a large LISP-machine to run the shell upon. For the cost factor, by far most important is the development time needed for the TODOS project.

A great deal of the productivity stems from the fact that most TCM entities translate almost directly into KEE-structures, preserving the combined functionality of TCM and KEE.

Additionally the Flavor/KEE-system offers with the WINDOW/KEE pictures system an extremely rich toolbox to construct flexible graphical system surfaces. Therefore we think this choice for a prototyping tool really pays.

5.3 Office operations

Now we are going to look at operations, which might occur in an office system.

It is difficult to define, what office work is: "it is easier to define what offices are not" (Tsichritzis 1985). There are a lot of analyzing methodologies for office information systems, analyzing the work of different offices. But, which are those office functions (activities as a synonym) being characteristic for "all" office systems?

Current office systems mainly support the access to large databases and communication networks.

There is no standard in the area of office business. Some ESPRIT projects aim at developing standards and descriptions of those activities. The elaboration of standards

is not part of the TODOS project. It will just make use of the results of other projects. There exists an increasing interest in those standards.

The main task in most office systems is the handling of information, the exchange of office data. Defining whether a task is part of an office or not is difficult and depends on the office objectives.

5.4 Office primitives

The matter of office work is information, appearing as speech, data, text and graphics/pictures. Permanent information, that means in written, painted or stored form, are called office documents.

One can split office work in the following tasks:

- communication
- store / retrieve information
- process information

Office activities can be described as connections of single tasks, which may be distributed to several workstations connected to each other. The connection of several tasks together are called office processes, or actions (in TCM). Examples of such office processes are contract preparation, budget planning, technical documentation, and so on.

(Balzert 1985) provides a list of office applications. Some of them will be presented here, with a short description:

- graphics	creation of diagrams, pictures
- information retrieval	database and knowledge base queries, e.g. library
- electronic archive	storage and retrieval of documents
- calendar	management of terms
- chart calculation	calculation in a matrix, allowing manipulations on table elements
- pocket calculator	ordinary arithmetic functions
- word processing	ordinary text editing functions, additionally integrating data, graphics and so on
- statistics	creation and evaluation of statistics
- electronic	transmission and receipt of electronic documents

mailing

- project planning — planning and management of projects

A basic office primitive is a basic function in an office, allowing the execution of this function. Several office primitives can be composed to obtain global functions, the actions of an office model.

Office primitives in the TODOS project are basic operations like:

- store
- retrieve
- edit
- create instance

In the conceptual design phase, those primitives are used to specify the actions in detail, without considering implementation aspects on software and hardware.

In the prototyping phase those primitives are implemented in code and then will be executed when running the office prototype.

Actions in TCM, as described before, are containing a property 'steps', where the primitives are placed. They are separated by semicolon (for sequence) and & (for parallel execution).

All primitives are defined as procedures, not functions. This means that primitives do not give back a result, but consist only of their side-effect. If one is interested in the value of some operation (as it is the case especially for the 'compute' primitive), one has to use an additional parameter which will contain the computed value after the execution of the primitive.

Since primitives are used in TCM to describe more complex actions, it is often necessary that there is a communication between primitives. For example in *some* action one might want to create an instance of *some* entity and immediately edit some of its properties; then the 'edit' primitive has to "know" the name of the newly created instance in order to be able to access it.

Primitive parameter

Interfaces between the office primitives are managed by the parameter list in the procedure calls. If an action is composed of several interdependent primitives they are passing the parameters as follows:

The parameter "par1" is placed in the "in" property and will be the input for the primitive "prim1". "par2" and "par3" are just internal variables, but "par4" an output parameter, because it is placed in the "out" property. The "prim1" may calculate the value of "par2" using the value of "par1".

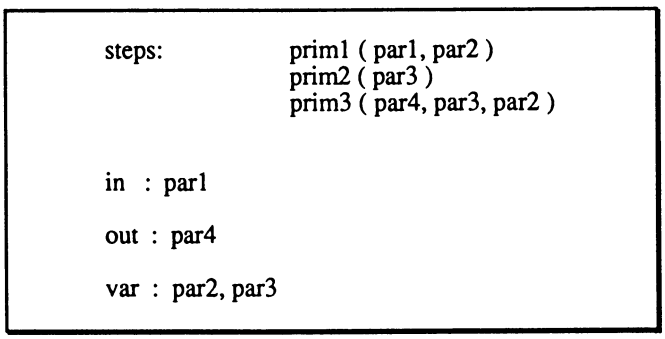


Figure 11: Primitive parameters

5.5 User interface definitions

In this section we describe the main components of the user interface to the prototyping support tool and the prototype. We define hardware and software components which are foundations of the user interface implementation in the prototyping phase and finally, we give a brief survey on the interface between the user interface and the prototyping support tool.

Definition of user interface techniques

As an analysis of the available interfaces on the market shows, the existence of menu-driven and window-based systems has become the state-of-the-art of computer systems, software packages, and information systems. In the following we illustrate the user interface techniques used in the TODOS tool.

The development of the TODOS user interface is restricted by the hardware and software used in the construction of the tool, i.e. the LISP-machine EXPLORER from TEXAS INSTRUMENTS and the software development environment KEE. Furthermore the lack of powerful speech input and natural language interfacing facilities restricts additionally the shaping of the interface. The hardware and software components of the interface are presented in the following paragraphs.

Hardware part

First of all, the interface uses a keyboard. Even a menu-driven information system needs a keyboard to enter names, data, text, and so on.

The use of the mouse is a convenient way to interact with the prototyping support tool. The mouse features buttons that enable the user to execute commands without acting on the keyboard reduce or replace the operations on the keyboard. The operations the user can invoke with the mouse buttons in a particular situation will be described in a documentation window. The mouse clicks are dependent on the software behind them. Clicking an item in a menu stands for selecting that menu point and performing

an operation or invoking another menu. So the mouse will be a suitable and excellent tool for user interactions with the prototyping support tool as well as with the prototype.

Software part

In this section we give some information about the software requirements for modeling the user interface. Window and menu techniques are treated but also unrealistic targets are described.

Windows are special input/output interfaces to display system answers or requests and to enter user actions. In the prototyping tool windows are used to develop the designer and user interface. These windows are permanent ones, the user cannot delete them. Other windows may appear according to the field of application, as frame of forms, menus, text editors, and so on. Temporary windows disappear after the user moves the cursor outside of the menu.

Menus usually are temporary windows for a special purpose that offer the user the choice of items or options. Using the mouse enables him to select mouse-sensitive items from menus quicker than typing a command or pressing a sequence of keystrokes. Command menus present a series of options where the user can select one and execute it, multiple-item menus give a list of items from which he can select several and execute these by clicking "Do it" at the bottom of the menu.

The function of windows and menus within the prototyping tool are described in the following paragraphs.

So the hardware and software part of the user interface lay the foundation for modeling the interface to the prototyping support tool and the prototype.

Interface between the user interface and the prototyping support tool

The user interface of the prototyping support tool is restricted by the fact that acceptable natural language user interfaces are not commercially available and that such an interface is not feasible in the framework of the prototyping phase. Nevertheless we wanted to develop an easy-to-use interface for the prototyping support tool. Therefore we utilized parts of the commercially available interface of the expert system shell KEE and of the EXPLORER Lisp machine.

The interface to the support tool (KEE and Common Lisp) describes the user interaction with KEE functionalities and LISP functions which are presented to the user via a LISP Listener and/or menus. This interface offers the designer of the prototype a variety of possibilities to design his interface to the prototype. These functionalities include keyboard and mouse from the hardware side and KEE command language or menus from the software part.

By means of this interface, the designer can select, after the accomplished implementation of the user interface features, from the total functionality of the tool the desired one, and arrange his intended prototype. When he acts by means of command language, he operates on commands from KEE and functions written in Common Lisp; choosing menus, he can select his functionality from the menus offered at the current state of his work.

In the TODOS project, the user interface serves as an interface for the development of the prototype from the prototyping tool as well as an access possibility for a future user of the prototype in forming his individual office environment on the screen. Here

the main aim lies in interfacing components as help or explanation components, user instruction or access control mechanisms.

5.6 User interfaces and rapid prototyping

Rapid prototyping is an effective communication link between the designer and the end-user, providing a medium for exploration of the end-user's needs in regard to the man-machine interface. Integrated tools for evaluation and modification encourage this exploration, and are also valuable when updating applications.

Demonstrable prototypes for the purpose of showing the functionality of the prototype with its user interface and of getting a feedback on the user requirements should be presented in the earliest stage possible of the prototyping tool evaluation. Rapid prototyping and the simulation of the future user interface will be efficient facilities to reorganize the shape of the interface. Premature simulations, whose results and latest findings can be integrated in the modeling of the interface, are mostly more cost effective than later necessary modifications and adaptations.

The procedure of rapid prototyping for the user interface starts with the recommendations, the given facts and the analysis of the tasks to fulfill. The two former points lead to the definition of the user interface, the latter to the definition of task processes. Both definitions end up in the implementation and demonstration of the user interface by means of prototyping techniques. The demonstration can change the definition of the user interface as well as the definition of task processes but also the analysis of the tasks to fulfill. After a finite number of flows through this loop, the designer lays down the final definition of the user interface and the tasks.

In the following we will describe more deeply the designing of the TODOS user interface from the point of view of rapid prototyping. Doing that we will not direct our attention to interface components as help or explanation components, user instruction or access control mechanisms, but we will point out the importance of the user interface with respect to rapid prototyping.

In the TODOS project, we have to distinguish two interfaces:

1. *tool interface*: the interface through which the designers interact with the prototyping tool

2. *prototype interface*: The interface of the generated prototype, through which user interaction takes place during sessions with the prototype.

The tool interface may be considered part of the prototyping tool, and as such it is fixed and designed to allow for an easy access to the various tool components.

The prototype interface, on the other hand, is constructed during the prototyping phase, and may be designed individually for every single user. It serves not only as a mean for the user to interact with the prototype, but also as a model for the interface of the future OIS. Therefore this interface had to be held more flexible than the tool interface.

Designer's view

From our point of view, the designer of the prototype should profit at any time from the functionality of the user interface, therefore all access possibilities of the tool like

keyboard and mouse from the hardware side or command language and menu from the software part should be available. According to his requirements or to the most efficient application, the user should decide unrestrainedly on the use of tool access.

By means of this user interface, the designer selects from the total functionality of the tool the desired one and arranges his anticipated prototype. He operates on commands from KEE and functions in Common LISP, choosing menus, he can select his functionality from the menus offered at the time.

For the time being the designer the interface looks like as follows:

- a Lisp Listener, on which the designer communicates with the tool by means of KEE commands and LISP functions,
- an EXIT button, which allows to leave the tool,
- a window region to represent pop-up menus,
- a status area, where error messages can be displayed,
- a work space to display file contents, to configure forms or documents.

Through this interface the designer is establishing the executable functional specifications, before the designer and the user are developing the related user interface. The designer may propose a layout for the user, who then refines it, or the user is present right at the beginning, when creating layouts.

User's view

Each user must have the possibility to configure his user interface at the beginning of his work with the prototyping tool or else at the start of every work session. Of course, this work should not be lost when leaving the tool/prototype. Therefore, every user can save his interface in a sort of profile so that he will get his interface at any start of the prototype. Just as the designer, the user can choose between the mouse and the menus or the keyboard and the LISP Listener.

As the case may be, the screen can be described as follows:

- in the first case (mouse with menus), the screen looks like the user interface of the system designer, or the user arranges his screen from the offered (by default) possibilities.
  The messages or documents which have to be displayed during a prototype session appear within the "work space".
- in the second case (keyboard and LISP Listener), the user acts on the prototype by means of LISP functions or KEE commands. In a separate area, set down by the prototyping tool, a window displays messages or help texts.

The user of the prototype can use by means of the possibilities of rapid prototyping the advantages of this software technology in a twofold way:

- In the first place, the user profits from the new possibilities of an easier development of the user interface by applying rapid prototyping techniques.

- Second, the user interface developed with the prototyping tool lightens and improves the access to the prototype of TODOS.

Through this interface the user plays with the OIS prototype and watches its behavior. The behavior is also stored in a history file, for the designer to relate the change request by the user to that behavior.

6 Conclusion

Rapid prototyping is certainly a promising approach for designing an OIS. It offers a comfortable frame to model very quickly user requirements in order to get an early feedback. This does not release the designer from subsequent development steps, since a considerable designing and programming effort is left in order to convert the concept to a marketable product. In fact, rapid prototyping needs the transfer of software and tools, developed on huge and expensive AI machines, to middle-sized systems. Nevertheless rapid prototyping seems to be a promising attempt to manage the software crisis, and to narrow the software gap.

Chapter 5

Architectural Design

The design of an Office Information System (OIS), as described in Chapter 1, is seen in the TODOS Methodology as consisting of four strongly integrated phases: Requirements Collection and Analysis, Conceptual Design, Rapid Prototyping and Architecture Design.

The Architecture Design phase is a contribution of the TODOS Methodology to the area of OIS design. The aim of the Architecture Design phase is the identification of an Architecture that realizes the OIS being designed. By Architecture any set of interconnected hardware components, running software packages, is meant. Thus, rather than realizing the OIS by developing software, the TODOS methodology focuses on the use of existing software packages and emphasizes the design of an Architecture that is able to support the OIS.

To achieve its aim, the Architecture Design phase is subdivided into two stages. In the first stage, which we call Architecture Generation, a number of alternative Architectures, suitable for realizing the OIS, are identified. The input to the Architecture Generation is quantitative and qualitative information about the office activities. The qualitative information is provided by the conceptual design phase, in the form of a conceptual schema describing the data objects manipulated by the office activities and the functions abstracting such activities (see Chapter 3). The quantitative information is provided by the requirements collection and analysis phase (see Chapter 2), and integrates the conceptual schema with data such as the frequency of activities, the location of the office workers carring them out, the size of the involved data, etc. The Architecture Generation is carried out in the framework outlined by the *Architecture GEneration Methodology* (AGEM). AGEM starts with the above mentioned information and through a transformation process, in which a close interaction with the Architecture designer is required and backtracking is allowed to undo the effects of decisions taken at some previous steps, produces alternative office Architectures suitable for processing the office activities. The office Architectures are specified completely at the last step of AGEM by using the *Architecture SPEcification System* (ASPES), a tool that assists the Architecture designer in interactively defining the hardware and software constituents of the office Architecture.

In the second stage of the Architecture Design phase, called Architecture Selection, the most appropriate Architecture, among those identified in the first stage, is selected. To support this choice a performance modelling phase is proposed in which performance measures are associated to the previously generated Architectures. The performance measures delivered by the performance modelling study are twofold. On the one hand, the requirements, in terms of user-oriented performance measures (response times for and throughputs of user transactions), must be reflected. On the other hand, the modelling study must deliver system-oriented performance measures (utilization of Architectures components and queue lengths for system resources), a system designer might be interested in order to identify critical parts of a proposed Architecture. The final task of the selection process, that is the comparison of the proposed architectural solutions, takes the

performance measures into account and identifies the most appropriate Architecture on the basis of a cost-benefit tradeoff. Figure 1 schematizes this view of Architecture Design.

The three parts that comprise this chapter describe in detail the different steps of the Architecture Design phase. Part I focuses on AGEM. The concept of *Office View*, which provides a framework for representing in a uniform way the different abstraction level description of the office identified in the Architecture Generation stage, is introduced and the methodological steps of AGEM are outlined. Part II completes the description of the Generation stage, by focusing on ASPES. First an informal model of office Architecture is presented, then it is formalized as a first order theory. Finally a prototypical implementation of ASPES is described. It consists of an assistant in the Architecture Specification, that is a computer program that supports the user in the construction of a model for the given Architecture theory, and a graphical interface which makes the interaction between the Architecture designer and the program easier. Part III describes the approach to performance evaluation chosen within TODOS, that is based on queueing networks models. These models have been incorporated into the Queueing Network Analysis Package that contains a collection of resolution algorithms and a common interface for description, analysis control and result presentation.

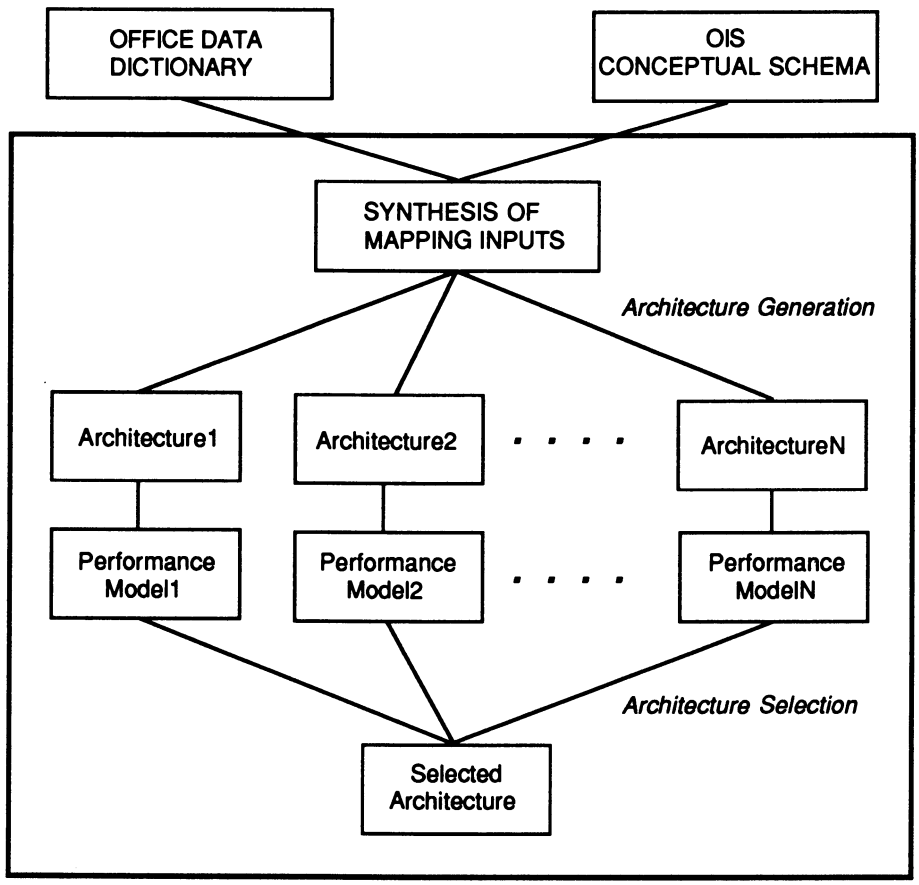


Fig. 1. - Architectural Design

Chapter 5 - Part I

Architecture Generation in TODOS

Daniela Musto

1. Introduction

The Architecture Generation in TODOS consists in the derivation of a set of alternative computer system Architectures suitable for realizing an Office Information System (OIS) from a collection of requirements and a conceptual model specified for the office. Such requirements and conceptual model are those respectively produced by the Requirements Collection and Analysis phase and by the Conceptual Design phase of the TODOS methodology, illustrated in Ch.1. The office Architectures are defined according to the model introduced in Part II of this chapter.

In order to realize the Architecture Generation, a methodology has been defined, by which one can obtain an architectural specification for the office from its abstract specification provided by the requirements and the conceptual model. Such a methodology, called in the following *Architecture GEneration Methodology* (or, shortly, AGEM), permits to derive a number of architectural solutions from the same abstract specification. All these solutions satisfy by construction the properties of the office given as input.

AGEM consists of several steps, in which a close interaction with the Architecture designer is required. At the final step the specification of the office Architecture is supported by ASPES, the tool described in Part II of this chapter. ASPES not only assists the Architecture designer in defining the hardware and software constituents of the office Architecture, but also guarantees the feasibility of the proposed Architecture, that is ASPES ensures that each Architecture derived by AGEM is composed of elements that can be physically connected together in an effective way.

This part of Ch.5 focuses on AGEM, and contains three major sections: Section 2, which introduces the basic ideas and concepts underlying AGEM; Section 3, which describes the steps of AGEM; and, finally, Section 4, which presents some concluding remarks.

2. AGEM: ideas and concepts

The Architecture Generation is realized by AGEM by consulting the Specification Database (shortly, SDB) and the Office Data Dictionary (shortly, ODD), produced by the Conceptual Design phase (see Ch.3) and by the Requirements Collection and Analysis phase (see Ch.2), respectively. SDB provides the conceptual model of the office, while ODD provides the collection of office requirements. In particular, SDB gives qualitative information concerning the office, such as the form of the activities performed in the office, and the data objects manipulated by them, while ODD gives quantitative information concerning the office, such as the periodicity of the office activities, the location

of the office workers carrying them out, the volume of the data involved, etc. ODD is also consulted for analyzing objectives and constraints for the OIS design, like the budget available for the OIS automation, the inclusion in the office Architecture of hardware and software products already used in the office, the skill or the preferences revealed by the office workers, etc.

AGEM consists of a step-wise refinement process carrying out successive transformations of an initial office description, reflecting the abstract specification of the office given as input, to produce an Architecture *scheme* and, finally, an Architecture that satisfies that scheme. The initial office description provides the main functional and non functional characteristics of the office, which are established on the basis of the informations contained in ODD and SDB. The Architecture scheme fixes the kinds (or categories) of hardware components (e.g., personal computers, laser printers, workstations, bus networks etc.), and the classes of software packages running on the computer components (e.g., graphics tools, electronic mail, spreadsheets, etc.), that will realize the office Architecture, together with their connections, their location in the office environment and their assignment to the office workers. The office Architecture is composed of compatible commercial products, and is derived from the Architecture scheme by specifying for each category of components pointed out in the scheme a particular commercial product selected among those that are available in the ASPES Catalogue for that category (see Part II of this chapter). We will say in the sequel that the Architecture *instanciates* the Architecture scheme.

At each transformation step of AGEM one specific problem concerning the mapping of the abstract specification of the office into an architectural one is tackled, and the involved decisions are emphasized. Usually a number of choices is allowed for each required decision, thus each step of AGEM leads to a potentially non empty set of alternative solutions. Possibly, some of these solutions come out to be inappropriate or useless during the next steps, and thus are afterwards rejected. If all or some of the proposed alternative solutions run with success up to the last but one step, where the Architecture scheme is defined, then different proposals of Architecture schemes become available. The Architectures derived by AGEM from the same abstract office specification may therefore differ one from the other not only for the included hardware and software instances but also for their scheme.

Remark that AGEM never accomplishes decisional steps, but asks every time the Architecture designer for that. However the fact that no decision process is allowed inside AGEM must not be taken as a strong limitation of our proposal. In fact this lack of decisionality is entirely due to the inherent complexity of the problem into consideration, which involves a large amount of knowledge and human expertise not yet formalized. In addition, AGEM provides a number of default solutions to support the Architecture designer during the Architecture Generation process.

In order to model the office during the Architecture Generation process, AGEM introduces the concept of *Office View*, that provides a framework for uniformly representing the different abstraction level descriptions of the office identified within the Architecture Generation. An *Office View* is a description of the office, and consists of the elements (called in the following *constitutive* elements) that make up the office at the abstraction level selected for the description. The working places individuated in the office, the communications occurring between them, the categories of hardware and software components included in the office Architecture, and the particular hardware and software commercial products realizing the office Architecture are all examples of constitutive elements. The constitutive elements of an *Office View* may have a complex structure (as it occurs, for example, for the hardware components), or may be described from different viewpoints

(as is the case of the office working places, for which the steps of the activities carried out on them by the office workers or the services to be provided by the architectural components in order to satisfy such activities can be specified). Therefore additional elements (called in the following *internal* elements) are included into the *Office View*, with the purpose of describing the form of the constitutive elements, whenever required. In turn these internal elements may be specified in terms of lower level descriptive elements, and so on. In the following we will represent an *Office View* as a 3-tuple $\langle \mathcal{B}, \mathcal{C}, \mathcal{D} \rangle$, where:

- $\mathcal{B}$, called the *Base*, points out the constitutive elements of the office at the selected abstraction level, and provides their description in terms of the internal elements included in $\mathcal{C}$;
- $\mathcal{C}$, called the *Catalogue*, presents the internal elements contributing to the description of the office, and possibly specifies them in terms of lower level descriptive elements included in $\mathcal{D}$;
- $\mathcal{D}$, called the *Dictionary*, presents the lower level descriptive elements considered for the office.

AGEM consists of six successive steps, and at each step an appropriate *Office View* is defined through interactions with the Architecture designer. The initial *Office View* is derived from the information contained in SDB and ODD. For each subsequent step, the *Office View* is generated by processing the one produced at the immediately preceding step. The abstraction level selected for describing the office in each *Office View* strictly depends on the step in which the *Office View* is derived. In particular, Step 0 focuses on the location of the working places in the office and on the activities performed at them; Step 1 focuses on the services to be provided by the Architecture in order to support the activities described at Step 0; Step 2 focuses on the definition of *Service Access Points* (shortly, SAPs) in the office, which provide the services indicated at Step 1; Step 3 focuses on the identification of *Subsystems* and *Networks*, defined respectively as clusters of SAPs and as clusters of Subsystems, which, according to the model of Architecture introduced in e Part II of this chapter, partially fix the scheme for the Architecture to be derived; Step 4 focuses on the internal structure of the Subsystems and the Networks identified at Step 3, defined in terms of hardware and software categories of components, which completes the definition of the Architecture scheme; finally, Step 5 specifies one or more Architectures instanciating this scheme, by using the tool ASPES.

Remark that there may be cases in which no Architecture is derivable from an Architecture scheme. This occurs either because commercial products satisfying that scheme are not present in the ASPES Catalogue, which describes the software and hardware commercial products available for realizing Architectures, or because the products in the ASPES Catalogue that satisfy the scheme are not compatible among them [1]. In these cases ASPES does not produces any output, therefore alternative schemes must be taken into account to generate an Architecture proposal. This requires a backward skip from Step 5 to Step 4 (and possibly to previous steps of AGEM): the sequence of steps for the Architecture Generation may therefore be not strictly sequential.

Since all the steps successive to Step 0 possibly produce alternative *Office Views*, the Architecture Generation may be described by a tree (called in the following *Architecture Generation Tree*, or, shortly, AGT). Each step of AGEM, in particular, will provide storing facilities to freeze some paths of AGT and backtrack facilities for avoiding its undesirable paths [2]. The root of AGT consists of the *Office View* preliminarily derived from SDB and ODD. The internal nodes of AGT are the different *Office Views* corresponding to

[1] Refer to Part II of this chapter.

[2] An undesirable path of AGT may be, for example, one for which no Architecture is derivable, due to compatibility reasons.

the intermediate steps of AGEM. The leaves of AGT are the different *Office Views* that describe office Architectures, which are expressed by using an output of ASPES.

Remark that alternative *Office Views* may be derived also at Step 0, depending on the information available in ODD and SDB. It is the Architecture designer that may propose them. Thus in the general case the Architecture Generation is more appropriately described by a forest rather than by a unique tree.

Fig. 1 schematizes AGEM, and provides an indication of the focal aspects examined at each step together with a picture representing a possible AGT.

SKELETON OF THE ARCHITECTURE GENERATION METHODOLOGY

Step 0 *location* of the working places and *activities* performed at them.
Step 1 *services* required by the activities performed at the working places.
Step 2 *Service Access Points* (SAPs).
Step 3 *Subsystems* (as cluster of SAPs) and *Networks* (as cluster of Subsystems).
Step 4 *internal structure* of Subsystems and Networks.
Step 5 office *Architectures*.

ARCHITECTURE GENERATION TREE

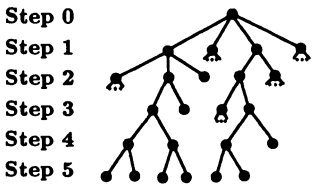


Fig. 1 - Architecture Generation Methodology

3. The Steps of AGEM

In the following we describe the steps of AGEM. For each step we present the *Office View* produced within the step and emphasize the decisions required for its definition. The formal and complete description of the *Office Views* outlined here is given in (TODOS TR 4.3.3).

3.1. Step 0

At this step the *Office View* $OV_0 = (\mathcal{B}_{OV_0}, \mathcal{C}_{OV_0}, \mathcal{D}_{OV_0})$ is produced, which represents the starting point of AGEM. OV_0 provides a high level description of the office, by specifying the location of the working places, the activities that take place at them and, furthermore, the communications and the data access required by these activities.

3.1.1. The Office View OV_0

The constitutive elements of OV_0 are the following:

(a) *Logical Working Places*;
(b) *Logical Communications*;
(c) *Logical Archives*;
(d) *Logical Data Accesses*.

A *Logical Working Place* is an abstraction for a site in the office where one or more office workers sequentially perform some activities. A *Logical Working Place* is described in terms of the office workers that have access to it, the activities that are performed at that working place and its location. Note that an office worker may have access to more than one working place in the office (this is especially true for offices that do not provide

strongly integrated electronic facilities). At the Architecture level each *Logical Working Place* will be joined to an appropriate computer, (e.g., a mainframe, a personal computer, a workstation, etc.). In order to support the activities associated to the *Logical Working Place*, the computer will run particular software packages and, possibly, will be connected to input-output or secondary-memory peripherals. The link between the *Logical Working Place* and the computer as well as the links between the computer and the peripherals will be realized either by local connections or by remote connections via a Local Area Network (LAN).

A *Logical Communication* establishes a link between *Logical Working Places*, and represents a direct communication required in the office between the corresponding working places. The *Logical Communication* is described by giving the structure of the messages exchanged. These messages may concern formatted data, text, graphics, voice, multimedia data, etc., and possibly have different structures. All the *Logical Communications* pointed out in OV_0 must be satisfied at the Architecture level. AGEM, in particular, assumes that each working place in the office is connected with every other through a telephone line. The office communications not supported by telephone (which messages can be verbally transmitted and which are not is a decision to be taken by the Architecture designer, on the basis of the structure presented by the messages [3]) will be realized in the Architecture in one of the following ways:

- via a physical cable, either as a point-to-point connection between two different computers, or as a multi-point connection established by a LAN;
- via the computer internal memory, whenever the *Logical Working Places* involved in the communication are either the same or are supported by the same computer.

In both cases appropriate software is required in order to guide the message exchange.

A *Logical Archive* is an abstraction for an office archive or a portion of it, and is described by giving the types of the data it is concerned with, their volume and their location in the office. The office Architecture will provide both the software and the hardware for the *Logical Archives*. In particular, when the scheme for the Architecture will be completely defined [4], each *Logical Archive* will be joined to a software support. This support will be either a Word Processing or a DBMS package, depending on the structure of the informations associated to the *Logical Archive*: e.g., for formatted and strongly interrelated data the DBMS would be a preferable solution, whereas for unformatted data a Word Processing would be more convenient. At the same time each *Logical Archive* will be connected to a hardware component. This hardware component will be either a computer or a sequential- or random-access secondary-memory peripheral. In the former case the hardware support for the *Logical Archive* will be the computer hard-disk; in the latter case it will be either a magnetic disk, or an optical disk, or a cartridge, etc. The hardware support will be dedicated to a single office worker or shared among different office workers, depending on whether the informations associated to the *Logical Archive* are personal or not [5]. In general more than one *Logical Archive* may be joined to the same physical support. A *Logical Archive*, in turn, may be duplicated or distributed on several physical supports, depending on the office requirements. These decisions, together with that concerning the choice between a Wordprocessing or a DBMS package, must be taken by the Architecture designer after Step 0, according to the request of AGEM.

A *Logical Data Access* establishes a link between a *Logical Working Place* and a *Logical Archive*, and represents the request to access the data corresponding to the *Logical Archive* from the working place represented by the *Logical Working Place*. The *Logical*

[3] See Step 1.

[4] See Step 4.

[5] Following Ch.2, AGEM classifies the office archives into personal, departmental and central ones.

Data Access is described in terms of the collection of data to be read or written by this working place. Each *Logical Working Place* may refer to zero, one or more *Logical Archives*; in turn, each *Logical Archive* may be connected to one or more *Logical Working Places*, depending on the office requirements. The *Logical Data Access* will be realized in the Architecture according to the modalities adopted to realize the involved *Logical Archive*: only by software, if the *Logical Archive* is supported by the same computer that provides the *Logical Working Place*, or, otherwise, with the addition of a physical link between the secondary-memory peripheral supporting the *Logical Archive* and the computer supporting the *Logical Working Place*, defined either via a point-to-point connection or via a LAN.

To summarize, the elements of $\mathcal{B}_{OV_0}$ satisfy the following properties:

- each *Logical Working Place* establishes a working place in the office, and provides its location, the activities performed on it, and the office workers realizing them.
- each *Logical Communication* establishes a communications between working places, and provides the structure of the messages exchanged between them.
- each *Logical Archive* points out a collection of permanent data in the office, and provides their structure, their volume and their location in the office.
- each *Logical Data Access* defines the access to (a portion of) an office archive from a working place, and provides the structure of the data that the working place requires to be transferred to or from the archive.

In order to complete the description of the office, OV_0 gives the following additional informations:

- for each office worker:
  - ◇ the name;
  - ◇ the set of roles (possibly more than one) that define his tasks in the office;
  - ◇ the team in which he works, if any.
- for each information utilized in the office (according to Ch.3, the informations are classified into *documents*, *messages* and *objects*, which represent, respectively, concrete documents that exists in the office real world and that are stored in the OIS, data exchanged between office workers, and abstract information manipulated within the activities and possibly stored in the OIS):
  - ◇ the structure (i.e., the number of formatted fields, of text fields, etc.);
  - ◇ the volume;
  - ◇ the life time (for stored data only).
- for each activity:
  - ◇ the set of *conceptual primitives* it involves [6], together with the kinds of information such conceptual primitives act upon (the application of a conceptual primitive to a particular kind of information will be called in the sequel *conceptual primitive call*).
  - ◇ the periodicity with which it is performed in the office;
- for each location (the general case is modelled, in which a company has several divisions dislocated in different places, in the same nation or not; at the same address the office may comprise several buildings; several office units (e.g., departments) may share the same floor; the same office unit may be distributed on different floors):
  - ◇ the address;
  - ◇ the building at the given address;
  - ◇ the floor within the building;
  - ◇ the room at the floor;

[6] See Appendix A.

◇ the particular office unit involved.

Table 1, Table 2 and Table 3 summarize, respectively, the elements of $\mathcal{B}_{OV_0}$, $\mathcal{C}_{OV_0}$ and $\mathcal{D}_{OV_0}$.

Table 1. - Elements of the Base $\mathcal{B}_{OV_0}$

P, the set of *Logical Working Places.*
A, the set of *Logical Archives.*
$P \leftrightarrow P$, the set of *Logical Communications.*
$P \leftrightarrow A$, the set of *Logical Data Accesses.*

Table 2. - Elements of the Catalogue $\mathcal{C}_{OV_0}$

W, the set of office workers in the office.
$ACTV$, the set of activities performed by the office workers.
LOC, the set of locations pointed out in the office.
$INFO$, the set of informations involved in the office,
which, in turn, is specialized in:
MSG, the set of messages;
DOC, the set of documents;
OBJ, the set of objects.
$CONCEPT.PRIM.CALL$, the set of conceptual primitive calls.

Table 3. - Elements of the Dictionary $\mathcal{D}_{OV_0}$

ROL, the set of office worker roles identified in the office.
$TEAM$, the set of office worker teams present in the office.
$ADDR$, the set of addresses concerning the office.
BLD, the set of buildings which constitute the office.
FLO, the set of floors in the buildings.
$ROOM$, the set of rooms in the buildings.
$OFF.UNIT$, the set of office units.
$INFO.FIELD$, the set of fields characterizing the structure of the office information.
$CONCEPT.PRIM$, the set of conceptual primitives.

3.1.2. Derivation of OV_0

The basic task accomplished at Step 0 in order to derive OV_0 consists in the formulation of appropriate queries to ODD and SDB, and in the rewriting the selected information according to the *Conceptual View* formalism.This task can be partially automatized. In particular, all the elements concerning the Dictionary and the Catalogue can be directly derived from ODD or SDB, while the elements of the Base must be decided by the Architecture designer, on the ground of the available data. More specifically, the elements

of OV_0 which are extracted from ODD are: ROL, TEAM, ADDR, BLD, FLO, ROOM, OFF.UNIT, included into the Dictionary, and W and LOC, included into the Catalogue. The elements of OV_0 which are extracted from SDB are: INFO.FIELD, included into the Dictionary, and ACTV, INFO, and CONCEPT.PRIM.CALL, included into the Catalogue. The CONCEPT.PRIM in the Dictionary is a predefined set, derived from the Conceptual Design phase (see Appendix A).

The Architecture designer must decide the *Logical Working Places* and the *Logical Archives*. The *Logical Communications* and the *Logical Data Accesses* follow accordingly, from the form of the office activities. In order to support the Architecture designer, AGEM provides two default solutions for the definition of the *Logical Working Places* and the *Logical Archives*. They consist, respectively, in the introduction of one *Logical Working Place* for each office worker and in the introduction of one *Logical Archive* for each office archive. This second solution, in particular, does not allow any partition of the office archives specified in ODD into independent portions within the Architecture Generation process. The first default solution simply personalizes the sites in the office where the activities must be performed, and prepares the definition of architectural solution that maximize the number of hardware equipments dedicated to individual office workers.

3.2. Step 1

At this step the *Office View* OV_0 defined at Step 0 is processed through interactions with the Architecture designer, to finally produce an *Office View* $OV_1 = (\mathcal{B}_{OV_1}, \mathcal{C}_{OV_1}, \mathcal{D}_{OV_1})$ that provides a selection of services supporting the activities described in OV_0. Such selection will guide the subsequent steps of the Architecture Generation toward the search of appropriate software packages to be installed in the office. The services taken into consideration at this step, together with their parameters, are described in Appendix B. The specification of a service by giving particular values for its parameters will be called in the sequel *service instance*.

3.2.1. The Office View OV_1

OV_1 is similar to OV_0. In particular, OV_1 provides the same categories of constitutive elements that are given in OV_0 with some little differences in the contents of $\mathcal{B}_{OV_1}$ with respect to $\mathcal{B}_{OV_0}$ (see Table 4). Some other little differences between OV_1 and OV_0 concern the Catalogue and the Dictionary. The elements of $\mathcal{C}_{OV_1}$ and $\mathcal{D}_{OV_1}$ are summarized in Table 5 and Table 6, respectively.

3.2.2. Derivation of OV_1

OV_1 is derived from OV_0 through the definition of a management policy for each information manipulated in the office, and by translating accordingly the conceptual primitive calls involved by the activities, that are specified in OV_0, into services. Step 1 requires that the Architecture designer executes the following tasks:

(T1) *Document Analysis and Archive Reorganization*, in order to decide the management policy of the documents (that is, what documents must be stored as files and what as elements of a database), and to assign to the same *Logical Archive* documents managed with the same policy.

(T2) *Message Analysis*, in order to detect the messages to be handled electronically (those messages that will be exchanged through the telephone line will be disregarded after this step of AGEM).

(T3) *Object Analysis*, in order to decide the management policy of the objects.

Table 4. - Elements of the Base $\mathcal{B}_{OV_1}$

P, the set of *Logical Working Places.*
$\underline{A}$, the set of *Logical Archives* reorganized according to the management policy decided for the documents.
$\underline{P \leftrightarrow P}$, the set of *Logical Communications* representing communications to be exchanged electronically.
$P \leftrightarrow \underline{A}$, the set of *Logical Data Accesses*, as follows from the redefinition of the *Logical Archives.*

Table 5. - Elements of the Catalogue $\mathcal{C}_{OV_1}$

W, the set of office workers working in the office.
LOC, the set of locations pointed out in the office.
$\underline{INFO}$, the set of informations involved in the office, which, in turn, is specialized in:
$\underline{MSG}$, the set of messages to be handled electronically;
$\underline{DOC}$, the set of documents with their decided management policy;
$\underline{OBJ}$, the set of objects with their decided management policy.
$SERV.REQ$, the set of service instance requests.

Table 6. - Elements of the Dictionary $\mathcal{D}_{OV_1}$

ROL, the set of office worker roles identified in the office.
$TEAM$, the set of office worker teams present in the office.
$ADDR$, the set of addresses concerning the office.
BLD, the set of buildings which constitute the office.
FLO, the set of floors in the buildings.
$ROOM$, the set of rooms in the buildings.
$OFF.UNIT$, the set of office units.
$INFO.FIELD$, the set of fields characterizing the structure of the office information.
$SERV$, the set of services.

After the accomplishment of the tasks described above, the mapping of the conceptual primitives which are required by the office activities into services can be obtained according to Table B.4 (see Appendix B), in an automatic way. In particular, the parameters values characterizing these services can be derived from the periodicity of the activities requiring the services, and from the structure of the information the conceptual primitives involved in these activities manipulate.

The solutions adopted by the Architecture designer during the execution of the three tasks indicated above depend on several factors. The most important of them are:

- for task (T1):

  for each document:

  - the form it has, that may be structured or not;

- ◇ the conceptual primitives working on it;
- ◇ the periodicity of the activities involving it;
- ◇ the number of its instances;
- ◇ the degree of access concurrence on it;
- ◇ constraints or preferences for its management policy, if provided in ODD (including the previous electronic treatment of the document, if any, in order to possibly reuse resources already employed in the office).

for each *Logical Archive*:

- ◇ the location in the office;
- ◇ the periodicity and the purpose of its access by the office workers [6].

- for task (T2):
  - ◇ the structure and the dimension of messages (for short text notices, for example, the telephone would be a preferable solution);
  - ◇ the periodicity at which messages are exchanged.
- for task (T3):
  - ◇ the structure and the dimension of the objects;
  - ◇ the conceptual primitives working on them;
  - ◇ the periodicity of the activities involving them;
  - ◇ the number of their instances.

Remark that the execution of the tasks (T1), (T2) and (T3) modifies the content of some of the sets derived from OV_0, and thus produces their redenomination in OV_1 (refer to Table 4 and Table 5, respectively).

3.3. Step 2

At this step an *Office View* $OV_2 = (\mathcal{B}_{OV_2}, \mathcal{C}_{OV_2}, \mathcal{D}_{OV_2})$ is produced which refines the description of the office provided by the *Office View* OV_1 generated at Step 1. In particular, OV_2 specifies how to distribute the requests of services identified in OV_1 into the Architecture under definition. To accomplish this task, OV_2 introduces the *Service Access Points* (SAPs), which represent locations in the office in which specific service instances are provided, and specifies how the *Logical Working Places* and the *Logical Archives* derived from OV_1 are linked to them. The selection of the *Service Access Points* that is made at this step, together with their joining to the *Logical Working Places* and to the *Logical Archives*, will give useful indications about the location of the services in the office to the next step of AGEM, where the scheme of the office Architecture will be delineated.

3.3.1. The Office View OV_2

OV_2 provides the same constitutive elements that are given in OV_1 and, in addition, the following ones:

(e) *Service Access Points* (SAPs),
which in turn are classified in:
(e1) *Computer SAPs* (C.SAPs);
(e2) *Input-Output SAPs* (IO.SAPs);
(e3) *Secondary-Memory SAPs* (SM.SAPs);
(f) *Logical Working Place to SAP Links*;

[6] The access purpose to an office archive, even if not explicitly provided in OV_0, is derivable from ODD (see Ch.2).

(g) *Logical Archive to SAP Links.*

A SAP represents a place in the office in which some services are provided with particular parameter values. Three kinds of SAPs are considered, according to the standard classification of the hardware components into computers, input-output peripherals, and secondary-memory peripherals: the C.SAPs, where the processing facilities are available, the IO.SAPs, where the inputs/outputs take place, and the SM.SAPs, where the storing facilities are supported. The SAPs are specified in terms of the service instances they provide, and their location in the office. For the SM.SAPs, in addition, the set of informations supported by them is given. Remark that the selection of SAPs does not heavily constrain the final office Architecture. However it becomes relevant in the next step of AGEM for the identification of the *Subsystems* and the *Networks* realizing the Architecture scheme: in fact, by showing where to provide the services , it points out what services should be preferably shared and what instead should remain local in the office, and, moreover, it emphasizes what are the similar service requirements among the *Logical Working Places* and the *Logical Archives.*

A *Logical Working Place to SAP Link* represents a link between a *Logical Working Place* and a SAP of kind Computer or Input-Output, asserting what service instances among those required by the *Logical Working Place* are provided by that SAP. The SAP may be linked to a single *Logical Working Place* or to more *Logical Working Places.* In both cases the SAP must satisfy the parameters values for all the services it supports: this may require an appropriate setting of the service instances provided by the SAP, in order to take into account all the requests specified by the *Logical Working Places* linked to it. In turn, a *Logical Working Place* may be connected to one SAP or to several SAPs. At least one of the SAPs, however, must be a C.SAP, since we assume that processing facilities are provided for each *Logical Working Place* (see Step 0). In the case in which more than one SAP is connected to the *Logical Working Place* three different possibilities arise:

(a) the SAPs are all of different kinds;
(b) the SAPs may be of identical kind, and possibly provide the same services, but they offer different parameter values for the replicated services.
(c) the SAPs may be of identical kind, may provide the same services, and furthermore offer identical parameter values for the replicated services.

Case (a) implies that the *Logical Working Place* is connected exactly to one *Computer SAP* and to one *Input-Output SAP.* Case (b) occurs when different requests of a service are recognized for the same *Logical Working Place*: e.g., for the service *print* one request may involve a low printing quality, to be provided by a dedicated low-price printer, and another one a high printing quality, to be provided by a laser printer shared with other *Logical Working Places.* Case (c) corresponds to the decision of the Architecture designer to insert a sort of redundancy in the office Architecture, to take into account fault-tolerance problems or a better load balance of the resources.

A *Logical Archive to SAP Link* represents a link between a *Logical Archive* and a SM.SAP, which asserts what services and what informations are supported by the SM.SAP among those required by the *Logical Working Places* linked to the *Logical Archive.* In general several *Logical Archives* may be connected to the same SM.SAP, with the constraint that the management policy selected for the data contained in them is for all the same. The SM.SAP, in this case, is shared among all these *Logical Archives*, which means that the *Logical Archives* can be grouped together in the same office location. In turn, a *Logical Archive* may be connected to more than one SM.SAP, which means that the *Logical Archive* can be distributed on several locations in the final office Architecture. Remark that the process concerning the decision of the physical allocation of the informations in-

cluded in the *Logical Archives* starts at this step of AGEM but will be entirely completed only in the subsequents steps.

Table 7 summarizes the elements of $\mathcal{B}_{OV_2}$. The elements of the Catalogue and of the Dictionary of OV_2 are the same as those provided by OV_1 (see Table 5 and Table 6, respectively).

Table 7. - Elements of the Base $\mathcal{B}_{OV_2}$

P, the set of *Logical Working Places*.
$\underline{A}$, the set of *Logical Archives* reorganized according to the management policy decided for the documents.
S, the set of *Service Access Points*,
which in turn is specialized in:
$C.S$, the set of *Computer Service Access Points*;
$IO.S$, the set of *Input-Output Service Access Points*;
$SM.S$, the set of *Secondary-Memory Service Access Points*.
$\underline{P \leftrightarrow P}$, the set of *Logical Communications* representing communications to be exchanged electronically.
$P \leftrightarrow \underline{A}$, the set of *Logical Data Accesses*, as follows from the redefinition of the *Logical Archives*.
$P \leftrightarrow S$, the set of links between *Logical Working Places* and *Service Access Points*.
$\underline{A} \leftrightarrow S$, the set of links between *Logical Archives* and *Service Access Points*.

3.3.2. Derivation of OV_2

In order to define OV_2 the Architecture designer must decide the SAPs and their links to the *Logical Working Places* and to the *Logical Archives*. The crucial decision to be taken by the Architecture designer at this step concerns where and how to serve each service request, that is if to introduce a local SAP or to associate the service request to a shared SAP, and if to distribute the service request on more than one SAP. A shared SAP will be preferably introduced each time is known that to provide the associated services expensive components are required, and also that the request of these services is not frequent in the office. On the other hand, the distribution of a service request of a *Logical Working Place* on several SAPs, that signifies that the same service can be required by the *Logical Working Place* to more than one SAP, become convenient each time a sort of redundancy is required in the Architecture for the reasons already mentioned before. Factors that may influence the choice of the Architecture designer at this step are, for example, the location of the *Logical Working Places* and the list of service instances specified for them, and the accessibility of the *Logical Archives*. Another important factor is represented by the periodicity of the service requests: in fact if requests of the same service that have all a low periodicity (and therefore a high frequency) are aggregated together in the same SAP then the resulting office Architecture will be probably affected by bottle-neck problems. Further constraints or preferences for the SAPs definition, that must be taken into account by the Architecture designer, can possibly be proposed as office requirements, and thus included in ODD.

AGEM assists the Architecture designer in the construction of OV_2 by supporting several default solutions. Among them, we recall the distributed solution which provides a dedicated C.SAP for each *Logical Working Place* and a dedicated SM.SAP for each *Logical Archive*, and the other one which proposes a partial centralized solution, by introducing

a single C.SAP for each office unit, or alternatively, for each floor, to be shared by all the *Logical Working Places* located on the office unit, or on the floor, respectively. In addition AGEM assists the Architecture designer by controlling that the service requests of each *Logical Working Place* are distributed among the SAPs in such a way that these requests are completely covered by the service instances provided by the accessible SAPs, and, furthermore, that the *Logical Data Accesses* produced in OV_1 are preserved by the definition of the SM.SAPs.

3.4. Step 3

At this step the *Office View* OV_2 defined in Step 2 is augmented through interactions with the Architecture designer, in order to produce an *Office View* $OV_3 = (\mathcal{B}_{OV_3}, \mathcal{C}_{OV_3}, \mathcal{D}_{OV_3})$ which provides the location of the *Subsystems* and *Networks* in the office. OV_3 partially fixes the scheme for the Architecture to be defined. Remark that this is the lowest step of AGEM where the physical office Architecture is taken into account, and thus the decisions taken at this step have a strong impact on the form of the office Architecture (see next steps).

3.4.1. The Office View OV_3

The constitutive elements of OV_3 are the same as those given in OV_2 with the addition of the following ones, reflecting the model of office Architecture presented in Part II of this chapter:

(h) *Subsystems*;
(i) *Networks*.

A *Subsystem* represents the simplest form of aggregation of resources occurring in the Architecture, which consists of resources connected point-to-point to each other. The *Subsystems* are defined in OV_3 as clusters of SAPs which realize a partition on the set of SAPs derived from OV_2.

A *Network* represents a remote aggregation of resources in the office, which is realized through a LAN. The *Networks* are defined in OV_3 as clusters of *Subsystems*. Each *Network* includes at least two *Subsystems*. In general a *Subsystem* may belong to more than one *Network* [7], which makes the *Networks* communicating each other. The *Subsystem* in this case becomes a place-holder for a gateway to be installed in the office Architecture.

Table 8 summarizes the elements of $\mathcal{B}_{OV_3}$. The elements of the Catalogue and of the Dictionary of OV_3 are the same as those provided by OV_1 (see Table 5 and Table 6, respectively).

3.4.2. Derivation of OV_3

In order to define OV_3 the Architecture designer must identify the *Subsystems* and the *Networks* that define the office Architecture. To realize this task the Architecture designer must decide what services to make locally accessible and what accessible in remote, and thus which SAPs to aggregate together into a single *Subsystem* and which *Subsystems* to connect together into a common *Network*. The specification of the internal structure of the derived *Subsystems* and *Networks* in terms of hardware and software components supporting the SAPs will be realized in the next step of AGEM, on the basis of the informations collected at this step.

[7] An office Architecture may includes also isolated *Subsystems*, according to the technological level selected for realizing the office. But in the sequel we shall only consider *Subsystems* integrated in the office.

Table 8. - Elements of the Base $\mathcal{B}_{OV_3}$

P, the set of *Logical Working Places*.
$\underline{A}$, the set of *Logical Archives* reorganized according to the management policy decided for the documents.
S, the set of *Service Access Points*, which in turn is specialized in:
$C.S$, the set of *Computer Service Access Points*;
$IO.S$, the set of *Input-Output Service Access Points*;
$SM.S$, the set of *Secondary- Memory Service Access Points*.
$SUBSYS$, the set of *Subsystems*.
$NETW$, the set of *Networks*.
$\underline{P \leftrightarrow P}$, the set of *Logical Communications* representing communications to be exchanged electronically.
$P \leftrightarrow \underline{A}$, the set of *Logical Data Accesses*, as follows from the redefinition of the *Logical Archives*.
$P \leftrightarrow S$, the set of links between *Logical Working Places* and *Service Access Points*.
$\underline{A} \leftrightarrow S$, the set of links between *Logical Archives* and *Service Access Points*.

Crucial factors that influence the definition of the *Subsystems* are, for example, the distance between the SAPs in the office, the service they provide, the number of *Logical Working Places* sharing the same SAPs, etc. In particular, SAPs that are far one from the other usually become part of independent *Subsystems*, in order to take into account technological hardware constraints. The same holds for SAPs that are shared by several *Logical Working Places*, for identical reasons [8]. Moreover, the examination of the services provided by the SAPs may give suggestions on how to partition the SAPs into *Subsystems*: for example, it may be natural to associate SM.SAPs with C.SAPs into the same *Subsystems*, since usually computers offer both processing and storing facilities.

The *Networks* must be defined by the Architecture designer in such a way that the *Logical Communications* and the *Logical Data Accesses* derived from OV_3 are all preserved. This fact is controlled by AGEM on the basis of the links established in OV_3 among *Logical Working Places*, *Logical Archives* and SAPs. Remark, in particular, that *Subsystems* collected into the same *Network* are supposed to be communicating. *Subsystems* related to different *Networks* can also communicate each other, provided that these *Networks* share a common *Subsystem*.

3.5. Step 4

At this step the *Office View* OV_3 defined at Step 3 is processed through interactions with the Architecture designer, in order to produce an *Office View* $OV_4 = (\mathcal{B}_{OV_4}, \mathcal{C}_{OV_4}, \mathcal{D}_{OV_4})$ which specifies for each *Subsystem* and *Network* the classes of hardware and software components they consists of. The Architecture scheme so derived will be instanciated with commercial products in the next and final step of AGEM, which will conclude the Architecture Generation process.

3.5.1. The Office View OV_4

[8] In fact it is obvious that if a SAP service is required by many *Logical Working Places* the component that will provide the service in the office Architecture cannot be linked in a point-to-point manner with all the components supporting the *Logical Working Places*.

The constitutive elements of OV_4 are the following:

(a) *Logical Working Places*;
(b) *Logical Communications*;
(c) *Logical Archives*;
(d) *Logical Data Accesses*;
(e) *Architectural Units*;
(f) *Subsystems*;
(g) *Networks*;
(h) *Logical Working Place to Architectural Unit Links*;
(i) *Logical Archive to Architectural Unit Links*;
(l) *Architectural Unit to Architectural Unit Links*;
(m) *Architectural Unit to Network Links.*

The *Logical Working Places*, the *Logical Communications*, the *Logical Archives* and the *Logical Data Accesses* are those derived from OV_3.

An *Architectural Unit* represents the basic component of the office Architecture, and is described in terms of a hardware products category available in the ASPES Catalogue. Each *Architectural Unit* which is a computer, in addition, is equipped with a set of software products categories, representing the classes of packages that run on that unit in the final office Architecture.

The *Subsystems* and the *Networks* are the same as those identified in OV_3, but a different specification is given for them, which completes the definition of the Architecture scheme started at Step 3. In particular, each *Subsystem* is defined as a set of *Architectural Units* point-to-point connected each other. Each *Network* is seen as a LAN with a given topology.

A *Logical Working Place to Architectural Unit Link* establishes a link between a *Logical Working Place* and an *Architectural Unit*, which means that the *Architectural Unit* will support the *Logical Working Place* in the office. This link is described by specifying what service requirements among those identified for the *Logical Working Place* are provided by the *Architectural Unit.*

A *Logical Archive to Architectural Unit Link* establishes a link between a *Logical Archive* and an *Architectural Unit.* This link means that the *Architectural Unit*, which is in this case a secondary-memory unit, will support the *Logical Archive* in the office. The link is described by specifying what informations among those included in the *Logical Archive* are stored in the *Architectural Unit.*

An *Architectural Unit to Architectural Unit Link* establishes a point-to-point connection between *Architectural Units* included into the same *Subsystem.*

An *Architectural Unit to Network Link* establishes a connection between an *Architectural Unit* and a *Network.* The *Architectural Unit* is, in this case, a computer, and connects the *Subsystem* to which it belongs to the *Network.*

Table 9 summarizes the elements of $\mathcal{B}_{OV_4}$. The Catalogue of OV_4 extends the Catalogue of OV_3 in order to support the specification of the Architecture scheme. The elements of $\mathcal{C}_{OV_4}$ are shown in Table 10. The elements of the Dictionary of OV_4 are the same as those provided by OV_1 (see Table 6).

3.5.2. Derivation of OV_4

In order to define OV_4 the Architecture designer must decide the internal structure of the *Subsystems* and the *Networks*, and must establish accordingly the connections among the *Logical Working Places*, the *Logical Archives* and the *Architectural Units* occurring in the office.

Table 9 - Elements of the Base $\mathcal{B}_{OV_4}$

P, the set of *Logical Working Places*.
$\underline{A}$, the set of *Logical Archives* reorganized according to the management policy decided for the documents.
$ARCH.UNIT$, the set of *Architectural Units*.
$SUBSYS$, the set of *Subsystems*.
$NETW$, the set of *Networks*.
$\underline{P \leftrightarrow P}$, the set of *Logical Communications* representing communications to be exchanged electronically.
$P \leftrightarrow \underline{A}$, the set of *Logical Data Accesses*, as follows from the redefinition of the *Logical Archives*.
$P \leftrightarrow ARCH.UNIT$, the set of links between *Logical Working Places* and *Architectural Units*.
$\underline{A} \leftrightarrow ARCH.UNIT$, the set of links between *Logical Archives* and *Architectural Units*.
$ARCH.UNIT \leftrightarrow ARCH.UNIT$, the set of links between *Architectural Units*.
$ARCH.UNIT \leftrightarrow NETW$, the set of links between *Architectural Units* and *Networks*.

Table 10 - Elements of the Catalogue $\mathcal{C}_{OV_4}$

W, the set of office workers working in the office.
LOC, the set of locations pointed out in the office.
$\underline{INFO}$, the set of informations involved in the office, which, in turn, is specialized in
$\underline{MSG}$, the set of messages to be handled electronically;
$\underline{DOC}$, the set of documents with their decided management policy;
$\underline{OBJ}$, the set of objects with their decided management policy.
$SERV.REQ$, the set of service instance requests.
$ASPES.CATALOGUE.SCHEME$, the set of hardware and software products categories that are specified in the ASPES Catalogue.

A crucial task required by these decisions is the mapping of the SAPs derived from OV_3 into *Architectural Units*. In general, an *Architectural Unit* may correspond to one or more *SAPs*. This last case represents the fact that the *Architectural Unit* offers several access points to the services it provides. The decisions concerning this correspondence are affected by several factors: among them, the functionality requirements associated to the SAPs, the workload, cost constraints, preferences concerning the hardware to be installed in the office that are expressed by the office workers, the availability of products already used in the office (if any), etc. All these informations are provided by ODD. Remark that the decisions taken at this step can also be influenced by the results of the performance evaluation produced by the Architecture Selection stage (see Part III of this chapter). To realize a SAP, in turn, several *Architectural Units* may be required: this case occurs, for example, whenever components like terminals, network servers and line switchers must be introduced in the *Subsystems* to appropriately support the characteristics of the office that are specified in OV_3.

Another crucial task to be accomplished at this step concerns the selection of the categories of software products that the computers individuated in the Architecture must run in such a way that the service instances associated to the SAPs are provided in the office. AGEM supports the Architecture designer in performing this task, by producing tables like those presented in Appendix B (e.g., see Table B.2).

3.6. Step 5

At this step the *Office View* OV_4 defined at Step 4 is processed through interactions with the Architecture designer, in order to produce an *Office View* $OV_5 = (\mathcal{B}_{OV_5}, \mathcal{C}_{OV_5}, \mathcal{D}_{OV_5})$ which instanciates the scheme of the Architecture provided by OV_4 with particular hardware and software commercial products. This step is supported by the tool ASPES and concludes the Architecture Generation process.

3.6.1. The Office View OV_5

The *Office View* OV_5 specify an Architecture for the office. OV_5 provides the same categories of constitutive elements that are given in OV_4 (see Table 9). The Catalogue of OV_5 is similar to the Catalogue of OV_4, but $\mathcal{C}_{OV_5}$ contains the whole ASPES Catalogue. The elements of $\mathcal{C}_{OV_5}$ are summarized in Table 11. The elements of the Dictionary of OV_5 are the same as those provided by OV_4 (see Table 6).

Table 11 - Elements of the Catalogue $\mathcal{C}_{OV_5}$

W, the set of office workers working in the office.
LOC, the set of locations pointed out in the office.
$\underline{INFO}$, the set of informations involved in the office,
which, in turn, is specialized in
$\underline{MSG}$, the set of messages to be handled electronically;
$\underline{DOC}$, the set of documents with their decided management policy;
$\underline{OBJ}$, the set of objects with their decided management policy.
$SERV.REQ$, the set of service requests.
$ASPES.CATALOGUE$, the ASPES Catalogue.

3.6.2. Derivation of OV_5

In order to derive OV_5 the Architecture designer must instanciate the Architecture scheme provided by OV_4 with commercial compatible products. This task is accomplished with the support of the tool ASPES, which assists the Architecture designer in interactively defining the particular components to be installed in the office and their mutual connections. The ASPES Catalogue provides a repository of the commercial products that are currently available to realize office Architectures, and gives a description of their allowed expansions. ASPES controls that the particular hardware products included in the Architecture, possibly expanded with respect to their basic commercial version, are compatible among them and that can be physically connected together in an effective way. In addition, ASPES controls that the Architecture is equipped with compatible software, and that the appropriate network software is installed wherever remote connections must be supported.

Among the crucial factors that the Architecture designer must take into account at this step, we recall the ones already mentioned at Step 4, and, in addition, the following ones: the role of the office workers, the preferences of the office workers and the teams to which they belong, the quantitative parameters associated to the service instances required in the office, the characteristics of the particular commercial products. Suggestions for the definition of the office Architecture can be derived by the Architecture designer also at this step from the results of the Architecture Selection stage.

4. Conclusions

In this paper we have presented a methodology for generating alternative computer system Architectures appropriate for realizing an OIS. This methodology, called Architecture GEneration Methodology (or, shortly, AGEM), receives as input a collection of functional and non functional informations concerning the office, and produces as outputs Architectures that satisfy the input specification.

Each Architecture is generated by AGEM through a step-wise refinement process that carries out successive transformations of a description of the office preliminary derived from the contents of two databases, respectively providing a collection of requirements and the conceptual model of the office. These transformations produce different views of the office, each view emphasizing some particular aspects in a coherent way with the others and presenting the office at a different abstraction level. The final view of the office derived by AGEM provides an Architecture that satisfies all the properties of the office given as input.

The major contributions of AGEM are:

(1) the identification of six focal steps toward which distributing all the decisions required to generate an Architecture satisfying the office needs;

and

(2) the introduction of a descriptive framework for representing the office which allows a uniform definition of the different abstraction level descriptions of the office produced within the Architecture Generation process.

In order to be applied AGEM requires a close interactions with the Architecture designer: AGEM supports the correctness of the transformation steps and provides some default solutions to the Architecture designer, but the Architecture designer is responsible for all the decisions that must be taken within the Architecture Generation process. In particular, at each step of AGEM the Architecture designer can either adopt one or more default solutions (if any), or autonomously define one or more solutions, or, finally, try both ways. AGEM guides the Architecture Generation process and controls that all the solutions adopted by the Architecture designer fit well with the office view currently under examination.

AGEM has been devised as an interactive tool which assists the Architecture designer in the definition of Architectures appropriate for the OIS. The lack of decisionality of AGEM is entirely due to the complexity of the application domain, which involves a large amount of knowledge and human expertise not yet formalized. Therefore it must not be taken as a strong limitation of our proposal. An application of AGEM to the test case illustrated in Ch.6 is sketched in (TODOS TR 4.3.4).

Presently no automatic tool implementing AGEM is available, but the last step of AGEM is supported by the automatic tool ASPES, which implementation is described in Part II of this chapter. ASPES assists the Architecture designer in the specification of the components of the office Architecture, and of the connections among them. A tool implementing AGEM would provide an assistance which extends that provided by ASPES to aspects that concern the satisfaction of the office requirements, so that the matching of the Architecture with such requirements become not integrally demanded to the Architecture designer. It seems to us that a tool implementing AGEM can be realized through a straightforward extension of the tool ASPES, provided that the performance of this realization can be maintained within acceptable values in terms of both space and time factors. In fact AGEM adopts the same model of office Architecture that is used by ASPES, and integrates this model with additional informations concerning the spatial location of the architectural components in the office and their assignment to the office

workers. Moreover an implementational framework analogous to that used to represent office Architectures can also be used to represent the other views of the office produced within the Architecture Generation process, due to the uniform approach adopted by AGEM to describe all the views of the office taken into consideration. The results of a preliminary investigation concerning the feasibility of such an automatic tool are shown in (TODOS TR 4.3.3).

5. Appendix A: Conceptual Primitives

This appendix focuses on the conceptual primitives that have been identified within TODOS to specify the office activities (refer to Ch. 3), and which apply to the informations manipulated in the office (documents, messages, etc.). The list of conceptual primitives is presented in Table A.1. These conceptual primitives represent the basic steps of the office activities, and can be combined together to define complex steps by using control structures, like decision structures, looping structures, etc.

In order to generate office Architectures, the conceptual primitives listed in Table A.1 must be mapped into services to be provided by the components of the Architecture being defined [7].

Table A.1 - Conceptual Primitives

COMPUTE
COPY
CREATE
EDIT
MODIFY
PRINT
REFER
REMOVE
RETRIEVE
TRANSMIT

6. Appendix B: Services

This appendix focuses on the services that support the work done in the office. A service is a facility offered by a software product running on an appropriate hardware which is constituted by at least one computer and, possibly, by additional input-output and secondary-memory peripherals; furthermore it is equipped with all the software packages required to run that particular software product. Remark that the selection of the software and hardware products to be included in the Architecture in order to satisfy the office requirements is one of the crucial points of the Architecture Generation, and that another important problem consists just in the identification of the services that are suitable for implementing the conceptual primitives characterizing the office activities.

In the following we will concentrate on the basic services provided by the application software, since they seem to us the most significative ones for the Architecture Generation. Therefore we will disregard in the sequel both the services provided by the operating system and other basic software, and the services provided by the advanced application

[7] See Appendix B.

packages (like time schedule, project planning, etc.). The basic software, in fact, constitutes the standard software equipment for computers, and hence it does not require any choice from the Architecture designer during the Architecture Generation process. On the other hand, the inclusion of advanced application packages in the software equipment of the computers of the Architecture improves the facilities offered by the Architecture but does not influence the definition of its form.

The application software covers different categories of tools, that we shall call in the sequel *Office Support Tools*. Table B.1 gives a list of these categories, while Table B.2 sketches a list of services provided by each of them. Remark that the class LANGUAGE that appears in Table B.1 provides the union of all the facilities offered by the other Office Support Tools (these facilities, in fact, can all be obtained through the construction of appropriate programs), thus, without entering into details concerning the selection of the most appropriate commercial languages for the office needs, a LANGUAGE represents the default solution for every requirement of application software in the office. A list of parameters that characterize the services presented in Table B.2 is shown in Table B.3.

Table B.1 - Office Support Tools

LANGUAGE
WORD PROCESSING
DBMS
GRAPHICS TOOL
SPREADSHEET
ELECTRONIC MAIL
OFFICE ADVANCED TOOL

Table B.2 - Services offered by the Office Support Tools

LANGUAGE		
(the union of all the facilities listed below)		
WORD PROCESSING		
create file	**delete file**	**modify file**
edit file	**retrieve file**	**spell file**
display file	**copy file**	**print file**
DBMS		
create database-record-type	**delete database-record-type**	**modify database-record-type**
edit database-record-type	**create database-entry**	**delete database-entry**
modify database-entry	**edit database-entry**	**display database-report**
print database-report	**query database**	
GRAPHICS TOOL		
edit graphic-data	**process graphic-data**	**visualize graphic-data**
SPREADSHEET		
edit spreadsheet	**process spreadsheet**	**visualize spreadsheet**
ELECTRONIC MAIL		
send/receive		
OFFICE ADVANCED TOOL		
(advanced facilities: not considered here)		

While performing the Architecture Generation a mapping between the conceptual primitives proposed in Appendix A and the services pointed out in this appendix is required in order to bridge the gap existing between them, which is originated by the different abstraction levels they are concerned with when modelling the office. A list of possible correspondences is proposed in Table B.4. Note that the established mapping is one-to-many: in fact the same conceptual primitive can be translated into several different services (see, e.g., EDIT), according to the management policy decided for the information to which it applies. The selection of the management policy concerns Step 1 of AGEM, and is left to the Architecture designer.

Table B.3 - Parameters characterizing the Services offered by the Office Support Tools

copy file (*file-size*, *periodicity*)
create database-entry (*data-size*, *periodicity*)
create database-record-type (*data-size*, *periodicity*)
create file (*file-size*, *periodicity*)
delete database-entry (*data-size*, *periodicity*)
delete database-record-type (*data-size*, *periodicity*)
delete file (*file-size*, *periodicity*)
display database-report (*data-size*, *periodicity*)
display file (*file-size*, *periodicity*)
edit database-entry (*data-size*, *periodicity*)
edit database-record-type (*file-size*, *periodicity*)
edit file (*file-size*, *periodicity*)
edit graphic-data (*data-size*, *periodicity*)
edit spreadsheet (*data-size*, *periodicity*)
modify database-entry (*data-size*, *periodicity*)
modify database-record-type (*data-size*, *periodicity*)
modify file (*file-size*, *file-percentage*, *periodicity*)
print database-report (*data-size*, *periodicity*)
print file (*file-size*, *quality*, *periodicity*)
process graphic-data (*data-size*, *periodicity*)
process spreadsheet (*data-size*, *periodicity*)
query database (*data-size*, *periodicity*)
retrieve file (*file-size*, *periodicity*)
send/receive (*message-size*, *periodicity*)
spell file (*file-size*, *periodicity*)
visualize graphic-data (*data-size*, *periodicity*)
visualize spreadsheet (*data-size*, *periodicity*)

Table B.4 - Correspondence Table between Conceptual Primitives and Services

COMPUTE	COPY	CREATE
process graphic-data	**copy file**	**create database-entry**
process spreadsheet	**query database**	**create database-record-type**
spell file	**edit graphic-data**	**create file**
	edit spreadsheet	**edit graphic-data**
		edit spreadsheet
EDIT	MODIFY	PRINT
display database-report	**modify database-entry**	**print database-report**
display file	**modify database-record-type**	**print file**
edit database-entry	**modify file**	**visualize graphic-data**
edit database-record-type	**process graphic-data**	**visualize spreadsheet**
edit file	**process spreadsheet**	
edit graphic-data		
edit spreadsheet		
REFER	REMOVE	RETRIEVE
retrieve file	**delete database-entry**	**query database**
query database	**delete database-record-type**	**retrieve file**
edit graphic-data	**delete file**	**edit graphic-data**
edit spreadsheet		**edit spreadsheet**
TRANSMIT		
send/receive		

Chapter 5 - Part II

Architecture Specification in TODOS

Donatella Castelli, Carlo Meghini, and Daniela Musto

1. Introduction

This paper introduces the Architecture Specification System (ASPES), a computer tool which assists the designer in last step of the Architecture Generation phase of the TODOS methodology. ASPES provides an interactive environment, accessible through a graphical interface, to incrementally define the specific hardware and software machinery to be used in the office architecture being designed. In addition, ASPES guarantees that the architecture being incrementally constructed by the designer is feasible, that is it consists of components that can be effectively combined together.

ASPES is implemented as a knowledge based system which embodies knowledge on commercial hardware and software components and on the possible ways of combining these components into components of increasing complexity, up to the level of office architectures. Such knowledge is represented and manipulated via a procedural semantic network, a powerful data structure which enriches the expressive power of semantic networks with procedural attachment. The knowledge of ASPES is formally specified as first order theory, called Architecture Specification Theory (AST), whose intended models are the office architectures definable by means of ASPES. Such models must satisfy two basic requirements: on the one hand they must be able to represent real world Architectures whose functionality can be matched with the architectural constraints implicitly stated in the conceptual schema and in the requirements database; on the other hand they must represent architectures at the level of detail suitable to represent the conditions of compatibility and connectability among architectural components.

The paper consists of three major parts: in Section 2, the model which inspires AST is informally introduced, by illustrating the concepts that have been thought as relevant in representing Architectures. This model is then formalized in Section 3, as a first order theory, whose intended models are office Architectures, and represent the formal counterparts of the Architecture concepts previously introduced. For space reasons, the formalization in Section 3 regards only a part of the informal model, but a significant one, which includes all the relevant aspects of Architectures. In Section 4, a prototypical implementation of ASPES is described. The implementation consists of an assistant in Architecture specification, that is a computer program that supports the user in the construction of a model for our Architecture theory, and of a graphical interface, which makes the interaction between the user and this program easier.

2. A Model of Office Architectures: Informal View

Our model views an office Architecture as consisting of interconnected hardware components, supporting the operations of software components. A hardware component is any

physical device that can be employed in an information system Architecture; computers, input/output peripherals, local area networks are typical hardware components. Software components are the software packages that run on the computers of Architectures, performing tasks that determine the functionality of hardware components. The connections among hardware components may be point-to-point or multipoint. A point-to-point connection is a physical link between two hardware components that allows the communication between them. A multipoint connection enables the communication among several hardware components, so establishing a computer network.

In modelling multipoint connections, we restrict to Local Networks, which typically provide interconnection of a variety of data communicating devices within a small area. Furthermore, among the various categories of Local Networks that have been proposed (Stalling 1984), our model includes only Local Area Networks, which are the most appropriate for the kind of communication under consideration. Therefore, from a structural viewpoint, Architectures, in their most general form, are modelled as consisting of Local Area Network which communicate between each other through common hosts. In order to avoid any confusion between a Local Area Networks seen as a set of hosts, and the hardware machinery that is employed to establish such a network (cables, receivers, absorbers and the like), we will use the term 'Office Network' for the former, while reserving the word 'LAN' for the latter. An Office Network is a set of hosts, or Subsystems, where a Subsystem is constituted by one or more computer Units, which may be point-to-point connected to peripheral Units. This view determines a top-down decomposition of Architectures through four levels:

(1) Architecture level, where Architectures are defined in terms of Office Networks;
(2) Office Network level, which we will simply call Network level, where Office Networks are defined as sets of interconnected Subsystems;
(3) Subsystem level, comprising the definition of Subsystems as sets of point-to-point connected hardware Units; and
(4) Unit level, the lowest level, where Units are defined as instances of commercial hardware and software components.

The set of commercial hardware and software components whose instances can be employed in an Architecture is included in the Catalogue. The Catalogue also contains the relationships among components which are relevant in the definition of Architectures, such as, for instance, compatibility relationships. Any element of the Catalogue that is to be included in an Architecture must then first be instanciated[1] through an apposite Unit. Since the same architectural component may appear in several different places of an Architecture, the instance relation between the Catalogue elements and Units is clearly one-to-many: many Units may be instances ofthe same Catalogue object, and a Unit must be instance of exactly one Catalogue element.

2.1. Architectures

Architectures may be of two kinds: simple Architectures, which consist of just one Subsystem, and Complex Architectures, which consist of a (non empty) set of communicating Office Networks. Simple Architectures capture the special case of an office information system supported by point-to-point connected devices. This may be the case of a single workstation directly connected with few peripheral devices, as well as that of a mainframe supporting a number of more or less intelligent terminals. In general, an Architecture is expected to be of the Complex kind, with its constituting Office Networks corresponding to the branches of the office.

[1] We use the term 'instanciate' as a synonym of 'create an instance'.

2.2. Office Networks

Office Networks can be characterized in terms of topology, which may be a bus, ring, or tree topology. A Subsystem is connected to an Office Network through an 'escape Unit', that is a computer Unit of the Subsystem which is directly linked to the Office Network. Of this connection, we abstract the details concerning transceivers and transceiver cables, as they are not relevant to our framework. There is, however, a notion of compatibility in LAN connections that must be represented, as not any computer Unit can be connected to any LAN. To this end, we introduce the concept of network interface and associate it with those of computer and LAN, as follows: a computer has a set of network interfaces, whereas a LAN requires one of a set of network interfaces. Now we can define a computer to be compatible with (hence connectable to) a LAN if it has, or can have by expansion, at least one of the network interfaces required by the LAN. The basic configuration of a particular computer may not have a certain network interface, but the computer may be later expanded in order to acquire that interface, and this must be taken into account in defining the notion of network compatibility. We will return on the concept of computer expansion when detailing the Unit level of our decomposition.

2.3. Subsystems

Subsystems are hosts of Office Networks, and model the simplest form of aggregation of Units that can be found in an Architecture, i.e. that established through point-to-point connections. The presence of at least one computer in a Subsystem guarantees the 'autonomy' of the Subsystem. In fact, whether the Subsystem is a host of an Office Network, or it stands alone, thus representing the simplest form of Architecture, it must necessarily have a computer in order to be able to operate. Another important constraint on the definition of a Subsystem is that there be no isolated Units within the Subsystem. This constraint can be expressed more formally by viewing a Subsystem as an undirected graph whose nodes represent the Units and whose arcs represents the point-to-point connections of the Subsystem, and imposing a condition, that we call seriality, on the graph, which says that there must be a path from any node of the graph to any other node.

Similarly to LAN connections, point-to-point connections are represented by introducing the concept of external interface: a point-to-point connection between two hardware Units is established by relating an appropriate external interface of each Unit. There is of course a notion of compatibility also at the Subsystem level, which is stated in terms of point-to-point connectable external interfaces. Since an external interface is always associated to exactly one hardware component, this is equivalent to declare which components may be linked to each other when making up a Subsystem. As already pointed out, point-to-point connections may in general be established either between computers or between a computer and any kind of peripheral; the only allowed peripheral-peripheral connection is that between magnetic disks.

2.4. Units

A Unit is a member of a Subsystem and an instance of one Catalogue object. There is one kind of Unit for each kind of objects that are in the Catalogue, and one relationship between Units for each Catalogue relationship that is structurally relevant in the construction of an Architecture. For instance, the relationship between a certain computer and its network interfaces is relevant in the construction of Architectures, as it describes the ability of the computer to participate in an Office Network; thus an instance of that computer will be associated to an instance of each one of its interfaces through a relation that mirrors at the Unit level the Catalogue relation between computers and their network

interfaces. The same applies to external interfaces. In other words, a Unit is given all the relevant properties of the Catalogue object that it instanciates. In addition, a computer Unit may acquire further properties by being expanded.

Expanding a computer Unit means to increase one of the computer's functionalities by adding an appropriate device to the computer. This device may be a hardware component or a software package. In the former case, we have a hardware expansion, whereas in the latter we have a software expansion. The notion of expansion captures an operation that the current technology has made very common in the practice of computer systems configuration. For this reason, a considerable number of expansions are currently possible. We have limited the hardware expansions to the following kinds:

- coprocessor, that is the installation on a computer of an additional CPU;
- main memory;
- diskette, that is the addition of a diskette drive;
- fixed disk, that is the addition of fixed disk drive or the augmentation of the storage capacity of an existing fixed disk drive;
- cartridge, similar to the diskette expansion;
- external interface;
- network interface.

Regardless of its kind, a hardware expansion is modelled through the concepts of expansion slot and expansion board. In particular, each computer of the Catalogue is associated (via an apposite relation) with a set of expansion slots; each of these slots may be used for an expansion, that is it can be occupied by an expansion board, where an expansion board is a Catalogue object which bears some other component(s). Which expansion boards can be placed on which expansion slots is told by hardware expansion compatibilities, which associate expansion slots to the expansion boards that can be installed on them. Although the information needed for performing expansions is contained in the Catalogue, an expansion is a notion associated to a Unit, hence a concept of the Unit level.

The representation of software expansions requires less conceptual machinery, as the installation of a program on a computer does not involve the manipulation of any physical device. In fact, the only relation needed to model software expansions in the Catalogue is one which states software expansion compatibilities; these are richer than their hardware counterparts, as each of them describes: the operating system, the amount of main memory and the set of software packages that are required in order to install a certain software package on a certain computer. A software expansion is performed as an hardware expansion, that is by associating via an apposite relation the expanded computer with the installed software package.

2.5. The Catalogue

We have already introduced all the elements that comprise the Catalogue. To summarize, the Catalogue consists of two parts. The first maintain the hardware and software components whose instances are used to build Architectures, along with the relationships among such components. For instance, computers, their network and external interfaces, and their expansion slots are in this part of the Catalogue.

The second part of the Catalogue gives the compatibilities among components, and is consulted in the construction of higher level architectural objects, like expanded Units and Subsystems. As we have seen, there are the following kinds of compatibilities:

(a) point-to-point compatibilities, stating which external interface can be connected to which external interface in order to establish a point-to-point connection between two hardware devices;

(b) hardware expansion compatibilities, asserting which boards can be placed on which slots in performing a hardware expansion to a computer;
(c) software expansion compatibilities, describing under which conditions a software package can be installed on a given computer.

3. A Theory of Architectures

In this section we will present a formalization of the concepts introduced in the previous section by defining the Architecture Specification Theory (AST), a first order theory whose intended models are office information system Architectures. The first order language of our theory of Architectures is called Architecture Specification Language (ASL). The ASL symbols, grouped by level they refer to, are given in Appendix A, together with an informal description of the intended meaning of each of them. In the next section, we will give the proper axioms of our theory of Architectures.

3.1. Axiom System

As for the definition of the predicate symbols of the language in Appendix A, we will group the axioms of our Architecture theory by level, and provide a brief and informal explanation when required. In order to improve the readability, in this section we will list only some of the AST axioms. The missing axioms are given in Appendix B. As we have set- and number-theoretic symbols in our language, we should provide axioms also for number theory and set theory. To avoid the repetition of well-known axiom systems, we assume that one of the proposed axiomatizations of formal number theory and set theory be included in the logical axioms.

3.1.1. Catalogue Axioms

The Catalogue axioms concern the hardware and software components whose instances are to be used in the construction of Architectures, and the compatibility of these components.

The most important among hardware components are computers and peripherals, whose properties are given by the following axioms:

$$(\forall x)(Computer(x) \supset (\exists x_1 x_2 x_3 x_4 x_5 x_6 x_7)(Has\_Slot(x, x_1) \wedge Has\_External\_Interface(x, x_2) \wedge Has\_Network\_Interface(x, x_3) \wedge Has\_Memory(x, x_4) \wedge Has\_Peripheral(x, x_5) \wedge Has\_Operating\_System(x, x_6) \wedge Has\_Software(x, x_7)))$$

$$(\forall x)(Peripheral(x) \supset (\exists x_1) Has\_External\_Interface(x, x_1))$$

The first axiom says that a computer must necessarily be associated to an expansion slots set, an external interfaces set, a network interfaces set, an amount of main memory, a peripherals set, an operating system, and a software packages set. The second axiom states that only external interfaces are to be associated to peripherals.

Each property may have a unique value, and this is enforced by the property uniqueness axioms, an example of which is:

$$(\forall xy)(Has\_Slot(x, y) \supset (\forall z)(Has\_Slot(x, z) \supset Same\_Set(y, z)))$$

where $Same\_Set(y, z)$ is true if and only if y and z denote the same set.

The other property uniqueness axioms can be found in Appendix B. An expansion slot may not belong to more than one computer, and the same applies to external and network interfaces, and to peripherals. The next axiom, and the similar ones in Appendix B, express the disjointness of the appropriate sets:

$$(\forall x_1 x_2 x_3 x_4)((Has\_Slot(x_1, x_3) \wedge Has\_Slot(x_2, x_4)) \supset (\exists y)((Member(y, x_3) \wedge Member(y, x_4)) \supset (x_1 = x_2)))$$

The other hardware components of the Catalogue are expansion boards, defined as follows:

$$(\forall x)(Board(x) \supset (\exists x_1)(Bears\_External\_Interface(x, x_1) \otimes Bears\_Network\_Interface(x, x_1) \otimes Bears\_Memory(x, x_1)))$$

where the exclusive or connective is used to express the fact that an expansion board is only one of external interface, network interface, main memory expansion board.

As far as compatibility is concerned, it has been already pointed out that there are three kinds of compatibility: expansion, point-to-point, and network compatibility. An expansion compatibility may be hardware or software. The former are declared through the *Hw_Expansion_Compatible* predicate. In order to guarantee the meaningfulness of hardware expansion compatibilities, it must be enforced that the involved slot effectively belongs to the involved computer, as stated by the axiom:

$$(\forall xyz)(Hw\_Expansion\_Compatible(x, y, z) \supset (\exists u)(Has\_Slot(x, u) \wedge Member(y, u)))$$

where x stands for a computer, y for a slot and z for an expansion board. Software expansion compatibilities are a bit richer assertions, which involve the operating system, the amount of main memory, and the software packages that a computer must have in order to be expanded with a software package. As a consequence, the meaningfulness of a software compatibility is more complex to express. It amounts to say that: (a) the computer is required to support a specific operating system; (b) the computer must have at least the amount of memory required by the compatibility, or it must be expandable to reach such amount; and (c) the computer must have all the software packages required by the compatibility, or it must be software expandable to acquire such packages. Here is the axiom:

$$\begin{aligned}&(\forall x_1x_2x_3x_4x_5)(Sw\_Expansion\_Compatible(x_1, x_2, x_3, x_4, x_5) \supset \\ &\quad Has\_Operating\_System(x_1, x_3) \wedge \\ &\quad ((\exists y)(Has\_Memory(x_1, y) \vee ((\exists zu)Hw\_Expansion\_Compatible(x_1, z, u) \wedge \\ &\qquad Bears\_Memory(u, y)) \wedge (y \geq x_4))) \wedge \\ &\quad (\forall v)(Member(v, x_5) \supset ((\exists w)(Has\_Software(x_1, w) \wedge Member(v, w)) \vee \\ &\qquad (\exists t)Sw\_Expansion\_Compatible(x_1, v, x_3, x_4, t)))))\end{aligned}$$

We have used indentation to help the reading of this axiom, whose second line corresponds to point (a) above, third and fourth to point (b), and fifth and sixth to point (c).

The point-to-point compatibility is expressed by the *PTP_Compatible* predicate, whose first and third argument are the connectable devices, i.e. either a computer or a peripheral, whereas the second and fourth arguments are the external interfaces to be used in the connection. Point-to-point compatibilities clearly enjoy symmetry:

$$(\forall x_1x_2x_3x_4)(PTP\_Compatible(x_1, x_2, x_3, x_4) \supset PTP\_Compatible(x_3, x_4, x_1, x_2))$$

and make sense only if the involved computers or peripherals have, or can acquire by expansion, the involved external interfaces:

$$\begin{aligned}&(\forall x_1x_2x_3x_4)(PTP\_Compatible(x_1, x_2, x_3, x_4) \supset \\ &\quad ((\exists y)(Has\_External\_Interface(x_1, y) \wedge Member(x_2, y)) \vee \\ &\qquad (\exists yzw)(Hw\_Expansion\_Compatible(x_1, y, z) \wedge \\ &\qquad\quad (Bears\_External\_Interface(z, w) \wedge Member(x_2, w)))) \wedge \\ &\quad ((\exists y)(Has\_External\_Interface(x_3, y) \wedge Member(x_4, y)) \vee \\ &\qquad (\exists yzw)(Hw\_Expansion\_Compatible(x_3, y, z) \wedge \\ &\qquad\quad (Bears\_External\_Interface(z, w) \wedge Member(x_4, w)))))\end{aligned}$$

The third kind of compatibilities are network compatibilities, which assert, by means of the *Network_Compatible* predicate, that a computer can be connected to a Local Area Network. As for the compatibilities encountered so far, network compatibilities have a

special axiom that enforces their meaningfulness. In particular, the network compatibility axiom guarantees that the involved computer has, or can acquire by expansion, the required network interface and software package:

$$(\forall x_1 x_2 x_3 x_4)(Network\_Compatible(x_1, x_2, x_3, x_4) \supset$$
$$(((\exists y)(Has\_Network\_Interface(x_1, y) \wedge Member(x_3, y)) \vee$$
$$(\exists vwt)(Hw\_Expansion\_Compatible(x_1, w, v) \wedge$$
$$(Bears\_Network\_Interface(v, t) \wedge Member(x_3, t)))) \wedge$$
$$((\exists z)(Has\_Software(x, z) \wedge Member(x_4, z)) \vee$$
$$(\exists urs)Sw\_Expansion\_Compatible(x_1, x_4, u, r, s))))$$

The Catalogue axioms are completed by set-typing axioms, which can be found in Appendix B.

3.1.2. Unit Axioms

Units are instances of Catalogue objects, therefore the language provides one Unit predicate symbol for each Catalogue object predicate symbol. The relationship between these two predicate types is given by the instanciation axioms, that is axioms like the following:

$$(\forall x)(Computer\_Unit(x) \supset (\exists x_1)(Computer(x_1) \wedge Instance(x_1, x)))$$

The axioms like the previous one guarantee that any Unit is an instance of at least one Catalogue object. In order to avoid the case where a Unit is an instance of two or more Catalogue objects, the following axiom is needed:

$$(\forall xy)(Instance(x, y) \supset (\forall z)(Instance(z, y) \supset (x = z)))$$

Sets of Units mirror at the Unit level the corresponding sets of Catalogue objects.The set-typing axioms in Appendix B guarantee that the member of Unit sets are in fact objects of the appropriate type.

The definitions of computer, peripheral and board Units reflect those given at the Catalogue level:

$$(\forall x)(Computer\_Unit(x) \supset (\exists x_1 x_2 x_3 x_4 x_5 x_6 x_7)(Has\_Slot\_Unit(x, x_1) \wedge$$
$$Has\_External\_Interface\_Unit(x, x_2) \wedge$$
$$Has\_Network\_Interface\_Unit(x, x_3) \wedge Has\_Total\_Memory(x, x_4) \wedge$$
$$Has\_Peripheral\_Unit(x, x_5) \wedge Has\_Operating\_System\_Unit(x, x_6) \wedge$$
$$Has\_Software\_Unit(x, x_7)))$$
$$(\forall x)(Peripheral\_Unit(x) \supset (\exists x_1) Has\_External\_Interface\_Unit(x, x_1))$$
$$(\forall x)(Board\_Unit(x) \supset (\exists y)(Add\_External\_Interface(x, y) \otimes Add\_Memory(x, y) \otimes$$
$$Add\_Network\_Interface(x, y)))$$

The predicate *Add_External_Interface* for board Units is analogous to the predicate *Bears_External_Interface* for Catalogue boards, in that it associates to an external interface board Unit the external interfaces that it brings when installed on a computer Unit. The same applies to the *Add_Network_Interface* and *Add_Memory* predicates.

The slot Unit set that is associated to a computer Unit upon instanciation, cannot have as members arbitrary slot Units, rather its members must be one to one instances of the slots associated to the Catalogue computer being instanciated. The same applied for the peripheral Unit set and the Operating System of a Computer Unit. The next axiom expresses this condition on the slot Unit set:

$$(\forall xy)(Has\_Slot\_Unit(x, y) \equiv (\exists x_1 x_2)((Has\_Slot(x_1, x_2) \wedge (Instance(x_1, x)) \wedge$$
$$(\forall x_3 x_4)(Instance(x_3, x_4) \supset (Member(x_3, x_2) \equiv Member(x_4, y))))))$$

The same condition must be expressed for the external interface Units associated to a computer or peripheral Unit, with the exception that an interface Unit may be also acquired by expansion. Thus, the external interfaces of a Unit are those given upon instan-

ciation (indicated by the predicate $Standard\_External\_Interface$), and those acquired by expansion (indicated by $Added\_External\_Interface$):

$$(\forall xy)(Has\_External\_Interface\_Unit(x,y) \equiv (\exists y_1 y_2)(Standard\_External\_Interface(x,y_1) \wedge Added\_External\_Interface(x,y_2) \wedge (y = (y_1 \cup y_2))))$$

The external interface Units given upon instanciation are one to one with the Catalogue's external interfaces of the object being instanciated:

$$(\forall xy)(Standard\_External\_Interface(x,y) \equiv (\exists x_1 x_2)(Instance(x_1,x) \wedge Has\_External\_Interface(x_1,x_2) \wedge (\forall x_4 x_5)(Instance(x_4,x_5) \supset (Member(x_4,x_3) \equiv Member(x_5,y)))))$$

The external interface Units that the Unit has gained by expansions are collected from the external interface expansions made to the Unit in question (expansions are explained later):

$$(\forall xy)(Added\_External\_Interface(x,y) \equiv (\forall x_1)(Member(x_1,y) \supset (\exists x_2 x_3 x_4)(Hw\_Expanded(x,x_2,x_3) \wedge Member(x_1,x_4) \wedge Add\_External\_Interface(x_3,x_4))))$$

The Units brought by an external interface expansion board Unit must be instances of the external interfaces brought by the Catalogue expansion board.

$$(\forall xy)(Add\_External\_Interface(x,y) \supset (\exists zu)(Board(z) \wedge Instance(z,x) \wedge Bears\_External\_Interface(z,u) \wedge (\forall x_1 x_2)(Instance(x_1,x_2) \supset (Member(x_1,u) \equiv Member(x_2,y)))))$$

Similar axioms hold for the network interface Units, the software Units and the main memory of a computer Unit.

Finally, we have to deal with hardware and software expansions of computer Units. Introducing the informal model, we have spoken of seven kinds of hardware expansions, but we will deal here only with three kinds: main memory, external interface, and network interface expansion. These expansions are somewhat more important than the others because they directly affect the possibility of building a larger number of Architectures with the same machinery. In fact, by acquiring main memory, a computer Unit is able to acquire new software by being software expanded, whereas by acquiring an external (network) interface it gains the possibility of being connected to other computer or peripheral Units (Office Networks). The axioms for the other hardware expansions quoted in the informal model and not dealt with (coprocessor, diskette, cartridge and fixed disk expansions), would be a more or less obvious application of the principles being described here.

Hardware expansions are represented by instances of the $Hw\_Expanded$ predicate. Two constraints must be expressed on hardware expansions: first, the same slot may be used only for one expansion, which is to say that an expansion can be made only if the involved slot has not been used for another expansion:

$$(\forall xyz)(Hw\_Expanded(x,y,z) \supset (\forall x_1)(Hw\_Expanded(x,y,x_1) \supset (x_1 = z)))$$

Second, an expansion may be made only if there is an appropriate expansion compatibility among the Catalogue objects whose instances are the Units involved in the expansion:

$$(\forall xyz)(Hw\_Expanded(x,y,z) \supset (\exists x_1 x_2 x_3)(Hw\_Expansion\_Compatible(x_1,x_2,x_3) \wedge Instance(x_1,x) \wedge Instance(x_2,y) \wedge Instance(x_3,z)))$$

Software expansions are stated through the $Sw\_Expanded$ predicate, which relates a computer Unit to the software package Unit with which it is being expanded. A software

expansion is legal if there is a software compatibility in the Catalogue involving the appropriate computer and software package, such that: (a) the computer and software package Units are instances of the computer and software package that appear in the software compatibility; (b) the operating system Unit of the computer Unit under expansion is an instance of the operating system that appears in the software compatibility; (c) the total memory of the computer Unit under expansion is greater than or equal to the amount of memory required by the expansion; (d) the computer Unit has among its software package Units at least one instance of each software package required in the compatibility. The next axiom expresses this constraint, by stating condition (a) above in its third line, condition (b) in the fourth, (c) in the fifth, and (d) in the last two lines.

$$(\forall xy)(Sw\_Expanded(x,y) \supset$$
$$(\exists zuvx_1x_2x_3x_4x_5)(Sw\_Expansion\_Compatible(x_1,x_2,x_3,x_4,x_5) \wedge$$
$$Instance(x_1,x) \wedge Instance(x_2,y) \wedge$$
$$Has\_Operating\_System\_Unit(x,z) \wedge Instance(x_3,z) \wedge$$
$$Has\_Total\_Memory(x,u) \wedge (u \geq x_4) \wedge$$
$$(\forall x_6)(Member(x_6,x_5) \supset (\exists t)(Instance(x_6,t) \wedge$$
$$Has\_Software\_Unit(x,v) \wedge Member(t,v))))))$$

3.1.3. Subsystem Axioms

A Subsystem has been (informally) defined as consisting of a non empty set of point-to-point connected computer and peripheral Units, among which there is at least one computer Unit. As a consequence of this definition, it turns out that a computer Unit is in fact a Subsystem, although a 'simple' one, since it has no point-to-point connections. Subsystems with one or more such connections may then be thought of as 'complex' Subsystems. To this end, we have included in the language two predicate symbols to represent Subsystems: one is *Subsystem*, which applies to all Subsystems, and the other is *Complex_Subsystem*, which applies only to Subsystems with at least one point-to-point connection. Both computer Units and Complex Subsystems are Subsystems, as stated by the next axiom:

$$(\forall x)(Subsystem(x) \supset (Computer\_Unit(x) \otimes Complex\_Subsystem(x)))$$

The definition of a Complex Subsystem is given by the following axiom:

$$(\forall x)(Complex\_Subsystem(x) \supset (\exists x_1)(Has\_Unit(x,x_1) \wedge Serially\_Connected(x_1)))$$

The predicate *Has_Unit* relates a Complex Subsystem to the set of Units that constitute it. A number of constraints hold for the set of Units of a Complex Subsystem. First, only one such set can be associated to a Complex Subsystem:

$$(\forall xy)(Has\_Unit(x,y) \supset (\forall z)Has\_Unit(x,z)) \supset Same\_Set(y,z))$$

Second, since two Subsystems are not allowed to share Units, all the Unit sets of Subsystems must be disjoint:

$$(\forall xzx_1x_2)((Has\_Unit(x_1,x) \wedge Has\_Unit(x_2,z)) \supset (\exists y)((Member(y,x) \wedge$$
$$Member(y,z)) \supset (x_1 = x_2)))$$

Third, the Units that are members of a Complex Subsystem may only be computer or peripheral Units:

$$(\forall xy)(Has\_Unit(x,y) \supset (\forall x_1)(Member(x_1,y) \supset (Computer\_Unit(x_1)$$
$$\vee Peripheral\_Unit(x_1))))$$

Last, at least one of such Units must be a computer Unit:

$$(\forall xy)(Has\_Unit(x,y) \supset (\exists x_1)(Member(x_1,y) \wedge Computer\_Unit(x_1)))$$

The Units of a Complex Subsystem must be serially connected:

$$(\forall xy)(Has\_Unit(x,y) \supset Serially\_Connected(y))$$

that is, any Unit of a subsystem can be reached from any other Unit by following point-to-point connections. This condition avoids the partition of the Complex Subsystem into isolated fragments, and is guaranteed by the following axiom, which just says that any pair of Units of a Unit set must be reachable from each other:

$(\forall x)(Serially\_Connected(x) \equiv (\forall x_1 x_2)((Member(x_1, x) \wedge Member(x_2, x)) \supset Reachable(x_1, x_2))))$

In the simplest case, two Units are reachable if they are point-to-point connected:

$(\forall xyuv)(PTP\_Connected(x, y, u, v) \supset Reachable(x, u))$

Furthermore, two Units are reachable if they are point-to-point connected to the same Unit, which is to say that the reachability relation is transitive. It is clearly also reflexive and symmetric, hence the following axioms hold:

$(\forall x)\ Reachable(x, x)$

$(\forall xy)(Reachable(x, y) \supset Reachable(y, x))$

$(\forall xyz)((Reachable(x, y) \wedge Reachable(y, z)) \supset Reachable(x, z))$

We have already introduced the *PTP_Connected* predicate, which relates two point-to-point connect Units, and the external interface Units directly involved in the connection. The external interface Units that appear in a point-to-point connection must belong to the external interface Unit set of the connected Units:

$(\forall xyuv)(PTP\_Connected(x, y, u, v) \supset (\exists x_1 x_2)(Has\_External\_Interface\_Unit(x, x_1) \wedge Has\_External\_Interface\_Unit(u, x_2) \wedge Member(y, x_1) \wedge Member(v, x_2)))$

Between two Units there cannot be more than one point-to-point connection:

$(\forall xyuv)(PTP\_Connected(x, y, u, v) \supset (\forall zw)(PTP\_Connected(x, z, u, w) \supset ((z = y) \wedge (v = w))))$

A point-to-point connection is bidirectional:

$(\forall xyuv)(PTP\_Connected(x, y, u, v) \supset PTP\_Connected(u, v, x, y))$

Finally, a point-to-point connection may be established only if there is a compatibility assertion between the involved computer and/or peripheral Units. As point-to-point compatibility is stated in terms of Catalogue objects, the axiom that enforces the compatibility condition is:

$(\forall xyuv)(PTP\_Connected(x, y, u, v) \supset (\exists x_1 x_2 x_3 x_4)(Instance(x_1, x) \wedge Instance(x_2, u) \wedge Instance(x_3, y) \wedge Instance(x_4, v) \wedge PTP\_Compatible(x_1, x_3, x_2, x_4)))$

3.1.4. Network Axioms

Network axioms concern the Office Networks that appear in an office Architecture. These Networks consist of three parts: the LAN Unit, which is an instance of the hardware machinery employed in the realization of the Network; the set of Subsystems which constitute the Network; and the set of connections, called LAN-connections, from an element of each Subsystem to the Network Unit. This is reflected by the axioms that defines Office Networks:

$(\forall x)(Office\_Network(x) \supset (\exists x_1 x_2)(Has\_LAN(x, x_2) \wedge Has\_Subsystem(x, x_1) \wedge Hosts(x_1, x_2)))$

The *Has_LAN* predicate relates an Office Network to the LAN Unit upon which the Network is realized. No two LAN Units may be used to realize the same Office Network:

$(\forall xy)(Has\_LAN(x, y) \supset (\forall z)(Has\_LAN(x, z) \supset (y = z)))$

The predicate *Has_Subsystem* relates an Office Network to the set of Subsystems that constitute the Network. As for LAN Units, no two Subsystem sets may belong to the same Network:

$(\forall xy)(Has\_Subsystem(x,y) \supset (\forall z)(Has\_Subsystem(x,z) \supset (y = z)))$

The *Hosts* predicate is used in the language to associate the set of Subsystems of an Office Network with the Network's LAN Unit. This connection is made explicit through the *LAN_Connected* predicate, which is used in the axiom below to state which property must satisfy each member of a Subsystem set:

$(\forall xyz)((Hosts(x,y) \wedge Member(z,x)) \equiv (\exists uv)LAN\_Connected(z,u,v,y))$

Strictly speaking, only the third and fourth argument would have sufficed to represent a LAN connection; these are the network interface and the LAN Unit which are at the two ends of the connection. To enrich the information carried by instances of the *LAN_Connected* predicate, we have included also the specification of the Subsystem and of the computer Unit. This redundancy makes necessary two consistency conditions. First, the computer Unit must be a Unit of the Subsystem. Second, the network interface Unit used in the connection must belong to the computer Unit. These two conditions are expressed, in the order, by the next two axioms.

$(\forall xyzu)(LAN\_Connected(x,y,u,z) \supset (\exists x_1)(Has\_Unit(x,x_1) \wedge Member(y,x_1)))$
$(\forall xyzu)(LAN\_Connected(x,y,u,z) \supset$
$\quad (\exists x_1)(Has\_Network\_Interface\_Unit(y,x_1) \wedge Member(u,x_1)))$

No two computer Units of the same Subsystem may be connected to a LAN Unit:

$(\forall xyzuvw)((LAN\_Connected(x,y,v,z) \wedge LAN\_Connected(x,u,w,z)) \supset (y = u))$

The same network interface Unit may not be used for more than one LAN connection:

$(\forall xyzvw)((LAN\_Connected(x,y,v,z) \wedge LAN\_Connected(x,y,w,z)) \supset (v = w))$

Finally, in order to establish a LAN connection, the appropriate network compatibility must hold among the Catalogue objects whose instances are used in the connection, and the involved computer must have, among its software package Units, an instance of the required software package:

$(\forall xyzu)(LAN\_Connected(x,y,z,u) \supset (\exists x_1x_2x_3x_4v)(Instance(x_1,y) \wedge$
$\quad Instance(x_2,z) \wedge Instance(x_3,u) \wedge Has\_Software(y,v) \wedge$
$\quad Member(w,v) \wedge Instance(x_4,v) \wedge Network\_Compatible(x_1,x_2,x_3,x_4)))$

3.1.5. Architecture Axioms

An Architecture has been defined as a set of zero or more Office Networks, which communicate through common Subsystems. An Architecture with zero Office Networks is meant to consist of just a single Subsystem. In order to represent this situation, our Architecture specification language distinguishes between two kinds of Architectures: general Architectures, represented via the *Architecture* predicate, which applies to all the legal Architectures; and Architectures with at least one Office Network, represented through the *Complex_Architecture* predicate. The relationships among these two kinds of Architectures and Subsystems is given by the following axiom:

$(\forall x)(Architecture(x) \supset (Subsystem(x) \otimes Complex\_Architecture(x)))$

A Complex Architecture is made up by a set of Office Networks sharing Subsystems for communication:

$(\forall x)(Complex\_Architecture(x) \supset (\exists x_1)(Has\_Network(x,x_1) \wedge Communicate(x_1)))$

The *Has_Network* predicate associates a Complex Architecture with the set of Office Networks that constitute it. Clearly, an Architecture can have only one set of Office Networks:

$(\forall xy)(Has\_Network(x,y) \supset (\forall z)Has\_Network(x,z) \supset (y = z)))$

whereas two different Architectures cannot have the same Office Network set:

$$(\forall xyzv)((Has\_Network(x,y) \wedge Has\_Network(z,v)) \supset (\exists w)((Member(w,y) \wedge Member(w,v)) \supset (x = z)))$$

The Office Networks of an Architecture must communicate:

$$(\forall x)(Has\_Network(x,y) \supset Communicate(y))$$

that is, any two Networks of an Architecture must be reachable from each other, via common Subsystems:

$$(\forall x)(Communicate(x) \equiv (\forall x_1 x_2)((Member(x_1,x) \wedge Member(x_2,x)) \supset Connected(x_1,x_2))))$$

The behaviour of the *Connected* predicate is defined analogously to that of the *Reachable* predicate between Subsystems. In the simplest case, two Office Networks communicate if they share at least one Subsystem:

$$(\forall xy)(Office\_Network(x) \wedge Office\_Network(y) \wedge (\exists x_1 x_2 z)(Has\_Subsystem(x,x_1) \wedge Has\_Subsystem(x,x_2) \wedge Member(z,x_1) \wedge Member(z,x_2)) \supset Connected(x,y))$$

The general case is handled by stating the transitivity of the *Connected* predicate (symmetry and reflexivity follow from the previous axiom):

$$(\forall xyz)((Connected(x,y) \wedge Connected(y,z)) \supset Connected(x,z))$$

4. Architecture Specification System

The Architecture Specification System (ASPES) is a computer program that supports the definition of Architectures as models of the Architecture Specification Theory. The user of the current implementation of ASPES is the designer of the office system, who incrementally makes up the Architecture of the office being designed by interacting with the ASPES interface. By doing so, the designer will obtain a formal specification of his Architecture, being at the same time guaranteed on the feasibility of such Architecture with respect to the compatibility of the employed hardware and software components. Such formal specification will be used either as an unambiguous description of the future office Architecture and as input to the Architecture selection phase.

This section is structured as follows: first, a brief description of the implementation language that has been employed to realize ASPES is be given. Then the structure of an ASPES knowledge base will be sketched, along with the operations that are provided to construct an Architecture. Finally, the ASPES user interface is illustrated. A presentation of ASPES implementation can be found in (Castelli *et al.* 1988). For a more detailed exposition, see (TODOS TR 4.3.4).

4.1. The Implementation Language

PSN (Levesque and Mylopoulos 1979) is a knowledge representation language that formalizes traditional semantic network concepts within a procedural framework. PSN provides the mechanisms for representing and manipulating objects and binary relationships between them, according to the modelling principles of object oriented languages. These principles can be summarized as follows:

(1) there is a one-to-one correspondence between the objects in the reality being modelled (in our case office information system Architectures) and the model objects, and between the real world (binary) relationships and the model relationships;

(2) three of these relationships are factored out and used as abstraction mechanisms that permit the organization of the knowledge in the model; they are:

(2.1) the instance-of relationship, corresponding to the classification abstraction mechanism, by which objects with common properties are gathered into classes; an object is then an "instance-of" the class it belongs to; in turn, classes may be instances of metaclasses. Metaclasses help the organization of meta level knowledge, allowing, among other things, the definition of properties of classes.

(2.2) the part-of relationship, corresponding to the aggregation abstraction mechanism, by which an object is seen as the aggregate of the objects which are related to it; these relations can be further divided into structural (the ones that "constitute" the object, also called properties), and assertional (those that merely make an assertion about the object, and can be later retracted) (Woods 1975);

(2.3) the is-a relationship, corresponding to the specialization abstraction mechanism, by which a class of object is seen as a special case (or subclass) of another class; the former class is then "is-a" the latter.

The three abstraction mechanisms interact with each other by means of inheritance: a subclass inherits all the properties defined by its superclasses, whereas an instance of a subclass is also an instance of its superclasses.

Four operations are possible on PSN classes:

- create an instance of a class, realized by the to-put procedure associated to the class;
- remove an instance from a class, performed by the class to-rem procedure;
- get all the instances of a class, for which the to-get procedure is defined;
- test whether an object is an instance of a class, corresponding to the to-test procedure.

These procedures give the semantics of the class, as they interpret the class structure in the intended way. The PSN interpreter provides a standard procedure for each of the four operations, to be used when the semantics of a class is standard. However, a class may be given a non standard semantics by specifying *ad hoc* programs to perform one or more of its four operations. This feature gives to PSN classes the flavor of abstract data types [2], and turns out to be an elegant way of embodying procedural knowledge into a knowledge base.

4.2. ASPES Knowledge Bases

In order to support the construction of models of AST, ASPES relies on a number of procedures that enforce the constraints corresponding to the axioms of AST upon a given interpretation of ASL. These procedures carry out their task by manipulating the relations that interpret the ASL predicate symbols. The collection of the PSN definitions of the axiom enforcement procedures, of the relations interpreting the ASL predicates, and the relationships between these two, is the ASPES Kernel Knowledge Base. An Architecture, i.e. an AST model, is an ASPES Knowledge Base (KB), which is generated from the kernel KB by providing an extension (or state), that is a set of tuples to each relation definition, that is by specifying which are the constant values that satisfy the corresponding predicate. Different Architectures are obtained from the kernel KB by varying the extension of the interpreting relations.

As we have seen in the previous section, a unary relation can be naturally represented in PSN as the extension of a class, i.e. as the set of the instances of a class. Therefore, every ASL unary predicate symbol is interpreted in an ASPES KB by the extension of a PSN class, which we call the class interpreting that predicate. In order to maintain an intuitive link between ASL and its interpretation, a PSN class is given the same name as the predicate symbol that it interprets. But to avoid any confusion between the language and its interpretation, we will use for class names a font different from that of predicate

[2] PSN does not have the concept of type

symbols. For instance, objects representing computers are represented in an ASPES KB as instances of the class `Computer`, which interprets the ASL predicate symbol *Computer*.

The PSN class mechanism has also been used to circumvent the semantic networks idiosyncrasy in representing relations with an arity greater than two. The problem has arisen in our context when giving the semantics to n-places predicate symbols, with n greater than two, like for instance the *Hw_Expansion_Compatible* and the *Hw_Expanded* predicate symbols, both 3-placed. In these cases, a class with n slots has been used, each slot corresponding to one and only one position of the n-place predicate symbol interpreted by the class.

Two-place predicate symbols of ASL are represented in PSN in two different ways: as properties (or PSN slots), if such predicate symbols must be interpreted as representing structural attributes of components; as assertions (or PSN relations), if they are to be intended as representing time varying statements. For instance, as we want to give a structural meaning to the *Has_External_Interface* predicate symbol, relating a computer to the set of its external interfaces, the slot `Has_External_Interface` has been defined for the class `Computer`. On the contrary, as the set of the external interface Units of a computer Unit may vary due to the fact that some of these interface Units may be used for point-to-point connections, the interpretation of the *Has_External_Interface_Unit* predicate symbol has been realized as a relation having the class `Computer_Unit` as domain and the class `External_Interface_Unit` as range.

The enforcement of the constraints which are the model theoretic counterparts of the AST axioms, is performed in two ways: either implicitly by the PSN interpreter, for those constraints that turn out to be inherent to PSN, or explicitly by a PSN program, for the other constraints. Of course, by PSN program we mean one of the four programs that can be attached to a PSN class. The constraints corresponding to typing axioms for two-places predicates fall in the first category, and are automatically enforced by the PSN interpreter, as these predicates are represented as PSN links (whether slots or relationships), whose specification correctness is checked by the PSN interpreter. The same can be said for the constraints corresponding to some axioms on the uniqueness of attribute values: if such attributes are represented as PSN slots, then the PSN interpreter guarantees that only one value is given for such slots. Instead, there is no such automatic checking for relations, so, when required, the uniqueness of a pair must be explicitly controlled by the non-standard "to-put" procedure associated to the involved assertion class. This is also the case of all the non-inherent constraints, which, as we will see later in this section, are usually checked by non-standard "to-put" procedures.

So far we have described how an interpretation of AST is realized in PSN, and how it is handled by PSN programs in order to guarantee that it is in fact a model of AST. But, as we have already mentioned, in order to maintain the theory at a reasonable level of compactness and readability we have presented a simplified version of AST. An ASPES KB is so much richer in semantics than a model of the illustrated theory. In particular in addition to what has been presented so far, an ASPES KB contains also:

(1) a set of properties defined for hardware and software components classes, which we call descriptive properties. Descriptive properties are not involved in the construction of an Architecture, but are there in order to give more information on hardware and software components to the ASPES user. An examples of a descriptive property is the `Has_Display` property, which associates a computer with its display. Of course, the introduction of such properties requires the introduction the classes where they range, like the class `Display` required by the `Has_Display` property. Both descriptive properties and their range classes can be formalized in AST as the other properties and classes of an ASPES KB, e.g. as binary and unary predicates with the appropriate typing and uniqueness axioms;

(2) a subdivision of components and Units for representing more specific functionalities. Computers, for example, are divided in **Personal_Computer**, **Network_Server**, **Word_Processor**, **Integrated_Workstation**, **Mini_Computer**, **Main_Frame**. This level of granularity of knowledge is required in order to have a meaningful representation of Architectures with respect to the aims of Architecture Design within the TODOS methodology. This subdivision can be formalized proof theoretically by axioms that have the form of an implication, whose antecedent is the unary predicate representing the less general relation, and whose consequent is the unary predicate representing the more general relation (Reiter 1984). The PSN is-a relation allows to easily implement these refinement relations by partially ordering the classes in a is-a hierarchy according to their functionality, with the most general class representing the least specified functionality. The introduction of the is-a hierarchy requires the introduction of new classes, which can be understood as partial specifications of the is-a leaves.

4.3. Implementational Remarks

In this section we will present some implementation details of the ASPES kernel knowledge base in order to illustrate the use of PSN in implementing an Architecture builder system.

A PSN metaclass is a class whose instances are classes. Metaclasses have been introduced in the ASPES KBs for two reasons. First, they have been used as "handles" for set of classes; this turns out to be very useful in dealing with higher order functions, as the enforcement of constraints like 'there must exist a compatibility class such that ...'. The second reason is that metaclasses allow the definition of properties of classes, which are the only place in a PSN KB to record information about collections of objects. The is-a hierarchy at the metaclass level is shown in Figure 1.

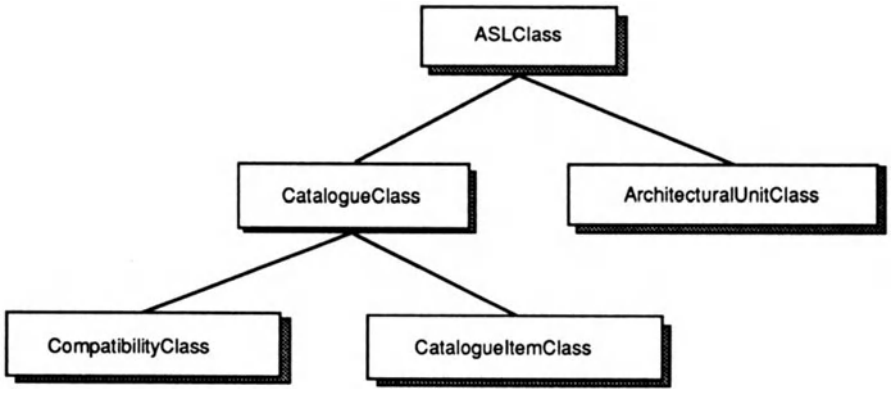


Fig. 1. - Metaclass level is-a hierarchy

The most general ASL metaclass is **ASL_Class**, which is specialized into **CatalogueClass** and **Architectural_Unit_Class**. The former is the metaclass of the classes constituting the Catalogue, while the latter has as instances the classes corresponding to the four levels of Architecture abstraction (Unit, Subsystem, Network, and Architecture). In turn, **Catalogue_Class** has two specializations: **Compatibility_Class**, the metaclass of all compatibility classes, and **Catalogue_Item_Class** the metaclass of classes modelling Catalogue components.

As we have already mentioned, the PSN procedure mechanism has been used to enforce constraints on the Architecture models. The four programs attached to a class are, when necessary, appropriately redefined to give the correct semantics to the class. This is, for example, the case of the `PTP_Compatibility` classes whose semantics is given by their to-test procedure. In general, the to-test procedure of a point-to-point compatibility class receives as inputs two hardware units and searches the class extensions to see whether there exists an instance asserting the compatibility of their base models. In doing so, the procedure tests also if the given units have still available the external interfaces of the type required for the connection, that is if they have not already been used in a previous connection. Such a behavior is impracticable for the terminal/computer and printer/computer compatibility due to the very high number of compatible devices pairs. An ad-hoc solution has been adopted for these cases. As the compatibility between a terminal or a printer and a computer simply requires that they have a common interface, the terminal/computer and printer/computer compatibility has been represented by means of a class with no instances and an attached to-test procedure that checks whether the computer has available any external interface that matches one of the free external interfaces of the terminal or printer. The procedures are defined in such a way that they always change the knowledge base in a consistent way, that is if the initial state is consistent, the state after the execution of the procedure never violates the constraints. To this end, each procedure has associated a set of preconditions that, if satisfied, guarantee the correctness of the change. For example, the to-put procedure associated to the class `PTP_Complex_Subsystem` that requires as inputs the list of Units that are to be included in the Subsystem, and a list of Unit pairs, each pair representing a point-to-point connection that must be established between two Units of the Subsystem, performs the following steps:

(1) it checks whether the input data are consistent; this involves a number of controls, ranging from the check on the type of the data to that on the congruency between the list of Units and the list of Unit pairs;
(2) it checks whether the specified Subsystem topology is correct, i.e. if the graph is serial;
(3) for each point-to-point connection to be established, it checks whether the Units to be connected are compatible and whether they have an available free external interface Unit;
(4) if all the above conditions are satisfied, it establishes the point-to-point connections by creating appropriate instances of the class `PTPConnection` used to implement the knowledge on the topology;
(5) it creates the Subsystem.

Sometimes the creation of an object can be performed only after the creation/removal of other objects. In order to relieve the Architecture designer from the burden of it, the procedures make such operations automatically when possible. The to-put procedure of the `Expansion_Board_Unit`, for example, that is called in expanding a computer, works as follows:

(1) the computer Unit to be expanded is checked to ascertain whether it has a free expansion slot Unit of the required kind;
(2) if yes, the appropriate expansion compatibility class is searched, in order to find out whether there exists a board satisfying the user request that can be installed on the given Unit;
(3) if such a board is found, the expansion is made by accomplishing the following steps:
   - Step 1: a Unit representing the board is created;
   - Step 2: an instance of the appropriate expansion class is created;

- Step 3: an instance of the relation **HardwareExpansions**, having the expanded Unit as domain element, and the expansion instance as range element, is created, thus associating the expansion to the expanded Unit;
- Step 4: the pair <expanded Unit, used slot> is removed from relation **FreeExpansionSlots** , in order to represent the fact that the slot Unit of the expanded Unit that has been used for the expansion is no longer free;
- Step 5: operations that are specific to the particular expansion being made are performed; for instance, in case of an external interface expansion, the external interfaces carried by the expansion board must be added to the set of free external interfaces of the expanded Unit.

4.4. A Graphical Interface for ASPES

In this section, is illustrated the main features of a graphical interface developed for ASPES. In particular, it will be described: (1) the basic environment made available to the user at the beginning of an ASPES session; (2) the invocation of the ASPES user functions; (3) the visualization of the result of these operations; (4) the controls that are enforced on the operations at the interface level; (5) the error handling; (6) the user buffer. Each of these topics will be discussed in a separate section.

The Interface has been implemented as a C program that runs under Unix [3] and that uses the graphical facilities provided by the Sunview software library. It has been developed on a SUN 3/52 Workstation.

4.4.1. The Main Window

The Main Window is the only window that appears on the screen at the beginning of an ASPES session, and it is shown in Figure 2. The decomposition of an Architecture through five levels of abstraction is made transparent to the ASPES user by presenting him a Main Window with five icons, which are one-to-one with the Catalogue, Unit, Subsystem, Network, and Architecture levels. As Figure 2 shows, these icons are displayed in the top portion of the window, and the user cannot move them around as normal Sunview Sunwindows icons. A menu is associated with each icon, called the Icon Menu.

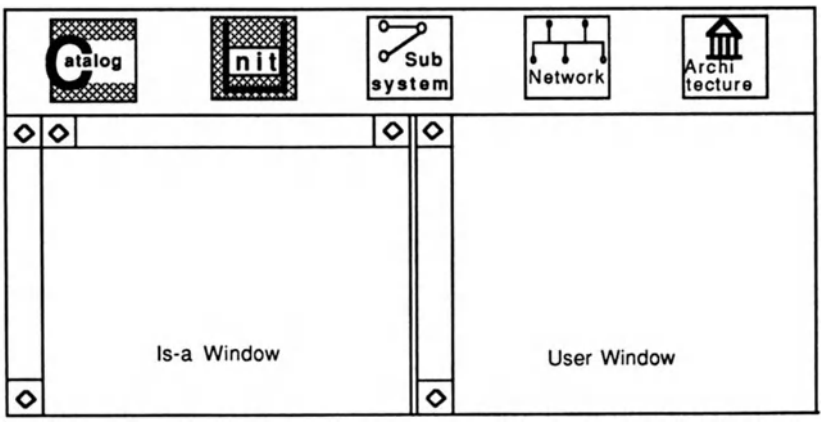


Fig. 2 - ASPES Interface Main Window

[3] UNIX is a trademark of AT&T

This menu presents the operations available at the ASPES level corresponding to the icon, and can be obtained by clicking the cursor on the icon.

The bottom portion of the Main Window is subdivided into two windows: one is the Is-a Window, where ASPES displays the Catalogue is-a hierarchy, upon a user request; the other is a standard Text Subwindow, reserved for the user needs, whence the name of User Window.

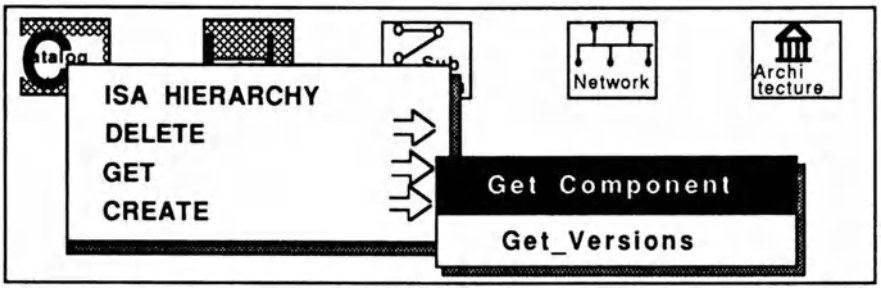
4.4.2. Invocation of ASPES User Functions

A selected list of the user operations is given in table 1 below. The operations available at each level can be viewed as hierarchically structured. This operation hierarchy is realized by having a menu hierarchy at each ASPES level. The root of the hierarchy is the Icon Menu associated to the level. In an Icon Menu, the name of each specialized operation is followed by an arrow: by following that arrow with the cursor, a menu containing the names of the lower level operations is displayed. This second level menu may contain in turn specialized operations, which are treated in the same way as in the Icon Menu. Figure 3.a shows a path of the Icon Menu of the Catalogue level.

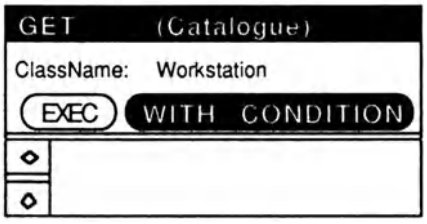
A window, called the Operation Window, is associated to each operation. The Operation Window appears after its operation has been invoked via an appropriate menu selection. An Operation Window is divided into two portions, the usage of which will be explained later, and can be resized and moved around the screen. In almost all cases, the selection of an operation starts a dialogue between the user and the system, having as final aim the specification of an ASPES operation. This dialogue takes place in the Operation Window, which therefore may assume different forms depending on which is the current state of this dialogue. To talk to the system the user selects one of the buttons that appear in the window. Upon this selection, the content of the window is interpreted by the interface as input data for the execution of the action corresponding to the selected button.

Table 1. - Selected List of ASPES User Functions

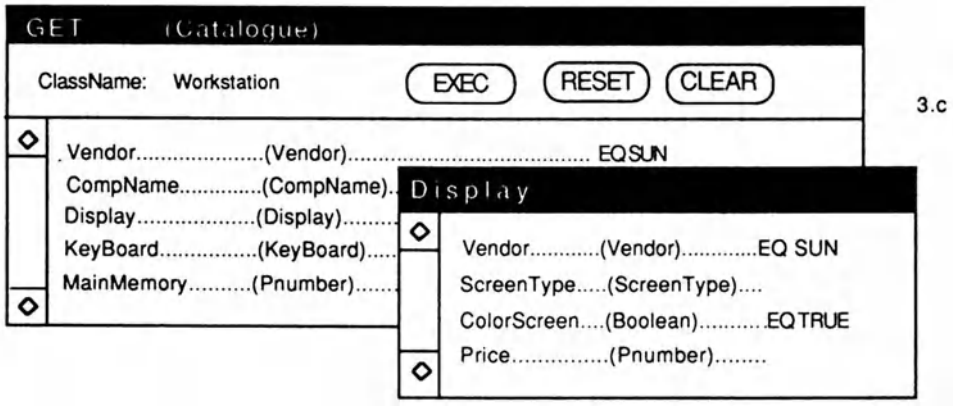
Catalogue Level	**Network Level**
Is-a	Get_Network
Get_Component	Display_Network
Display_Component	Create_Network
Instance_Of	Add_Subsystem_To_Network
Create_Component	
Create_Version	
Create_Main_Memory_Compatibility	
Create_Computer_Display_Compatibility	
Unit Level	**Architecture Level**
Get_Unit	Get_Architecture
Display_Unit	Display_Architecture
Create_Unit	Create_Architecture
Expand_With_Main_Memory_Board	Add_Network_To_Architecture
Subsystem Level	
Get_Subsystem	
Display_Subsystem	
Create_Subsystem	
Add_Unit_To_Subsystem	
Add_PTP_Connection_To_Subsystem	



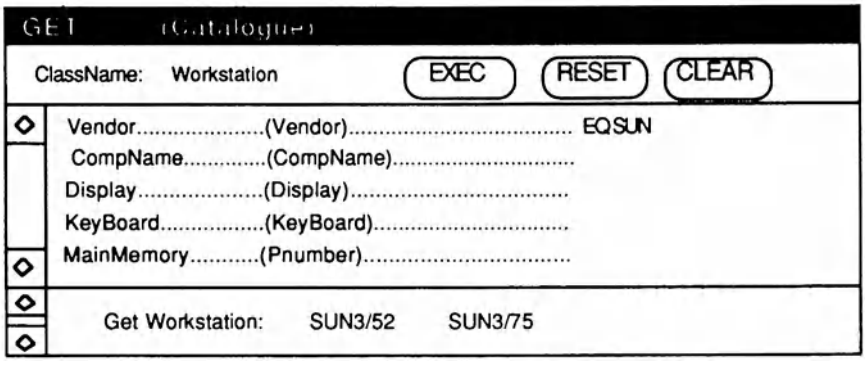
3.a



3.b



3.c



3.d

Fig. 3. - Steps of the specification of a *Get_Component* operation

The execution of an action may result in a call to ASPES, in which case the dialogue has reached its last step and an ASPES operation has been fired; or it may result in a change of the form of the Operation Window, which means that further information is required.

As an example, let us consider the invocation of a *Get_Component* operation at the Catalogue level. Figure 3 shows the various steps of this invocation. In Figure 3.a the selection from the second level menu is showed; as a result, the system display to the user the Operation Window associated to *Get_Component*, in its initial form. At this stage of the dialogue, the user is asked for the name of the class to be queried; having typed this name, the user can select one of the two buttons. Selecting **WITH CONDITION** (Figure 3.b) he will get from the system the list of the properties of the specified class. He can successively fill the space left after each property with a predicate to be satisfied by the result of the query. Complex conditions, involving objects which are property values of the object being queried, may be specified by invoking nested windows, as shown in Figure 3.c. Having done this, the user can select the **EXEC** button, to finally fire the ASPES operation that corresponds to the query. The result of the query is a list of qualifying objects, which will be returned by the system in the bottom portion of the Operation Window (Figure 3.d).

The Operation Window does not automatically disappear after the execution of an operation, because, as explained below, the system usually returns the result of the operation in that window. The user may remove the Operation Window at any time by using the functions provided by Sunwindow. If the use wishes to execute again an operation, he may go back to the desired state of the dialogue by using the **CLEAR** and **RESET** buttons provided (where appropriate) by the interface to this end. The **CLEAR** button restores the initial form of the Operation Window. In the *Get_Component* operation example, by clicking the **CLEAR** button after the display of the result, the user will get the Operation Window in its initial form. **RESET**, instead, causes the last values specified by the user to be reset, so that new values can be given. For instance, after the creation of an object has been executed, the Operation Window still contains the pairs (property, value) specified for the object just created. By **RESET**-ing at this point, the values will be removed, so that the user can insert new values to create a new object.

4.4.3. Visualization of Results

ASPES operations may in general have three kinds of results (excluding errors, which will be treated later): (1) a 'yes' result, which typically comes from a creation or remove operation. The system displays the result by opening, in the middle of the screen, a small window which contains the message that the operation has been successfully completed; (2) a set of objects; this is the result of a query on an ASPES class. The list of the returned objects is listed in the Operation Window from which the query has been issued (see Figure 3.d); (3) an object display. If the object is a token, its list of (property, value) pairs will be shown in a tabular form. If it is a class, the class definition will be given in tabular form, where each row of the table contains a pair (property, range). In both cases the result will be returned to the user in the bottom portion of the Operation Window associated to the specified operation. For Subsystem, Network, and Architecture objects, the result of the display may be given in both textual and graphical form.

To this end, the Operation Window of these operations consists of two different portions to display the two forms of the result. Figure 4 presents both the textual and the graphical display of a Subsystem consisting of four interconnected Units. The **EXEC** button in Figure 4, has to be selected only after having chosen the display form, which is controlled by the two circular buttons shown next. In the figure, the **TEMPLATE** button has been used in order to get the textual display given in the middle window; then, the oper-

ation has been re-executed with the **GRAPHIC** button selected, so obtaining the graphical display shown in the bottom portion of the window.

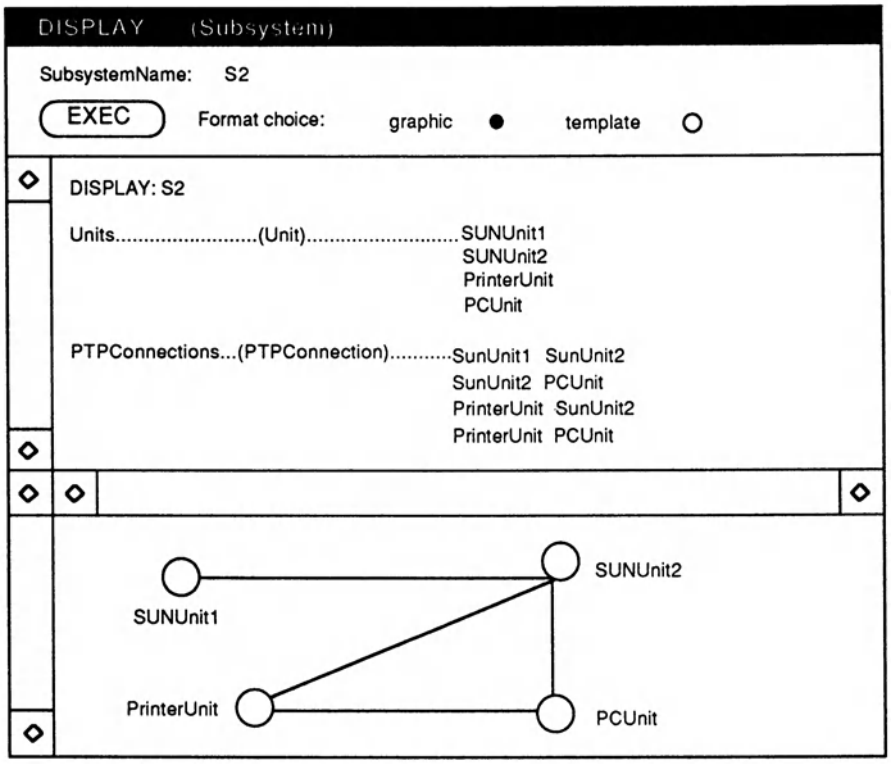


Fig. 4. - Textual and graphical display of a Subsystem

A special treatment is reserved for the visualization of the is-a hierarchy. As already pointed out, a portion of the Main Window, called Is-a Window, is always available for the displaying of a graphical description of the hierarchy. As the dimension of the graph does not fit in the Is-a Window, a scrolling facility is provided to move the visible portion of the hierarchy along the four dimensions.

4.4.4. Controls Enforced by the ASPES Interface

The kind of interaction between the user and the system allowed by the ASPES Interface presents a double advantage. The first advantage is that the system guides the specification of an operation, so that the user does not have to know in advance the parameters to be specified, their order, and their type. But there is another major advantage deriving from using this technique, and that is the automatic enforcement of certain constraints on the user operations. If the user interacted directly with the ASPES kernel, for example, there would be nothing that guarantees that the operations be specified in the correct form, that is with the right type and number of parameters. In this case, it would be the responsibility of the system to check the operation form, performing a usually high number of checks, which are tedious to code, and time consuming at run time. The presence of a graphical interface makes it possible to enforce many constraints when specifying the operation, as the interface may prevent the user from making certain mistakes.

As an example, let us consider again the *Get_Component* operation. When the user is given the list of properties (as in Figure 3.c), the only thing that he can do is to write something in the space left at the right-end of each row. Thus, ASPES is guaranteed that the query predicate only involves the properties of the queried class, and does not have to check for this. The same applies in the creation of an object, with the system providing the user with the list of properties to be filled in, and the user cannot change the portion of the window where the property names are displayed.

It would be too long to enumerate, operation by operation, the controls that are automatically enforced by the interface. We stress here only the effectiveness of a graphical interface in alleviating the language interpreter task concerning the operation consistency checking.

4.4.5. Error Handling

There can be two kinds of errors that can occur in the ASPES system. The first kind consists of the errors on the invocation of a user function, as for example when a parameter is missing or when an operation is invoked before the termination of the previous one. These errors result in a message which is displayed to the user in an 'ad hoc' window that appears in the middle of the screen. The rest of the screen is left unchanged so that the user can issue the correct operation after editing the incorrect specification. The window containing the error message may be removed by clicking any mouse button.

The second kind of errors regards not legal requests such for example the PTP connection of two units having no free external compatible interfaces. In this case, the message may be shown to the user either in an 'ad hoc' window, or in the bottom portion of the window associated to the operation that caused the error.

4.4.6. The User Window

The User Window occupies a part of the Main Window, and its *raison d'etre* is to provide the user of the ASPES system with a private workspace. The User Window is a standard Text Subwindow, i.e. a window for text editing that can be saved in a file, resized, and moved around the screen. Text can be selected by any other window and inserted in the User Window via the Sunwindow Selection Service, which relies on the mouse and a small set of functional keys of the SUN keyboard. Once the text has been put in the User Window, it can be processed via the normal editing facilities.

5. Conclusions

We have presented ASPES, a software system for the specification of office information system Architectures. ASPES is based on a formal theory (AST), whose models comprise such Architectures. The outcome of our work is twofold. On the practical side, we have showed how naturally the models of a theory can be translated into procedural semantic networks, with a straightforward correspondence between language and axioms from one hand, and classes and procedures from the other. On the theoretical side, we have encoded our perception of an architecture into a first order theory, thus providing a firm framework for the analysis of the properties of such perception. We have not performed this analysis, because it was beyond the scope of the Project, but we have proven the consistency of the theory, by effectively constructing ASPES Architectures, (TODOS TR 4.3.4). Our proof is of course informal, as it relies on the belief that PSN programs really enforce the constraints that are the model-theoretic counterparts of AST axioms.

6. Appendix A: Architecture Specification Language

We recall that a formal theory consists of the following parts (Mendelson 1964):

(1) a language, that is a set of well formed formulae (wffs) selected from the expressions over a given set of symbols;
(2) a set of wffs, called the axioms of the theory;
(3) a finite set of relations among wffs, called the rules of inference of the theory.

A first order theory is a formal theory such that: the language is a first order language; the axioms are divided into logical axioms, which are given once the language is given, and proper axioms, which vary from theory to theory; there are two rules of inference, generally called Modus Ponens and Generalization, whose schemata are, respectively:

(MP) from α and $(\alpha \supset \beta)$ infer β, and
(Gen) from α infer $(\forall x)\alpha$.

A first order language is built upon the following symbols:

- the parentheses ')' and '(';
- the propositional connectives $\supset$, conditional, and $\neg$, negation;
- the universal quantifier $\forall$;
- a (possibly empty) finite or denumerable set of constant symbols;
- a denumerable set of variable symbols;
- a (possibly empty) finite or denumerable set of function symbols;
- a finite or denumerable non-empty set of predicate symbols.

Among the expressions, i.e. finite sequences, that can be made out of the above symbols, the wffs of a first order language are defined as follows:

(a) constant and variable symbols are terms; if $t_1, t_2, ..., t_n$ are terms and f is an n-ary function symbol, then $f(t_1, t_2, ..., t_n)$ is a term;
(b) if P is a predicate symbol and $t_1, t_2, ..., t_n$ are terms, then $P(t_1, t_2, ..., t_n)$ is an atomic formula;
(c) an atomic formula is a well formed formula (wff); if x is a variable symbol and α and β are wffs, $(\neg\alpha)$, $(\alpha \supset \beta)$, and $((\forall x)\alpha)$ are wffs.

The existential quantifier ($\exists$) and the other propositional connectives usually found in first order languages ($\wedge$: conjunction, $\vee$: disjunction, $\otimes$: exclusive or, $\equiv$: biconditional), can be included in the language as syntactical abbreviations of their equivalent wffs. Furthermore, we will simplify the notation by using $(\forall x_1 x_2 \ldots x_n)\alpha$ as an abbreviation of $(\forall x_1(\forall x_2 \ldots (\forall x_n)\alpha \ldots))$, and by omitting parentheses according to the following rules:

- we omit the outer parentheses of a wff;
- when a wff contains only one binary connective, parentheses are omitted by association to the left;
- the connectives and quantifiers are ordered as follows: $\otimes$, $\equiv$, $\supset$, $\forall$, $\exists$, $\wedge$, $\vee$, $\neg$, and parentheses are eliminated according to the rule that, first, $\neg$ applies to the smallest wff following it, then $\vee$ is to connect the smallest wffs surrounding it, and so on. In applying this rule to occurrences of the same connective, we proceed from left to right;
- we omit parentheses around quantified wffs when they are preceded by other quantifiers.

A first order language can be interpreted via Tarskian semantics, by which a truth value can be assigned to any wff of the language. This is done by giving a non-empty set D, called the domain of the interpretation, and an assignment to each predicate symbol of a relation in D, to each function symbol of an operation in D, and to each constant symbol of some fixed element of D. Given such an interpretation, variables are thought of as ranging over the set D, and $\neg$, $\supset$ and $\forall$ are given their usual meaning. An interpretation

is said to be a model for a set of wffs if and only if every wff in the set is true for the interpretation.

6.1. Architecture Specification Language

The Architecture Specification Language is a first order language built upon the following symbols:

- variable symbols: $x, y, z, ..., x_1, x_2, ..., y_1, y_2, ...$;
- function symbols: the unary function symbol Σ, which has as argument a set of numbers, and returns the sum of these numbers (0 for the empty set); the binary function symbols $+$ (to be interpreted as the sum operation of number theory), and $\cup$ (to be interpreted as the set-theoretic union operation), both used in infix notation;
- predicate symbols: of the usual arithmetical symbols, ASL includes $=$ and $\geq$, which will be also used in infix notation; the set-theoretic equality predicate symbol *Same_Set*, and the following (grouped by level and associated with their informal semantics):

(a) Catalogue level predicate symbols

- $Computer(x)$: x is a computer;
- $Peripheral(x)$: x is a peripheral;
- $Local\_Area\_Network(x)$: x is the communication hardware device on which a Local Area Network relies;
- $Slot(x)$: x is an expansion slot;
- $Board(x)$: x is an expansion board;
- $External\_Interface(x)$: x is an external interface;
- $Network\_Interface(x)$: x is a network interface;
- $Number(x)$: x is a natural number;
- $Operating\_System(x)$: x is an operating system;
- $Software\_Package(x)$: x is a software package;
- $Slot\_Set(x)$: x is a set of expansion slots;
- $External\_Interface\_Set(x)$: x is a set of external interfaces;
- $Network\_Interface\_Set(x)$: x is a set of network interfaces;
- $Peripheral\_Set(x)$: x is a set of peripherals;
- $Software\_Package\_Set(x)$: x is a set of software packages;
- $Has\_Slot(x, y)$: the computer x has the slot set y;
- $Has\_External\_Interface(x, y)$: the computer or peripheral x has the external interface set y;
- $Has\_Network\_Interface(x, y)$: the computer x has the network interface set y;
- $Has\_Memory(x, y)$: the computer x has the amount of memory y;
- $Has\_Peripheral(x, y)$: the computer x has the peripheral set y;
- $Has\_Operating\_System(x, y)$: the computer x has the operating system y;
- $Has\_Software(x, y)$: the computer x has the set of software packages y;
- $Bears\_Memory(x, y)$: the expansion board x bears the amount of memory y;
- $Bears\_External\_Interface(x, y)$: the expansion board x bears the set of external interfaces y;
- $Bears\_Network\_Interface(x, y)$: the expansion board x bears the set of network interfaces y;
- $Hw\_Expansion\_Compatible(x, y, z)$: the computer x can be hardware expanded by inserting the board z into the slot y;
- $Sw\_Expansion\_Compatible(x_1, x_2, x_3, x_4, x_5)$: the computer x_1 can be software expanded by installing the software package x_2, provided that x_1 runs the operating

system x_3, has at least x_4 main memory, and has each one of the software packages in the set x_5;
- $PTP\_Compatible(x_1, x_2, x_3, x_4)$: the computer or peripheral x_1 can be point-to-point connected to the computer or peripheral x_3, through the external interfaces x_2 of x_1 and x_4 of x_3;
- $Network\_Compatible(x_1, x_2, x_3, x_4)$: the computer x_1 can be connected to the Local Area Network hardware x_2 through the network interface x_3, provided that x_1 has the software package x_4.

(b) Unit level predicate symbols

- $Instance(x, y)$: the Unit y is an instance of the Catalogue object x;
- $Computer\_Unit(x)$: x is a computer Unit;
- $Peripheral\_Unit(x)$: x is a peripheral Unit;
- $Slot\_Unit(x)$: x is an expansion slot Unit;
- $Board\_Unit(x)$: x is an expansion board Unit;
- $External\_Interface\_Unit(x)$: x is an external interface Unit;
- $Network\_Interface\_Unit(x)$: x is a network interface Unit;
- $Operating\_System\_Unit(x)$: x is an operating system Unit;
- $Software\_Package\_Unit(x)$: x is a software package Unit;
- $Slot\_Unit\_Set(x)$: x is a set of expansion slot Units;
- $External\_Interface\_Unit\_Set(x)$: x is a set of external interface Units;
- $Network\_Interface\_Unit\_Set(x)$: x is a set of network interface Units;
- $Peripheral\_Unit\_Set(x)$: x is a set of peripheral Units;
- $Software\_Package\_Unit\_Set(x)$: x is a set of software package Units;
- $Has\_Slot\_Unit(x, y)$: the computer Unit x has the set of expansion slot Units y;
- $Has\_Total\_Memory(x, y)$: the computer Unit x has the amount of memory y (y includes also the memory acquired by x through hardware expansions);
- $Has\_External\_Interface\_Unit(x, y)$: the computer Unit x has the set of external interface Units y (y includes also the external interfaces acquired by x through hardware expansions);
- $Has\_Network\_Interface\_Unit(x, y)$: the computer Unit x has the set of network interface Units y (y includes also the network interfaces acquired by x through hardware expansions);
- $Has\_Peripheral\_Unit(x, y)$: the computer Unit x has the set of peripheral Units y;
- $Has\_Operating\_System\_Unit(x, y)$: the computer Unit x has the operating system Unit y;
- $Has\_Software\_Unit(x, y)$: the computer Unit x has the set of software package Units y (y includes also the software Units acquired by x through software expansions);
- $Add\_Memory(x, y)$: the expansion board Unit x bears the amount of memory y;
- $Add\_External\_Interface(x, y)$: the expansion board Unit x bears the set of external interface Units y;
- $Add\_Network\_Interface(x, y)$: the expansion board Unit x bears the set of network interface Units y;
- $Hw\_Expanded(x, y, z)$: the computer Unit x has been hardware expanded by the insertion of the board Unit z in the slot Unit y;
- $Sw\_Expanded(x, y)$: the computer Unit x has been software expanded by the installation of the software package Unit y;
- $Standard\_External\_Interface(x, y)$: the computer or peripheral Unit x gets from the Catalogue the set of external interface Units y;

- *Added_External_Interface*(x,y) : the computer or peripheral Unit x gets from hardware expansions the set of external interface Units y;
- *Standard_Network_Interface*(x,y) : the computer Unit x gets from the Catalogue the set of network interface Units y;
- *Added_Network_Interface*(x,y) : the computer Unit x gets from hardware expansions the set of network interface Units y;
- *Added_Memory*(x,y) : the computer Unit x gets from hardware expansions the amount of memory y;
- *Standard_Software*(x,y) : the computer Unit x gets from the Catalogue the set of software package Units y;
- *Added_Software*(x,y) : the computer Unit x gets from software expansions the set of software package Units y.

(c) Subsystem level predicate symbols

- *Subsystem*(x) : x is a Subsystem;
- *Complex_Subsystem*(x) : x is a Complex Subsystem, i.e. a Subsystem with at least one point-to-point connection;
- *Unit_Set*(x) : x is a set of Units;
- *Has_Unit*(x,y) : the Complex Subsystem x consists of the set of Units y;
- *Serially_Connected*(x) : the set of Units x is serially connected;
- *Reachable*(x,y) : the Unit x is reachable from the Unit y by following a path of point-to-point connections;
- *PTP_Connected*(x,y,u,v) : the Unit x is point-to-point connected to the Unit u, and the connection takes the external interface Units y and v, of x and u, respectively.

(d) Network level predicate symbols

- *Office_Network*(x) : x is an Office Network;
- *LAN_Unit*(x) : x is a Local Area Network hardware Unit;
- *Subsystem_Set*(x) : x is a set of Subsystems;
- *Has_LAN*(x,y) : the Office Network x has LAN hardware Unit y;
- *Has_Subsystem*(x,y) : the Office Network x consists of the set of Subsystems y;
- *Hosts*(x,y) : the set of Subsystems x are hosts of the LAN hardware Unit y;
- *LAN_Connected*(z,u,v,y) : the computer Unit u of Subsystem z is connected to the LAN Unit y through the network interface v.

(e) Architecture level predicate symbols

- *Architecture*(x) : x is an Architecture;
- *Complex_Architecture*(x) : x is a Complex Architecture, i.e. an Architecture with at least one Office Network;
- *Network_Set*(x) : x is a set of Office Networks;
- *Has_Network*(x,y) : the Complex Architecture x consists of the set of Office Networks y;
- *Communicate*(x) : the Office Networks in the set x communicate via common Subsystems;
- *Connected*(x,y) : the Office Network x is connected to the Office Network y.

7. Appendix B: Completion to the AST Axiom System

This Appendix contains the axioms that complete the AST axiom system introduced in Section 3.1. The axioms have been partitioned according to the kind of constraint they represent.

7.1. Type Axioms

These axioms give the proper structure (or type) of predicate symbols.

$(\forall xy)(Has\_Slot(x,y) \supset Computer(x) \wedge Slot\_Set(y))$
$(\forall xy)(Has\_External\_Interface(x,y) \supset (Computer(x) \vee Peripheral(x)) \wedge$
$External\_Interface\_Set(y))$
$(\forall xy)(Has\_Network\_Interface(x,y) \supset Computer(x) \wedge$
$Network\_Interface\_Set(y))$
$(\forall xy)(Has\_Memory(x,y) \supset Computer(x) \wedge Number(y))$
$(\forall xy)(Has\_Peripheral(x,y) \supset Computer(x) \wedge$
$Peripheral\_Set(y))$
$(\forall xy)(Has\_Operating\_System(x,y) \supset Computer(x) \wedge Operating\_System(y))$
$(\forall xy)(Has\_Software(x,y) \supset Computer(x) \wedge Software\_Package\_Set(y))$
$(\forall xy)(Bears\_External\_Interface(x,y) \supset Board(x) \wedge External\_Interface\_Set(y))$
$(\forall xy)(Bears\_Network\_Interface(x,y) \supset Board(x) \wedge Network\_Interface\_Set(y))$
$(\forall xy)(Bears\_Memory(x,y) \supset Board(x) \wedge Number(y))$
$(\forall xyz)(Hw\_Expansion\_Compatible(x,y,z) \supset Computer(x) \wedge Slot(y) \wedge Board(z))$
$(\forall x_1x_2x_3x_4x_5)(Sw\_Expansion\_Compatible(x_1,x_2,x_3,x_4,x_5) \supset Computer(x_1) \wedge$
$Software\_Package(x_2) \wedge Operating\_System(x_3) \wedge Number(x_4) \wedge$
$Software\_Package\_Set(x_5))$
$(\forall x_1x_2x_3x_4)(PTP\_Compatible(x_1,x_2,x_3,x_4) \supset (Computer(x_1) \vee Peripheral(x_1)) \wedge$
$External\_Interface(x_2) \wedge (Computer(x_3) \vee Peripheral(x_3)) \wedge$
$External\_Interface(x_4))$
$(\forall x_1x_2x_3x_4)(Network\_Compatible(x_1,x_2,x_3,x_4) \supset Computer(x_1) \wedge$
$Local\_Area\_Network(x_2) \wedge Network\_Interface(x_3) \wedge$
$Software\_Package(x_4))$
$(\forall xy)(Has\_Slot\_Unit(x,y) \supset Computer\_Unit(x) \wedge Slot\_Unit\_Set(y))$
$(\forall xy)(Has\_External\_Interface\_Unit(x,y) \supset Computer\_Unit(x) \wedge$
$External\_Interface\_Unit\_Set(y))$
$(\forall xy)(Has\_Network\_Interface\_Unit(x,y) \supset Computer\_Unit(x) \wedge$
$Network\_Interface\_Unit\_Set(y))$
$(\forall xy)(Has\_Total\_Memory(x,y) \supset Computer\_Unit(x) \wedge Number(y))$
$(\forall xy)(Has\_Peripheral\_Unit(x,y) \supset Computer\_Unit(x) \wedge Peripheral\_Unit\_Set(y))$
$(\forall xy)(Has\_Operating\_System\_Unit(x,y) \supset Computer\_Unit(x) \wedge$
$Operating\_System\_Unit(y))$
$(\forall xy)(Has\_Software\_Unit(x,y) \supset Computer\_Unit(x) \wedge$
$Software\_Package\_Set(y))$
$(\forall xy)(Add\_Memory(x,y) \supset Board\_Unit(x) \wedge Number(y))$
$(\forall xy)(Add\_External\_Interface(x,y) \supset Board\_Unit(x) \wedge$
$External\_Interface\_Unit\_Set(y))$
$(\forall xy)(Add\_Network\_Interface(x,y) \supset Board\_Unit(x) \wedge$
$Network\_Interface\_Unit\_Set(y))$
$(\forall xyz)(Hw\_Expanded(x,y,z) \supset Computer\_Unit(x) \wedge Slot\_Unit(y) \wedge$
$Board\_Unit(z))$
$(\forall xyz)(Sw\_Expanded(x,y) \supset Computer\_Unit(x) \wedge Software\_Package\_Unit(y))$
$(\forall xyuv)(PTP\_Connected(x,y,u,v) \supset (Computer\_Unit(x) \vee Peripheral\_Unit(x)) \wedge$
$External\_Interface\_Unit(y) \wedge (Computer\_Unit(u) \vee Peripheral\_Unit(u)) \wedge$
$External\_Interface\_Unit(v))$
$(\forall xy)(Has\_LAN(x,y) \supset Office\_Network(x) \wedge LAN\_Unit(y))$
$(\forall xy)(Has\_Subsystem(x,y) \supset Office\_Network(x) \wedge Subsystem\_Set(y))$
$(\forall xy)(Hosts(x,y) \supset Subsystem\_Set(x) \wedge LAN\_Unit(y))$

$(\forall xyzv)(LAN\_Connected(x,y,z,v) \supset Subsystem(x) \wedge Computer\_Unit(y) \wedge$
$Network\_Interface\_Unit(z) \wedge LAN\_Unit(v))$
$(\forall xy)(Has\_Network(x,y) \supset Complex\_Architecture(x) \wedge Network\_Set(y))$

7.2. Set Type Axioms

Set type axioms bind the members of the argument of a set predicate symbols to their proper type.

$(\forall x)(Slot\_Set(x) \supset (\forall y)(Member(x,y) \supset Slot(y)))$
$(\forall x)(External\_Interface\_Set(x) \supset (\forall y)(Member(x,y) \supset External\_Interface(y)))$
$(\forall x)(Network\_Interface\_Set(x) \supset (\forall y)(Member(x,y) \supset Network\_Interface(y)))$
$(\forall x)(Peripheral\_Set(x) \supset (\forall y)(Member(x,y) \supset Peripheral(y)))$
$(\forall x)(Software\_Package\_Set(x) \supset (\forall y)(Member(x,y) \supset Software\_Package(y)))$
$(\forall y)(Slot\_Unit\_Set(y) \supset (\forall x_1)(Member(x_1,y) \supset$
$Slot\_Unit(x_1)))$
$(\forall y)(External\_Interface\_Unit\_Set(y) \supset (\forall x_1)(Member(x_1,y) \supset$
$External\_Interface\_Unit(x_1)))$
$(\forall y)(Network\_Interface\_Unit\_Set(y) \supset (\forall x_1)(Member(x_1,y) \supset$
$Network\_Interface\_Unit(x_1)))$
$(\forall y)(Peripheral\_Unit\_Set(y) \supset (\forall x_1)(Member(x_1,y) \supset$
$Peripheral\_Unit(x_1)))$
$(\forall y)(Software\_Package\_Unit\_Set(y) \supset (\forall x_1)(Member(x_1,y) \supset$
$Software\_Package\_Unit(x_1)))$
$(\forall x)(Unit\_Set(x) \supset (\forall x_1)(Member(x_1,x) \supset$
$(Computer\_Unit(x_1) \vee Peripheral\_Unit(x_1))))$
$(\forall x)(Subsystem\_Set(x) \supset (\forall x_1)(Member(x_1,x) \supset Subsystem(x_1)))$
$(\forall x)(Network\_Set(x) \supset (\forall x_1)(Member(x_1,x) \supset Office\_Network(x_1))))$

7.3. Property Uniqueness Axioms

The following axioms apply to those predicate symbols representing binary relationships for which the uniqueness of value of such relationships is required.

$(\forall xy)(Has\_External\_Interface(x,y) \supset (\forall z)(Has\_External\_Interface(x,z) \supset$
$Same\_Set(y,z)))$
$(\forall xy)(Has\_Network\_Interface(x,y) \supset (\forall z)(Has\_Network\_Interface(x,z) \supset$
$Same\_Set(y,z)))$
$(\forall xy)(Has\_Memory(x,y) \supset (\forall z)(Has\_Memory(x,z) \supset (y = z)))$
$(\forall xy)(Has\_Peripheral(x,y) \supset (\forall z)(Has\_Peripheral(x,z) \supset$
$Same\_Set(y,z)))$
$(\forall xy)(Has\_Operating\_System(x,y) \supset (\forall z)(Has\_Operating\_System(x,z) \supset$
$(y = z)))$
$(\forall xy)(Has\_Software(x,y) \supset (\forall z)(Has\_Software(x,z) \supset Same\_Set(y,z)))$
$(\forall xy)(Bears\_External\_Interface(x,y) \supset (\forall z)(Bears\_External\_Interface(x,z) \supset$
$Same\_Set(y,z)))$
$(\forall xy)(Bears\_Network\_Interface(x,y) \supset (\forall z)(Bears\_Network\_Interface(x,z) \supset$
$Same\_Set(y,z)))$
$(\forall xy)(Bears\_Memory(x,y) \supset (\forall z)(Bears\_Memory(x,z) \supset (y = z)))$

7.4. Set Disjointness Axioms

These axioms express a set disjointness condition on the external interfaces of a computer (or peripheral), and on the network interfaces and the peripherals associated to a computer.

$(\forall x_1 x_2 x_3 x_4)((Has\_External\_Interface(x_1, x_3) \wedge$
$Has\_External\_Interface(x_2, x_4)) \supset$
$(\exists y)((Member(y, x_3) \wedge Member(y, x_4)) \supset (x_1 = x_2)))$
$(\forall x_1 x_2 x_3 x_4)((Has\_Network\_Interface(x_1, x_3) \wedge$
$Has\_Network\_Interface(x_2, x_4)) \supset$
$(\exists y)((Member(y, x_3) \wedge Member(y, x_4)) \supset (x_1 = x_2)))$
$(\forall x_1 x_2 x_3 x_4)((Has\_Peripheral(x_1, x_3) \wedge$
$Has\_Peripheral(x_2, x_4)) \supset$
$(\exists y)((Member(y, x_3) \wedge Member(y, x_4)) \supset (x_1 = x_2)))$

7.5. Instanciation Axioms

The following axioms are the completion to the instanciation axioms.

$(\forall x)(Peripheral\_Unit(x) \supset (\exists x_1)(Peripheral(x_1) \wedge Instance(x_1, x)))$
$(\forall x)(Slot\_Unit(x) \supset (\exists x_1)(Slot(x_1) \wedge Instance(x_1, x)))$
$(\forall x)(Board\_Unit(x) \supset (\exists x_1)(Board(x_1) \wedge Instance(x_1, x)))$
$(\forall x)(External\_Interface\_Unit(x) \supset (\exists x_1)(External\_Interface(x_1) \wedge$
$Instance(x_1, x)))$
$(\forall x)(Network\_Interface\_Unit(x) \supset (\exists x_1)(Network\_Interface(x_1) \wedge$
$Instance(x_1, x)))$
$(\forall x)(Operating\_System\_Unit(x) \supset (\exists x_1)(Operating\_System(x_1) \wedge Instance(x_1, x)))$
$(\forall x)(Software\_Package\_Unit(x) \supset (\exists x_1)(Software\_Package(x_1) \wedge$
$Instance(x_1, x)))$
$(\forall x)(LAN\_Unit(x) \supset (\exists x_1)(Local\_Area\_Network(x_1) \wedge Instance(x_1, x)))$

7.6. Completion to the Unit Axioms

Finally, the list of Unit axioms that complete the description of the Unit level.

$(\forall xy)(Has\_Peripheral\_Unit(x, y) \equiv (\exists x_1 x_2)(Instance(x_1, x) \wedge$
$Has\_Peripheral(x_1, x_2) \wedge$
$(\forall x_3 x_4)(Instance(x_3, x_4) \supset (Member(x_3, x_2) \equiv Member(x_4, y)))))$
$(\forall xy)(Has\_Operating\_System\_Unit(x, y) \equiv (\exists x_1 x_2)(Instance(x_1, x) \wedge$
$Instance(x_2, y) \wedge Has\_Operating\_System(x_1, x_2)))$
$(\forall xy)(Has\_Network\_Interface\_Unit(x, y) \equiv$
$(\exists y_1 y_2)(Standard\_Network\_Interface(x, y_1) \wedge$
$Added\_Network\_Interface(x, y_2) \wedge (y = (y_1 \cup y_2))))$
$(\forall xy)(Standard\_Network\_Interface(x, y) \equiv (\exists x_1 x_2)(Instance(x_1, x) \wedge$
$Has\_Network\_Interface(x_1, x_2) \wedge$
$(\forall x_3 x_4)(Instance(x_3, x_4) \supset (Member(x_3, x_2) \equiv Member(x_4, y)))))$
$(\forall xy)(Added\_Network\_Interface(x, y) \equiv (\forall x_1)(Member(x_1, y) \supset$
$(\exists x_2 x_3 x_4)(Hw\_Expanded(x, x_2, x_3) \wedge Member(x_1, x_4) \wedge$
$Add\_Network\_Interface(x_3, x_4))))$
$(\forall xy)(Add\_Network\_Interface(x, y) \supset (\exists zu)(Board(z) \wedge Instance(z, x) \wedge$
$Bears\_Network\_Interface(z, u) \wedge (\forall x_1 x_2)(Instance(x_1, x_2) \supset$
$(Member(x_1, u) \equiv Member(x_2, y))))$
$(\forall xy)(Has\_Total\_Memory(x, y) \equiv (\exists x_1 x_2 x_3)(Instance(x_1, x) \wedge$
$Has\_Memory(x_1, x_2) \wedge (Added\_Memory(x, x_3) \wedge (y = (x_2 + x_3)))))$
$(\forall xy)(Added\_Memory(x, y) \equiv (\exists z)((y = \Sigma(z)) \wedge$
$(\forall x_1 x_2 x_3)(Hw\_Expanded(x, x_1, x_2) \wedge (Add\_Memory(x_2, x_3)$
$\equiv Member(x_3, z)))))$
$(\forall xy)(Add\_Memory(x, y) \supset (\exists z)(Board(z) \wedge Instance(z, x) \wedge$
$Bears\_Memory(z, y)))$

$$(\forall xy)(Has\_Software\_Unit(x,y) \equiv (\exists y_1 y_2)(Standard\_Software(x,y_1) \wedge$$
$$Added\_Software(x,y_2) \wedge (y = (y_1 \cup y_2))))$$
$$(\forall xy)(Standard\_Software(x,y) \equiv (\exists x_1 x_2)(Instance(x_1,x) \wedge$$
$$Has\_Software(x_1,x_2) \wedge (\forall x_3 x_4)(Instance(x_3,x_4) \supset$$
$$(Member(x_3,x_2) \equiv Member(x_4,y)))))$$
$$(\forall xy)(Added\_Software(x,y) \equiv (\forall x_1)(Member(x_1,y) \supset Sw\_Expanded(x,x_1)))$$

Chapter 5 - Part III

Performance Modeling Phase

Etienne L.M.E. van Dorsselaer and Frank J.M. Heijmink

1 Introduction

Numerous demands can be made concerning Office Information Systems (OISs). These include such factors as cost, reliability, availability, functionality, correctness, serviceability, security, and performance. The TODOS methodology supports a multi-criterion evaluation of OIS architectures, proposed during the architecture specification phase (Ch. 5, part II), with respect to the office requirements defined in the requirements collection and analysis phase (WP1). The increasing complexity of computer systems, and this certainly holds for OISs as well, results in a growing demand for tools and techniques that assist in the understanding of these systems during their whole life-cycle, from design until exploitation. *Performance Evaluation (PE)* of computer systems pursues the objective of quantitatively assessing the behaviour of these systems with respect to the computational tasks to be performed. Despite the advances in the field of computer performance evaluation made during the last few decades as a result of considerable research efforts, it has developed in substantial isolation with respect to such disciplines as computer systems design, implementation, and management (Ferrari 1986). One of the distinctive features of the TODOS Design Methodology is therefore the explicit integration of performance evaluation into the development process of Office Information Systems. The objective of the performance modelling phase within the TODOS methodology is to support the choice of the best alternative among a number of proposed OIS architectures.

In Section 2 a number of different approaches to solve PE problems are be shortly reviewed. The approach to PE chosen within TODOS is *performance modelling*, that is performance evaluation based on abstract system representations, here *queueing network models*, which will consequently get the most attention. Solution techniques for queueing network models are usually based on complex mathematical theories. In order to make them available to system designers, who normally are no experts in the field of queueing network theory, these techniques should be incorporated in performance evaluation tools. These tools should support their users in the tasks of specification of performance models and choice of appropriate solution methods. A small number of such tools are readily available. Within TODOS, we have chosen the *Queueing Network Analysis Package (QNAP2)* (Potier 1984), (Veran et al. 1985), developed by INRIA and CII-Honeywell Bull and currently marketed by Simulog. QNAP2 contains a collection of resolution algorithms and a common user interface for description, analysis control and result presentation.

A PE problem has three basic constituents:

1) the computing resources to be considered (*machine*);
2) the computational tasks to be performed (*load*);
3) the assessment metric to be employed (*performance measures*).

A solution to a PE problem consists of the mapping of a *machine* and a *load* onto a range of *performance measures* (Beilner 1985). Within TODOS the machine represents the architecture of the OIS, or more precisely those architectural components that are relevant to OIS performance, and the load represents the activities carried out by office workers, that is for as far as these are supported by software functions of the OIS.

The OIS architectures considered in TODOS are modularily constructed using the Architecture Specification Language (ASL) (TODOS TR 4.2) and are based on existing hardware (HW) and software (SW) components. In order to let even a system designer, who has very little knowledge of performance modelling, construct performance models for the OIS architectures, the PE tool employed should support construction of performance models in the same modular way from 'basic' performance components representing the HW and SW components employed during architecture specification. This required modularity has been achieved by developing a number of modelling techniques on top of the standard QNAP2-mechanisms (TODOS TR 4.2.5). In Section 3, a characterisation of the considered OIS architectures is presented, and the developed modelling techniques to support modular construction of performance models for these architectures are explained.

System performance may be specified in various ways. When quantitatively evaluating OIS architectures, we are interested in system-oriented as well as user-oriented *performance measures*. System-oriented performance measures are utilisation of resources, i.e. the fraction of time that a resource is busy, and population, i.e. the average length of queues for resources. Resources in this context are hardware components like processors, disks, printers, etc. User-oriented performance measures are response times of transactions, i.e. the average amount of time needed to completely serve a user request, and throughput of transactions, i.e. the rate at which transactions are being completed.

In the following, performance is used as a generic term for the set of performance measures for an OIS architecture in a given PE study, which corresponds to the way it is commonly used. The exact definition for the concept of performance may therefore only be given by referring directly to the performance measures chosen for a particular PE study, and may consequently differ from one study to another. Thus, for example, in one study performance may be used as a synonym for the interactive response time of a certain application, while in another study it may represent a combination of response times and throughputs.

It takes more than a PE tool supporting model construction and solution to obtain meaningful results from a performance modelling study as part of the TODOS methodology. Therefore, a methodological approach to the performance modelling process has been developed defining the interfaces with other activities in the TODOS methodology, the steps to be taken during a performance modelling study, and the role of both system designer(s) and PE tool during such a study (TODOS TR 4.2.6). This methodological approach is explained into some more detail in the Sections 4 and 5.

From it, one may learn that some of the most crucial steps of the performance modelling process, such as definition of objectives and workload characterisation, require (expert) knowledge about the OIS under development and its environment from the (team of) system designer(s) responsible for architecture design, and that consequently the performance modelling process will be manually driven. To illustrate the performance modelling process as part of the TODOS methodology and the operation of the TODOS performance modelling tool, in Section 6 an example OIS architecture is modelled and evaluated. The OIS architecture considered is based on the overall TODOS Test Case (Ch. 6).

2 Performance Evaluation Methods

PE methods, that is techniques to solve PE problems, may be sub-divided into two main categories, namely *measuring* and *modelling*. Measurements may be performed on existing systems or on physical prototypes.

2.1 Measurements on Existing Systems

System performance of existing systems may be measured under operational conditions, that is under a workload imposed by actual system users, or under an artificial workload (benchmark). Benchmarking is mainly used to compare different systems under the same conditions during system acquisition.

There exist two major types of measurement tools: hardware and software monitors. *Hardware monitors* are plugged into the system and measure electrical signals. They have the advantage of imposing only very small or no overhead to the system, but have only limited capabilities for the performance measures they can produce. *Software monitors* can produce much more detailed information about the system's operation, but they have, since they are programs which run on the system, the disadvantage of using system resources, which are difficult to precisely account for.

Clearly, measurements can only give some understanding of the behaviour of the current system. It is extremely difficult to use the data obtained by measurements to predict future behaviour of the system as workload and configuration may change, and measurements are rarely of much use during system design. Though measurements are not much in use for performance evaluation as such, they may prove very useful during performance modelling studies for parameterisation and validation of a baseline-model of an existing system.

2.2 Prototyping

An approach that overcomes the need of a running system for performing measurements is to build a physical prototype of the system under study. The idea is to incorporate as much (relevant) details of the system as possible in a prototype, and perform measurements on this working prototype, instantiated with an artificial workload. Prototyping may consequently be used during system design, but has the disadvantage of being rather labour-intensive and therefore is hardly applicable to evaluate different (design) alternatives for complex systems.

2.3 Modelling

A model of a system is an *abstract representation* that embodies the system's behaviour. A performance model may be built at many levels of detail, but even the highest level of detail must incorporate some of the complexities of the real system. Performance modelling may be employed during all phases of a system's life cycle. It may be easily used for evaluating performance under system change by adjusting model parameters according to an evolving workload or a change of system components. Another advantage of modelling is that it frequently helps in gaining a deeper understanding of how the system performs, since it forces a modeller to think conscientiously about which system details/components actually influence system performance and therefore should be incorporated in the abstract system representation.

The performance evaluation method, which has been chosen within TODOS consists of modelling of OISs in terms of *Queueing Network Models*, which are the models most widely used for performance evaluation. In the following a short introduction to queueing network models and the techniques to solve them is presented.

2.4 Queueing Network Models

Queueing network models are mathematical models that consist of one or more interconnected service centers, and are used to represent systems in which there is *contention* (and therefore queueing) for resources. In order to completely specify a service center, which consists of a queue and one or more servers (see figure below), it is necessary to describe the following parameters:

- *arrival process*, i.e. (a distribution of) the time between two successive customer arrivals at the service center;
- *service process*, i.e. (a distribution of) the time it takes to serve a customer at the service center;
- *queueing discipline*, i.e. the algorithm that determines in which sequence customers are served at the service center. Example of queueing (or scheduling) disciplines are first-come first-served (FCFS), priority queueing (with pre-emptive resume) (PRIOR (-PR)), processor sharing (PS), last-come first-served (with pre-emptive resume) (LCFS (-PR)), etc.

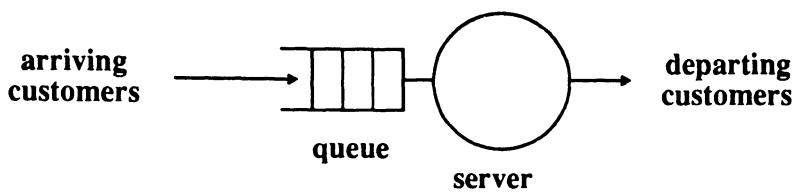


Fig. 1 A single server.

For a queueing network model consisting of multiple service centers, also the *sequence of service centers* visited by each customer has to be specified.

Two major approaches for solving queueing network models are *simulation* and *analysis*. Analytical methods seek equations relating desired performance measures to the available parameters. Simulative methods generate possible behaviour sequences of the system and measure the desired performance measures in those sequences.

Exact analysis of queueing network models may involve the solution of a complex set of state equations. Except for models that are inherently simple because of their limited size, an exact solution is feasible only for queueing network models that satisfy a number of assumptions, the so-called *product form networks*. Two approaches to the product form solution exist, namely the stochastic and the operational approach, which differ mainly in the way assumptions are formulated and hardly in the results obtained.

In the *stochastic approach* (Baskett et al. 1975), the assumptions are described in terms of the theory of stochastic processes such as service time and interarrival time distributions. The main disadvantage of this kind of assumptions is that in general it is very difficult to verify positively whether they are met or not.

The *operational approach* (Denning 1978) is based on the premise of *testability*. All parameters are defined from data taken over a finite period of time and the performance analyst can test whether the basic assumptions hold in any observation period. Of course, this advantage only exists when evaluating the performance of a computer system in operation. An important operational principle is the assumption of *flow balance*, which means that the number of arrivals at a given device must be (almost) the same as the number of departures from that device during the observation period. The job flow balance implies that the throughputs everywhere in the system are determined by the throughput at any one point. Since an increasing workload will drive some device into saturation, this assumption allows determining asymptotes on throughputs and response times.

For models that cannot be solved by exact analytical methods, approximate or simulative methods may be employed. *Simulative methods* may be used for solving virtually every model, but may be computationally very expensive. Another problem with simulation is concerned with the statistical nature of the simulation process. The analyst has a need for determining how close simulation estimates are to the correct values for the model, and the run time necessary to obtain estimates near the correct values. An approach to solve the first question is the estimation of confidence intervals, while the second may be handled by sequential stopping rules, i.e. repeated execution for a certain sampling period until a satisfying confidence level is reached. Statistical analysis of simulation results may be supported by performance evaluation tools.

Approximations may be attractive, because they are usually computationally inexpensive. A disadvantage of approximations is that it is exceedingly difficult to characterise the error made, since characterisation of the error implies a solution for a (presumably) unsolvable network model. In general, only some empirical evidence that the error is small can be provided.

2.5 The Queueing Network Analysis Package (QNAP2)

QNAP2 (Queueing Network Analysis Package 2) is a performance evaluation tool, developed by INRIA and CII-Honeywell Bull, to support the construction and solution of complex queueing network models of discrete flow systems or resource contention problems (e.g. communication networks or computer architectures). It contains a collection of resolution algorithms (analytical, both exact and approximate, and simulative as well as an exact Markovian solver) and a common user interface for description, analysis control and result presentation.

The modelling framework of QNAP2 is the representation of a system as a network of stations through which customers are circulating. The stations represent the physical or logical processing facilities of the system, the customers represent the processes being executed and competing for these facilities. The QNAP2 language is a high-level object-oriented interactive language, including two levels of specification : the command language and the algorithmic language. The *command language* consists of a sequence of commands, each corresponding to a specific function. It consists of commands for the creation of variables or structured objects (/DECLARE/), the description of the stations (/STATION/), analysis control (/CONTROL/), and the execution of a sequence of algorithmic statements (/EXEC/). The command language supports two important features of QNAP2: *interactivity* (because each command is interpreted sequentially the user may at any time input any command) and *structuring* (the command language enforces the modelling framework of QNAP2).

The *algorithmic language* provides access to a set of statements, procedures and functions for specifying any non-standard behaviour. It is close to PASCAL as far as expressions and statements are concerned. The algorithmic language includes a large set of system functions and procedures. These procedures may be used in a service description to specify modelling mechanisms, or in an /EXEC/ command to specify input/output operations or solver activations.

3 Modular Construction of Performance Models for OIS Architectures

During the architecture specification phase, OIS architectures suitable for the implementation of the office requirements resulting from the requirements collection and analysis (Ch. 2) and the logical design phase (Ch. 3), is represented as interconnected HW components, supporting the operations of a number of SW components. In general, there will be a number of architectures suitable to implement the office requirements, which may differ both in components being employed and the way components are being interconnected. In order to discriminate between several proposed architectures from a performance viewpoint, these architectures are modelled and evaluated in the performance modelling phase. As mentioned above, an objective of the performance modelling phase is to support modular construction of performance models for OIS architectures.

As mentioned before, a performance evaluation problem consists of three elements, namely *machine*, *load* and *performance measures*. In Section 3.1, we shortly characterise

the OIS architectures considered within TODOS, thereby focusing on the architectural aspects concerning *machine* and *load*, which are of primary interest during the performance modelling phase. This leads in Section 3.2 to the identification of a number of demands to the modelling techniques to be employed. The modelling techniques developed in order to meet these demands are described in the remaining of this Section 3.2.

3.1 Characterisation of OIS Architectures

Sharing of information and resources is a basic requirement in any OIS. Expensive hardware resources such as printers, plotters, disk and tape storage devices, etc. usually will be shared for economical reasons. Sharing of resources may be achieved by establishing a communication network among a number of computer systems, thus enabling communication between these system components. Within TODOS, the communication networks considered are restricted to Local Area Networks (LANs) (Part I), being the kind of networks typically employed in OISs. In OIS Architectures there is a trend towards *distribution of resources*. This means that resources will be distributed over the LAN, each point-to-point connected with a host machine, a so-called server, rather than being connected to one central mainframe machine. Resources will be accessed by office workers from personal workstations, so-called clients, which will typically provide their users with some local intelligence, this opposed to the convential dumb terminals, but with no or limited storage capacity.

In such an OIS enviroment, we have:

- workstations and PCs with no or limited storage capacity;
- file servers that provide archiving facilities; file servers have large storage capacity (disks, drums, tape drives);
- print servers that enable the printing of documents on connected (laser) printers;
- database servers that enable access to shared data bases;
- mail servers that enable distribution of documents among office workers;
- local area networks for communication between the different machines of an office system;
- gateway servers that enable communication with other networks and possibly enable connections to a central computer (mainframe).

3.1.1 OIS Applications

Within TODOS, all required OIS services are supposed to be provided by application software packages. Some of these are implemented as personal support tools, i.e. they are present on every workstation on the network when required by the office workers. One may think here of a word processing package or a personal data base management package. From a performance modelling point of view, these personal or single-user applications are not very interesting. They will generally be used by only one user at a time, namely the one working on the workstation on which the tool is implemented. Consequently, there is no contention involved in these applications. The response times that the user experiences simply consists of the sum of the service times that logically constitute the applications. The performance of the single-user applications depends

therefore only on the service times of the transactions, and may be conveniently evaluated by taking a stopwatch and measuring elapsed times of transactions.

Performance evaluation based on queueing network models is only useful for applications in which there is contention for resources. In the office environment considered in TODOS, these will generally be (distributed) applications which involve accessing of shared resources, such as storage devices or printers, which are connected to a server.

For instance, a computer with a disk connected to it may, according to the way data are structured on the storage device, act as a file server (when the disk contains file systems), a data base server (when the disk contains a shared data base) or as a mail server (when the disk contains users' mailboxes), while a printer is connected to a print server. Obviously, a computer can act as both file server, mail server and print server, as well as any combination of these.

To provide the service to the clients connected to the network, software components are implemented on both the clients and the server. On the server, a software package, called e.g. the file service or the print service, provides the actual access to the resource. On the client, a software package provides the service directly to the user and the user applications. These application software packages communicate using the transport service which is provided over the Local Area Network.

The transport service over the LAN therefore plays a central role in the OIS architectures considered within TODOS. Consequently, the transport service will play the same central role during the construction of performance models for the distributed OIS applications.

Consider as an example a file service. Figure 2 shows the actions performed when a client requests to read a file from disk.

After the call from the user (application) and some local computing (on the client cpu), a request to read the desired file is delivered to the transport service for transmission to the file server. There, after some local computing, the disk access is prepared and executed after which the response containing the file is returned to the client, again by means of the transport service.

3.1.2 Local Area Networks

A LAN is a data communication network limited in geographical scope. It provides high-speed data transmission at relatively low costs compared to a Wide Area Network (WAN) and therefore is typically employed in OISs.

A communication network can be seen as having a typical hierarchical structure which can be visualised by the Reference Model (RM) for Open Systems Interconnection (OSI), proposed by the International Standards Organisation (ISO) (ISO 1983), (Zimmerman 1980). The RM identifies seven layers which together form an open system, but in specific implementations of open systems some of the layers may be (nearly) empty, especially in LAN-environments. The LAN model employed for performance modelling is based on a simplified version of the RM. Only four layers of each site are explicitly represented : the application layer, the transport layer, the data link layer and the physical layer (see figure below).

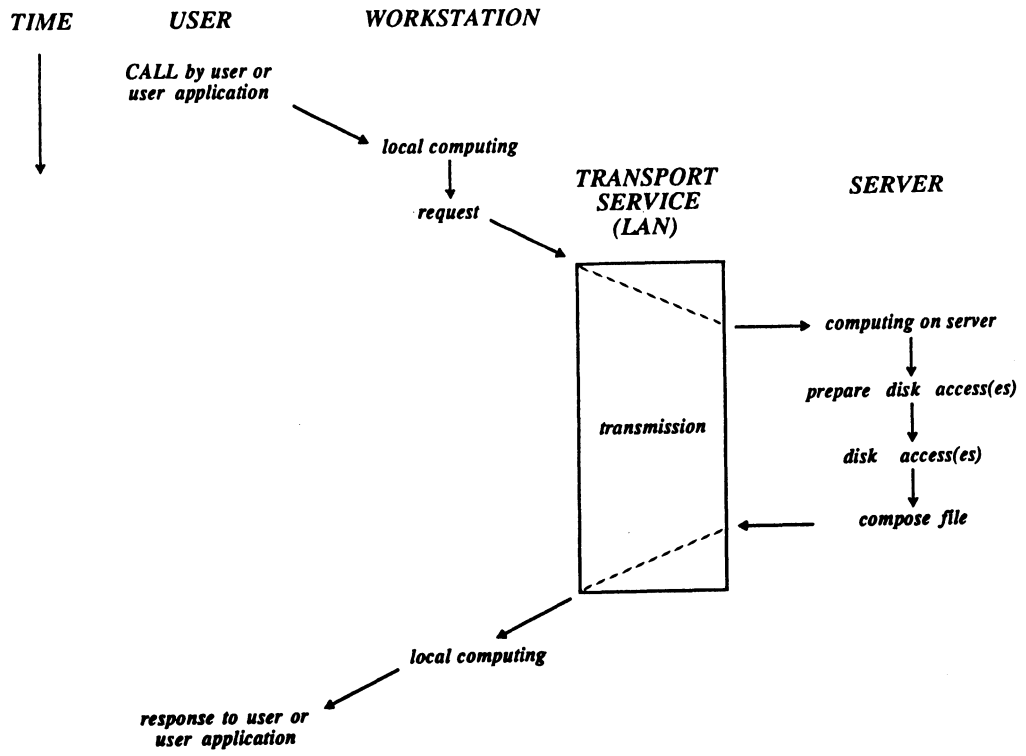


Fig. 2 : A Remote File Read Operation

In the model, the application layer relies directly on the transport service of the network, which in terms of the OSI-RM is the service provided by the layers up and including the transport layer. Examples of application layers to be inserted into the performance model for OIS architectures are a file service and a database service.

In the OSI-RM, two aditional layers may be identified between application and transport layer, being the presentation and session layer. The presentation layer performs functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems (e.g. text compression). The session layer is the user's interface into the network. It is with this layer that the user must negotiate to establish a connection with a process on another machine. In the model presented here, both presentation and session layer are empty. It is assumed that the tasks of these layers are performed in the application layer.

The model of the transport service over a LAN serves as a framework for the modelling of the distributed applications, which all rely on this transport service. The

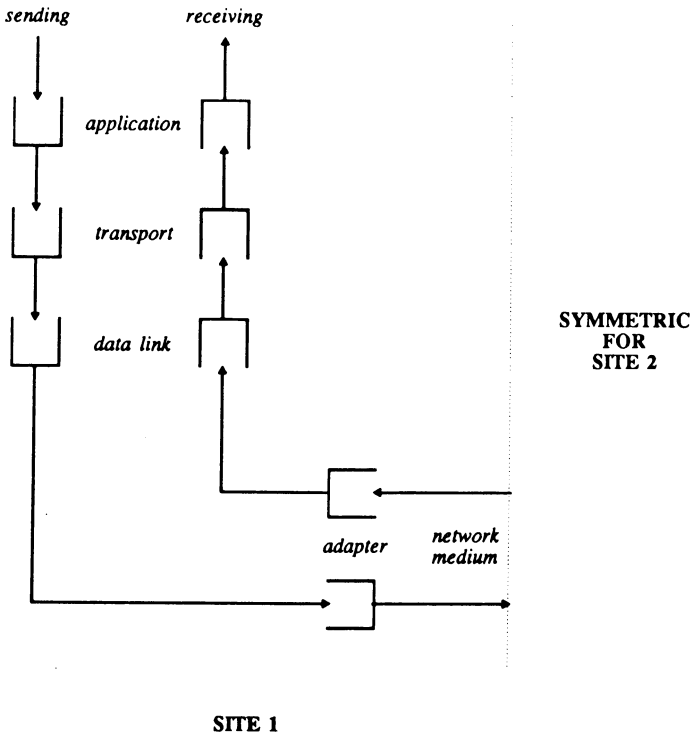


Fig. 3 : Layered queueing model of a LAN

transport layer provides a reliable end-to-end transfer of application messages. Segmentation of a message into packets takes place here. In a first version of the LAN model, the network layer is assumed to be empty. This is not unusual in OIS architectures since, the network layer only plays a role when more than one LAN are connected via a gateway server. The data link layer takes care of sending single packets over the medium. The LAN-adapter represents the physical layer and controls transmission of the packets over the network medium.

The transport service on each site relies on two hardware resources, a processor (cpu) and an adapter. The processor is the server for both the transport layer and the data link layer. The transport and the data link layer may be regarded in this case as being implemented in a single software package, the network package. In most networks, the transport layer protocol is indeed part of the operating system, but the data link layer is in many cases (at least partially) implemented in hardware.

In this model, a communication session between two sites consists of a sequence of packets sent from one site to another, and a number of acknowledgements sent back from receiver to sender. Acknowledgements may be sent at both data link and transport level.

No piggybacked acknowledgements are considered here.
The protocol of a layer is implemented in a program that is executed for each packet that visits the layer. An assumption made here is that the protocol code of both the transport and the data link layer is non-reentrant, meaning that the code can only be executed for one packet at a time. This is a usual implementation for the network package of a LAN.
Moreover, it is assumed that each site of the network may have only one single outgoing and one single incoming connection at a time, so each site can send at most one message at the time, and can also receive at most one message at a time. Again, this represents a typical case in a LAN.
A further assumption is the absence of errors (e.g. transmission errors). This assumption can be justified for a model of the transport service of a LAN since errors are usually very rare in such an environment.

3.2 Modelling Techniques for OIS Architectures

The objective of modular construction of performance models for the OISs considered within TODOS imposes two major demands on the modelling strategy employed.

First, there should be independent modelling of all architectural components involved, i.e. HW and SW components should be modelled independently of each other.
So, applied to the transport service of a LAN, there should be independence between the software which implements the transport and data link layer (the network package) and the hardware resources, being the cpu, the LAN adapter and the physical transmission medium. It should be possible to change or replace any of these components without affecting the other ones.

Second, since the transport service acts as a basic service on which all distributed applications in an OIS rely, independence between the transport service of the LAN and the application layers on top of it should be guaranteed. Replacement and addition of applications should be possible without the need to change the transport service and other applications.

3.2.1 Independence Between Hardware and Software Components

In a straightforward queueing network model this independent modelling of architectural components is not present. In general, the hardware is represented by servers (or stations) in the model, thus forming the actual queueing network. Software components, or more precisely processes executing program code, are then modelled as customers circulating through this network. The effect of the software is embedded in the description of the servers by adjusting the service descriptions and routing information to reflect the software workload, i.e. the aggregate demand of all software processes which use the service of the hardware resource represented by the server. The following technique is developed to achieve the desired modularity of the queueing network model.

Software components are modelled in a (number of) dummy station(s), in which the "protocol" of the software component is modelled. The actual execution of the code will naturally be simulated at the appropriate station, i.e. the server representing the hardware resource whose service is required by the software package.

In the following we take again the transport service of a LAN as an example. The network package, which contains the code of both the transport and the data link layer, will accordingly be represented by a number of stations for both layers separately. Each layer consists of some dummy stations in which the protocol is modelled, e.g. handling incoming and outgoing packets, updating the window counters, generation and handling of acknowledgements, blocking of the transmission in case of full windows etc. The execution of the code will then be simulated at the processor of the network site.

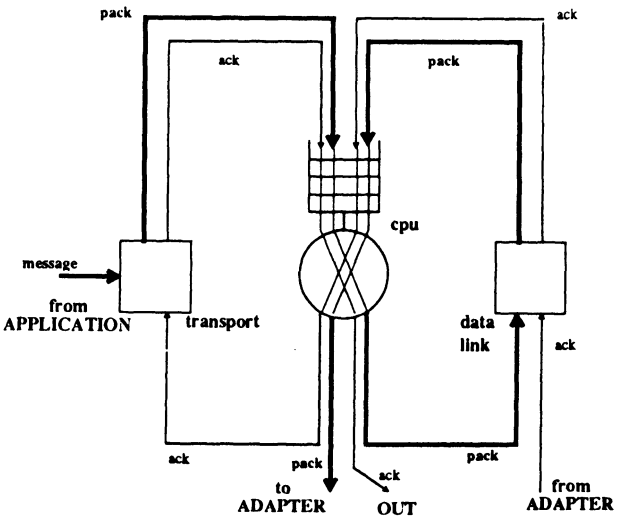


Fig. 4 Communication process at sending site.

To achieve an independent modelling of components, processing requirements at the processor to execute a particular part of the layer's protocol, and information to which station a customer is to be routed after receiving the service at the processor, are attached to the customers in specially defined attributes. These attributes will get their value in the dummy station modelling that particular part of the software protocol. Using the QNAP2 specification language, they may be defined as follows:

```
/DECLARE/
CUSTOMER REAL pr;
CUSTOMER REF QUEUE next;
```

The real *pr* represents the number of instructions the processor must execute for a layer's protocol code, the queue reference *next* refers to the queue the customer, which represents a process executing a software protocol, is routed to when leaving the processor. For each layer of the network package, the correct values are substituted in the station in which the protocol is modelled. There, the execution of the protocol code is simulated (the number of instructions to be performed is equal to pr) and the routing to the next layer is performed (the correct queue has been substituted in next).

The description of the cpu may consequently be very simple, and completely independent of the tasks it has to execute:

```
/STATION/
NAME=cpu;
SERVICE=EXP(pr);
TRANSIT=next;
```

Adding or replacement of SW components to the model will not affect the description of the cpu. Applying this modelling technique consistently to all HW and SW components, the required independence between hardware and software has been achieved.

3.2.2 Independence Between Applications and Transport Service

To enable a fast and efficient replacement or addition of applications in the queueing network model, the transport service should be modelled as a black box, meaning that the application layer should not be aware of the way the transport service is provided, only of the service itself which is the sending of a message to a remote destination on the network. To accomplish this independence between the application layer and the transport service, a customer is enhanced with two attributes :

```
/DECLARE/
CUSTOMER REF QUEUE des_appl;
CUSTOMER REF QUEUE ret_appl;
```

As explained above, the OIS architectures considered consist of a number of client workstations attached to a LAN, to which also a number of specific servers (file servers, database servers, print servers, etc.) workstations are attached. When an application process that is running on a client wants to access the service of a remote server, a request is sent to this server using the transport service over the LAN. The two defined attributes are substituted to perform this sending : des_appl will contain the queue in the queueing network that represents the application on the remote server that is accessed, and ret_appl will contain the queue to which the response of the remote server has to be returned. In the remote server, the attribute des_appl will be set to ret_appl. This implies that the transport service always delivers a message at des_appl and needs to know nothing about whether the traffic is client to server or the other way around. This modelling technique ensures the required independence between the transport service and the applications over a LAN.

The two modelling techniques described above, enable modular construction of performance models for OISs. Both aspects are achieved by adding attributes to the predefined QNAP2-type CUSTOMER. Some additional characteristics, e.g. the non-reentrancy of software modules, are modelled using semaphores. Since these QNAP2-mechanisms are being used, the only way to evaluate the resulting queueing network model is through simulation.

4 Performance Evaluation in the TODOS Methodology

The objective of the performance modelling phase within the TODOS methodology is to support the choice of the best alternative among a number of proposed OIS architectures. As has been described in (TODOS TR 1.2.3.1), the selection process consists in fact of three parts:

- definition of the requirements that should be met by the OIS to be implemented;
- evaluation of proposed OIS architectures;
- comparison of proposed OIS architectures.

In the TODOS methodology, the requirements are defined in the requirements collection and analysis phase (Ch. 2). These requirements will be stored in the TODOS Requirement Model (TRM), which combines the TODOS Descriptive Model (TDM) and the TODOS Performance Model (TPM). The TDM allows to describe the real world of offices and mainly deals with functional requirements, while the TPM is the model for the description of objectives and constraints to be met by the new system and mainly deals with non-functional requirements.

Following the TODOS Design loop through the logical design (Ch. 3) and rapid prototyping phase (Ch. 4), it is the task of the architecture design phase (Ch. 5) to come up with (a number of) architectures which meet office requirements and to evaluate the performance of suitable architectures with respect to the office workload. Correspondingly, the two major activities that may be identified during architecture design are architecture specification and architecture performance evaluation, both being executed by the same (group of) system designer(s). Architecture specification considers functional aspects, delivered by logical design in the TODOS Conceptual Model (TCM), as well as a number of requirements specified in the TPM, such as physical aspects of the equipments, part of the logical aspects (e.g. adequacy to user experience), cost aspects and compatibility aspects.

During Performance Evaluation, quantitative values are being assigned to the performance measures, which are of interest for the particular OIS architectures being proposed during architecture specification. The performance measures to be delivered by a performance modelling study within the TODOS methodology are twofold. On the one hand, the requirements specified in the TPM in terms of user-oriented performance measures (response times for and throughputs of user transactions) should be reflected. On the other hand, a modelling study should deliver system-oriented performance measures (utilisation of and queue lengths for system resources), a system designer might be interested in in order to identify critical parts of a proposed architecture.

As described in (TODOS TR 1.2.3.1), after performance evaluation the resulting user-oriented performance measures for each of the proposed architectures will be stored in the TPM. The final task of the selection process, the comparison of the proposed architectural solutions, will then be supported by the TPM, which will, according to pre-defined judgement rules, automatically compute marks for the different solutions for all of the defined requirements.

Having concluded the general description of the selection process in order to come up with the best alternative among the proposed OIS architectures, we are now in a position to explicitly state the interfaces of the performance modelling phase with the other activities

within the TODOS methodology. First, we consider the activities which will give inputs to the performance modelling phase. The most obvious one is clearly the activity immediately preceding the performance evaluation, namely the architecture specification phase.

Another source of input for the performance evaluation, which has already been mentioned, is the TODOS Performance Model from Requirements Collection and Analysis phase. In the TPM, among others, requirements for a number of user-oriented performance measures, which have to be met by the OIS under design, are stored. In order to check whether these requirements are actually met by the proposed architectures, the performance modelling phase obviously should deliver these measures. A final source of input is the TODOS Analysis Model (TAM) (TODOS TR 1.2.2.1), which will contain a characterisation of the office workload. An accurate characterisation of the workload which will be applied to the OIS is of the first importance, since obviously the performance of any system may only be determined with respect to the work it should be able to support, and consequently a performance modelling study which uses an inappropriate workload model will never be able to give meaningful estimates for the performance of the actually implemented system. More details on the required workload characterisation will be given in the section 5.

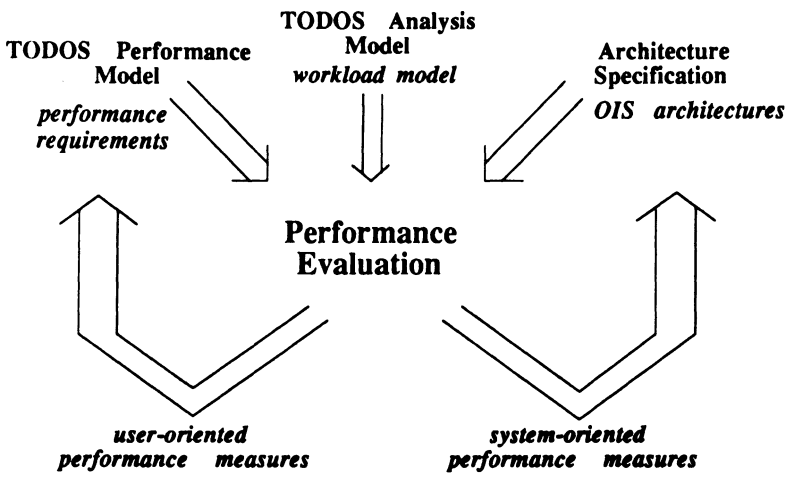


Fig. 5 Interfaces of the Performance Modelling Phase.

The first of the outgoing interfaces of the performance modelling phase has already been mentioned, namely the delivery of the required user-oriented performance measures for storage in the TPM, which will then support multi-criteria comparison of the proposed architectural solutions. Furthermore, outputs in the form of system-oriented performance measures may provide feedback to the architecture specification phase. Interpretations of resulting measures for utilisations of and populations at resources may thus lead to

reconsideration of the proposed OIS architectures. The way in which performance evaluation may support architecture specification will be further detailed in the presentation of the methodology to be employed for conducting a performance modelling study within TODOS.

The interfaces of the performance modelling phase and other activities within the TODOS Design Methodology are graphically presented in figure 5.

5 Conducting a Performance Modelling Study

The procedure of developing accurate performance models for the OIS architectures proposed in the architecture specification phase is not an easy task. Because of the complexities involved in the modelling process, methodological approaches are very beneficial (McNair et al. 1985). In the following, a methodological approach to the performance modelling phase within the TODOS design process will be presented. In this presentation, the performance modelling activity, which has been presented so far as a single activity, will be divided into a number of phases and the interfaces to other activities, which have been identified above, will be further detailed.

A performance modelling study may be divided into several phases:

1) Understanding the system design;
2) Definition of objectives;
3) Identification of relevant system components;
4) Model construction;
5) Parameterisation;
6) Model solution;
7) Interpretation of results.

The performance modelling phase within TODOS serves to support the choice between several proposed OIS architectures and will therefore in general involve the construction of several performance models, representing (part of) the proposed architectures. Obviously, this implies that some of the steps will have to be taken explicitly for each of the models to be constructed. To be more precise, this holds for the steps 3) to 6). As will be explained in the following, the steps 1) and 2) as well as 7) will be mainly concerned with the set of proposed architectures as a whole.

This enumeration of phases may give the impression of a linear process. It must however be emphasised that in practice a performance modelling study will be a highly *iterative* process. In addition to feedbacks to earlier phases due to discovery of errors/incompleteness, this iterative nature may in many cases already be imposed by the objective of the modelling study. Part of a modelling study may be a *modification analysis*, with the objective of assessing performance implications of architectural and/or workload changes. Such changes may be reflected in changing parameters (in the case of workload changes), or even in model changes (e.g. in the case of replacement of certain architectural components). Another integral part of a performance modelling study may be a *sensitivity analysis*, which has the objective of identifying critical parameters. It involves trying multiple values for a (set of) parameter(s) and observing the magnitude of changes

in the performance measures. If the results change quite a bit with only small changes in the parameter, the value of this parameter is critical, indicating that this parameter might need special attention.

The complete performance modelling cycle is depicted in Figure 6.

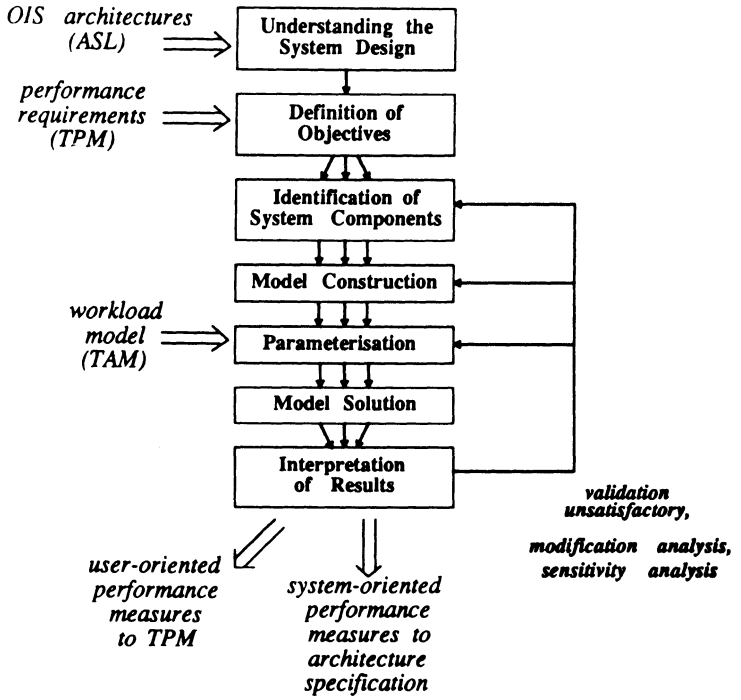


Fig. 6 The Performance Modelling Cycle.

5.1 Understanding the System Design

A first pre-requisite to construct an appropriate model of the system under study, is clearly that the modeller is familiar with the system design. Since the model is an abstract representation of the system, it must contain enough information to behave similarly to the real system as far as the performance aspects of interest are concerned. Of course, a model can be built at many different levels of detail, but even the highest level of detail must incorporate some of the complexities of the real system.

In the performance modelling phase within the TODOS methodology, the performance of a number of proposed architectural solutions will be evaluated in order to support the choice between these solutions. Consequently, the performance models constructed should be able to discriminate between the proposed architectures. Note that the performance modelling phase will be directed by the same (team of) system designer(s) who performed the architecture specification, which implies that enough (expert) knowledge on the systems

under study will be available. Therefore, this first activity of the performance modelling phase will not require extensive system study. Emphasis during the activity should be on making explicit the differences between the proposed architectures.

5.2 Definition of Objectives

A good understanding of the objectives of a performance modelling study is, though perhaps less obvious, as important as a thorough understanding of the system itself. The objective of a modelling study namely determines the *level of abstraction*, or level of detail, to be employed. Many system characteristics that would need to be represented in a fully general model may be irrelevant in a particular study, and identification of such characteristics may consequently lead to a simpler model and modelling study.

In general, the objective of the performance modelling phase within the TODOS methodology is to support the choice of the best alternative of a number of proposed OIS architectures. A TODOS performance modelling study should address the critical factors of the office's business, which were identified in the TODOS Performance Model. This might be the file service in an environment where office workers need frequent access to documents from shared archives, or the print service in an environment where it is vital to be able to rapidly submit mailings to clients. As explained above, this implies that the performance modelling phase should for each of the proposed architectures deliver user-oriented performance measures related to the performance objectives described in the TPM.

Furthermore, the performance modelling phase may support a system designer in identifying critical parts of the proposed architectures, which might need special attention. In order to get a better feeling for these critical parts, a system designer might be interested in certain system-oriented performance measures, which give information about utilisation of resources and queue lengths for these resources. Performance evaluation may therefore lead to reconsideration of proposed architectures. Accordingly, it may be also be employed to eliminate uncertainties when making rough architectural choices (e.g. centralisation vs. distribution of resources, personal usage vs. sharing of resources) during the architecture specification phase. Performance evaluation may thus lead to an early reduction of the solution space during architecture specification.

The level of abstraction to be employed during the performance modelling phase of TODOS is largely determined by the differences between the several proposed architectures. Performance models used to compare rough architectural choices, such as the choice between centralisation or distribution of storage devices, will in general be specified at a high level (system level), and will include subsystem units as basic building blocks. When performance models are used to discriminate between types of devices in a similar architecture, e.g. the choice between two types of storage devices in the same architecture, it may be necessary to descend into a lower level, specifying the model in terms of basic hardware units.

Summarising, the result of this activity should be a definition of the questions the performance modelling phase should answer and an identification of the abstraction level(s) to be employed during the construction of the performance models for the proposed architectural solutions.

5.3 Identification of Relevant System Components

The next step to be performed after describing the objective of the performance modelling study, and thus the level of abstraction to be employed, is the identification of the system components to be included in the performance models to be constructed. Note that this activity should be carried out for each of the performance models to be constructed. Obviously, this step depends heavily on the previous step.

The architecture specification phase is concerned with modular construction of OIS architectures, based on existing HW and SW components, while the performance modelling phase is concerned with construction of performance models for these architectures. One should note however that this does not imply that necessarily all architectural components will be represented in the performance model. As explained above, the objective of the modelling study will largely determine what applications should be modelled, and thus which of the HW and SW components should actually be included in the performance model. Furthermore, the objective of the study determines the level of abstraction to be employed, so while in some cases all relevant system components should be represented individually, in other cases the model could be restricted to a representation of subsystem units. These considerations imply that the performance evaluation tool will be manually driven in a sense that the system designer should decide which components to include in the performance model.

5.4 Model Construction

While the former phase was concerned with *which* system components should be represented in the performance model, the model construction phase deals with *how* to represent them. In this phase, the actual abstraction from the 'real' system to the queueing network has to be performed.

An objective of the model construction phase within TODOS is to provide a methodology for constructing performance models of complex OIS architectures out of 'simple' performance models of the architectural components. In order to achieve this desired modularity, the modelling mechanisms of QNAP2 have been enhanced as has been described in section 3.2.

5.5 Parameterisation

The model constructed in the previous phase will contain a number of parameters that will have to be instantiated before performance measures may be obtained by solving the model. This parameterisation is an important part of the modelling study, since obviously results obtained from model evaluation can be no more accurate than the parameter values provided.

Queueing network model parameters may be divided into two groups, parameters that specify the service centers, or machine, and parameters that specify the customers, or workload. Service center parameters should specify the service rate of a resource and the scheduling discipline of a resource. These parameters are integral properties of a resource and will therefore be part of the performance description of a HW component, as was described in (TODOS TR 4.1).

Workload parameters should specify service demands of customers at resources, visit ratios at resources, and (in case of a closed queueing system) the number of customers or (in case of an open system) the interarrival time distributions. Usually, the workload characterisation is the most critical element of the parameterisation.

In the TODOS methodology, the quantitative data needed to parameterise the workload parameters will be collected by the requirements collection and analysis phase to be stored in the TODOS Analysis Model. The workload in the office may be considered at different levels. First, we may consider the *real* office workload as imposed by office workers in terms of executed office support tool functions. This workload characterisation as such is not very useful in relation with service requirements to the shared resources, in which we are most interested during the modelling study. A workload description in terms of use of office support tools is useful though for a rough characterisation of the (expected) resource usage at personal workstations.

The level of workload characterisation that is most appropriate for our purposes will be in terms of apparent workloads, i.e. the workload observed as inputs to the system. The workload will be in terms of file handling, document printing, and document mailing. The workload characterisation at this level should represent the service requests to shared servers from the workstations of office workers. For example, with respect to file services these workload parameters should specify the (distribution of the) frequency of file accesses, or alternatively the mean time between two subsequent file accesses, and the distribution of document volumes involved. Clearly, accurate grouping of workload components is vital here, since these parameters may vary considerably between different work places in an office, or in some cases even between different office workers and support tools. For example, the user interface of office support tools, which governs the way of command input and results presentation, and user's experience with the tools he is working with, partly determine the time between completion of one request and issuing of the following, which usually is referred to as a user's think time.

Besides grouping, another important aspect of the workload characterisation is the period which it covers. Obviously, the office workload may vary from one period to another, and it should therefore be defined what workload should be considered during performance modelling. Usually we will be interested in performance measures for the period of time during which the workload is the highest, rather than on Friday afternoon, but in some cases it may be sufficient that performance requirements will be met for the "average" workload thereby accepting performance degradations during (short) periods of peak workload.

An approach to compensate for the lack of precise workload parameters is to perform a kind of sensitivity analysis by evaluating the model for a range of plausible parameters. This procedure may give indications of trends in performance measures under an increasing workload and of possible future bottlenecks in an architecture, which may in many cases be enough to discriminate between different architectural solutions.

5.6 Model Solution

In a performance modelling study supported by a performance evaluation tool such as QNAP2, a performance modeller does not need to bother about details of the solution method to be employed in order to obtain performance measures corresponding to the parameterised queueing network model.

The result of model solution using QNAP2 will be a set of basic steady-state performance measures (utilisation, population, response time, and throughput) for each of the stations in the network. Results may be presented in a standard report, but all performance measures may also be accessed individually using the QNAP2 result access functions. Consequently, they may be used for deriving other meaningful performance measures, such as the interactive response times for applications. QNAP2 thus offers its users the possibility to present resulting performance measures in the way they find most appropriate for further interpretation.

5.7 Interpretation of Results

Interpretation of results is an important phase of any performance modelling study. Reflecting directly to the TODOS performance modelling phase, two aspects should be taken into consideration during this activity. First, it should be checked whether the obtained performance measures actually represent system performance. This check may conceptually be made independently for each of the models. Once this first task has been satisfactorily executed, the second important step is to analyse the results with respect to the questions which were to be answered by the modelling study. This second task will in general require comparison of resulting performance measures for all models, and consequently, this task will be engaged with results of all models as a whole, rather than with the results of each model separately.

Validation is the process of ensuring that the model produces correct results. Model results may contain different errors, and a number of sources of these errors may be identified. One particularly common source of error is caused by inaccurately estimating model parameters. Some of the parameters may turn out to be extremely critical in producing the model's performance measures. Slight errors in estimating their values may result in large errors in some of the performance measures. This can be a very difficult problem to deal with. Workload forecasting for systems under development, such as in the TODOS methodology, is a complicated task in which one has to deal with many uncertainties. Critical parameters may be found by performing a *sensitivity analysis*, as has been explained above. As mentioned before, performing a sensitivity analysis may also be an approach to compensate for the lack of accurate workload parameters. In the TODOS methodology, a sensitivity-analysis may be useful to discover performance-sensitive parts of the proposed OIS architecture. QNAP2 supports a sensitivity analysis by offering the possibility of solving a model repeatedly in a loop in which some of the model's parameters are updated.

A second source of error may be errors in the model itself. The structure of the model can be incorrect, some key resources may be missing from the model, or the model may contain logical errors. These type of errors are in general not as difficult to deal with as the errors in the parameterisation phase.

When evaluating a model with simulation, we must also be concerned with the statistical variability in the results. Since there is a certain randomness involved in sampling from distributions, this randomness is also present in the results. QNAP2 includes several facilities for confidence interval estimations to deal with this statistical variability of simulation results.

Once all models have been validated satisfactorily, the next obvious step to be taken is to make sure that the questions defined during the definition of objectives may be answered by analysing the resulting performance measures. With respect to the user-oriented performance measures, which are to be stored in the TPM to support the choice between different proposed architectures, it should be checked whether the obtained results for the different models are appropriate in order to discriminate between the different architectures. When results for different models are too close to discriminate between them, the question should be answered whether this is the case because of model construction at a too high level of detail, or simply because proposed architectures have similar performance for the applications under study. If the former is the case, the system designer(s) should go back to the identification of components in order to construct more detailed models which may achieve the discrimination between the architectures. If the latter is the case the choice between the architectures should be made solely on the basis of other aspects defined in the TPM.

On the basis of both user-oriented and system-oriented performance measures, the system designer(s) may get more insight in the suitability of the proposed OIS architectures to support the office work in the office under study. It may lead to the decision to do away with specific architectural solutions, while others may be reconsidered. The latter may be the case for architectures for which measures are very good (low response times, utilisations of resources below 50%, small queues for resources), which might indicate that an overkill of resources is present, as well as for architectures for which measures are less favourable, indicating possible (future) bottlenecks. As mentioned before, one should not only include the results of a single evaluation of a model in these considerations but also the results of for example a sensitivity analysis. The system designer might ask himself some *what-if* questions, such as "what will be the performance implications of a slower/faster processor at a server?", or "what will be the implications of replacement of a I/O device by a slower/faster one?". These kind of questions may be answered during a *modification analysis*, which is the process of changing a performance model to reflect system changes or anticipated workload changes. Obviously, the latter is also relevant for the OISs developed within TODOS, since the OIS will also be supposed to support the office work in times to come, although possibly upgrades/reconfigurations may be necessary in the future. In many cases, these model changes may be made by simply adjusting one or more of its parameters, to reflect a change in the service rate of a resource or changing demands for a certain service. QNAP2 supports this kind of modification analysis by giving its user the option of interactive solution of the model. The user, in the TODOS methodology a system designer, might take over control after solution of the baseline model, change some parameters, and solve the model with those changed parameters.

Results of a modification analysis may be fed back to the architecture specification phase and lead to proposal of modified and even new OIS architectures, which defines the

link back from performance evaluation to architecture specification which has been identified in a former chapter of this report. The ease of interactive modification analysis in QNAP2 may lead to the consideration not to specify a separate performance model for each of the proposed architectures, but rather to evaluate similar architectures on the basis of a single performance model by making small model adjustments or changing some parameters. It is up to the system designers and their experience to decide what strategy to employ during the performance modelling phase.

6 Example

This section serves as an illustration of the performance modelling process within TODOS. The OIS architecture covered in the example consists, as depicted in fig. 7, of a number of client workstations, a file server and two database servers, connected by a LAN.

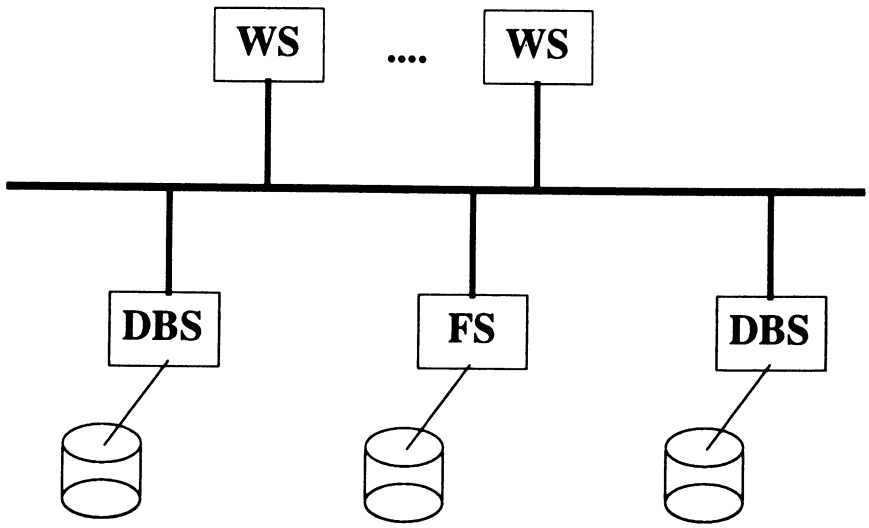


Fig. 7 Example of an OIS architecture.

The performance of this (class of) OIS architecture(s) will be evaluated as a function of an increasing number of connected clients. The performance will be described in terms of file service and database service response times, and utilisations of shared resources (file server cpu, file server disk, database server cpu, database server disk, and physical transmission medium).

In the following, the process of interactive model construction for all relevant architectural components is described. In this description, the focus is on the support of this process by the performance modelling tool. This support consists of ensuring model completeness by forcing the performance modeller to define all relevant performance components, and by giving default values for parameters. After that some results which

may be delivered by the performance modelling tool after simulation of the parameterised model are presented.

6.1 Global Configuration

First, the performance modeller has to specify the global configuration of the OIS architecture to be evaluated. At this point, the elements identified in this global configuration are the above-mentioned client workstations, file servers and database servers, which all will form a node of the LAN. Clearly, a node may be any combination of three elements, e.g. a combined database and file server, or a file server acting as a client workstation as well.

As a first question, the performance modelling tool will ask its user whether he wishes to vary the number of one of the elements and consequently have multiple simulations. In the case presented here, we indicate that we want to vary the number of client workstations. After this, the user will be prompted for the number of file servers, in our case 1, the number of database servers, in our case 2, and the number of clients. Since we have indicated that we would like to vary the number of clients, the tool will ask for the initial number of clients (we choose 10 workstations), the final number of clients (we choose 90 workstations), and the step size with which the number of clients should be varied between two consecutive simulations. We enter the value 20 for the step size, which implies that there will be 5 simulations.

Now that we have specified the global configuration of the OIS architecture, the performance modelling tool will guide us through the specification of all relevant performance components of the OIS under study. The following architectural components may be identified:

- Hardware components:
  - central processing unit (CPU); i.e. a processor, which clearly will be present at all nodes of the network.
  - disk subsystem; a disk will be present at both file and database servers.
  - LAN adapter.
  - physical transmission medium.
- System software components:
  - transport service; i.e. the software implementing transport and data link layer.
- Application software components:
  - file access application; this is the part of the file service running at client workstations.
  - file service application; this is the part of the file service running at the servers.
  - database access application.
  - database service application.

As an example, the figure below is a graphical representation of a performance model for a file server in which the architectural components may easily be identified. A connection between two architectural components means that the higher component uses the service provided by the lower one, e.g. the application SW component file service uses the CPU for processing and relies on the transport service for communication.

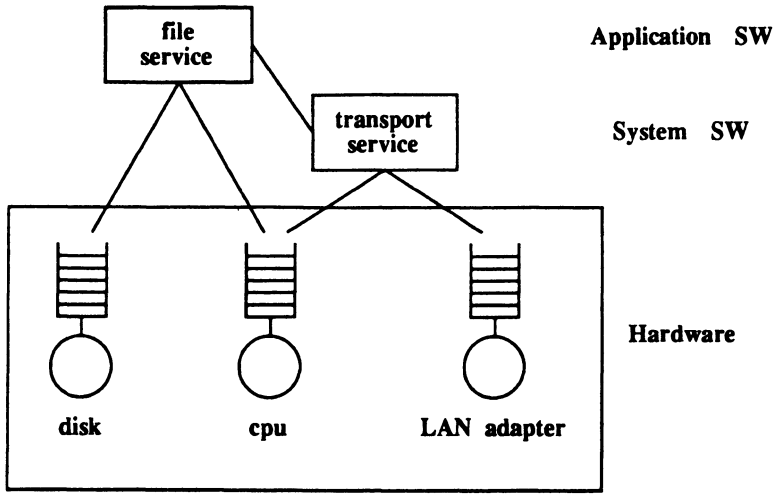


Fig. 8 Performance Model for a File Server.

6.2 Transport Service

Next the transport service over the LAN, including the physical aspects of the transmission medium, will be specified. The performance modeller is prompted for a number of parameters, but for each a default value is provided.

Parameters for the physical transmision medium are the LAN packet size (default = 1.0 Kbytes) and the LAN transfer rate (default = 10.0 Mbits per second, in other words Ethernet).

For the transport layer, the transport window size (default = 7), i.e. the number of data units a sender may send at transport level before suspending transmission and awaiting an acknowledgement, has to be provided. At the moment, only one transport layer implementation has been modelled, which has been obtained from (Meister et al. 1985).

The data link layer may be either connection-oriented (default) or connectionless. Only in the connection-oriented case, acknowledgements will be sent at data link level. In that case, the performance modeller should specify data link layer window size (default = 4) and whether or not there will be acknowledgement accumulation at the receiving site. If there is no acknowledgement accumulation, the receiving site will send a data link acknowledgement upon arrival of every packet, which is the default case. If there is acknowledgement accumulation, the performance modeller will have to specify the number of packets which will be collected at the receiving site before sending an acknowledgement. The performance modelling tool will check for the validity of the given values. For example, the value for the acknowledgement accumulation should be not larger than the data link layer window.

It may be noted that no parameters have to be specified for the physical layer, which will be represented by a LAN adapter. This is because up to this moment only a single performance component representing a LAN adapter has been included in the performance model base, whose characteristics again have been obtained from (Meister et al. 1985).

In this example we choose the default transport service configuration, that is a transport service with default values for all parameters.

6.3 CPU Configuration

Clearly, each node of the network will have a central processing unit. The performance modeller will have the option to have identical CPUs at all servers or have different CPUs at file servers and database servers. In case of multiple simulations, all client CPUs will be forced to be identical, since otherwise comparison of performance as function of a varying parameter will hardly be meaningful. In case of a single simulation, the performance modeller is free to define different CPUs at each site, for both clients and servers.

For each CPU to be defined, the performance modeller has to specify the scheduling discipline. At this moment, the performance modelling tool will present 4 options (first-in first-out, quantum scheduling, priority scheduling, and priority scheduling with a pre-emption distance of 2), which will all lead to different component descriptions. Furthermore, the performance modeller will have to specify the processor speed in MIPS (millions of instructions per second). Default value will be 1 MIPS, which represents the performance of an Intel 80286 (Heidelberger et al. 1988). The performance modeller is free to choose any processor speed, including both existing or realistic ones (e.g. a processor speed of 2 MIPS to represent a Motorola 68020), and more futuristic ones, for example in order to use the performance modelling study to obtain the necessary processing capacity to meet required performance for certain applications.

For our example, we will have CPUs which are priority-scheduling with pre-emption distance 2, and default processor speed of 1 MIPS at all sites.

6.4 Server Disk Configuration

Both file servers and database servers will have a disk subsystem connected to them. The performance modeller will get the option to have identical disks at all servers or have different disks at file servers and database servers.

At the moment, only a single type of disk subsystems has been included in the performance model base. This disk subsystem is characterised by two parameters, being the disk block size in Kbytes (default = 4 Kbytes), and the mean access time to fetch a block from disk or to write a block to disk (default = 20 msec.). The default values are typical for today's workstation technology, but again the performance modeller is free to choose whatever value he likes.

In our example we choose disks with a default block size of 4 Kbytes at all servers. The disk at the file server will have a mean access time of 15 msec., while the disks at both database servers will have the default mean access time of 20 msec.

6.5 Application Configuration at Servers

Up to this moment we have defined the hardware and system software configuration of the OIS architecture to be evaluated. Now we are ready to proceed with the application software components.

At file servers, a file service application will be present. For this application SW component no additional parameters will have to be specified. The performance component for this file service component has been obtained from (Lazowska et al. 1986).

At database servers, a database service application will be present. The description of the database service component has been obtained from (Heidelberger 1988) and represents a relational database with SQL. Two additional parameters will have to be specified for the database service. At the database server there will be a buffering of index and data pages in main memory, which implies that not all of the data required by a database transaction have to be fetched from disk. The buffer miss ratio may be defined as the mean percentage of pages which will not be present in the page buffer in main memory when requested and consequently have to be fetched from disk. Obviously, this buffer miss ratio depends on both database access pattern and the buffering strategy being employed, but (Heidelberger 1988) suggests 0.667 to be a reasonable value, which therefore is employed as default value. The performance modeller is however free to choose another value in order to adequately represent the database access pattern and the buffering strategy in the particuler OIS under study. Another parameter to be provided is the size of data pages employed in the database service, for which a default value of 2 Kbytes is proposed.

In our case we will choose the default values for both buffer miss ratio and data page sizes.

6.6 Application Configuration at Clients

At the client workstations, two types of application SW will run, being file access software and database access software. In case of multiple simulations, the application configuration at each client is forced to be identical, while in case of a single simulation the performance modeller has the option of defining the application configuration independently for each client workstation.

For the file access application, the performance modeller should first specify the distribution of file sizes accessed from a client workstation. At this moment three types of distribution have been implemented, namely constant file sizes, exponentially distributed file sizes, and a general file size distribution. For all distribution types, the performance modeller will have to specify the mean file size. For the general distribution additional parameters are the percentage of files smaller than 1 Kbyte, and the percentage of files between 1 and 4 Kbytes. A second parameter to be specified is the read/write ratio, which is the ratio between read accesses to files and write accesses of files.

In our case we choose the general file size distribution as observed in (Ousterhout et al. 1985), with a mean file size of 11 Kbytes, where half of the files is below 1 Kbytes, and approximately 25% between 1 and 4 Kbytes. Since usually file read operations will be more frequent than file write operations, we will choose the default read/write ratio of 3, meaning that the number of read operations will be approximately 3 times the number of write operations.

For the database access application, the performance modeller should specify the average database transaction complexity, i.e. the average number of SQL-calls per transaction. Obviously, the transaction complexity heavily depends on the application area. For example, according to (Heidelberger et al. 1988) the manufacturing transaction has a complexity of on the average 12 SQL-calls, while banking debit/credit transaction has a complexity of 5. In our case, we will use the latter transaction complexity.

6.7 Workload Characterisation

Up to this point, we have constructed the complete performance model for the OIS architecture under study, including all relevant components. Now we should characterise the workload in terms of server accesses. The workload characterisation consists of two elements. First, the mean time between two server accesses (or *think time*) and the percentage of these accesses going to the file service and the database service will have to be specified.

The workload in our example will be identical for all clients with a mean time between server accesses of 3 seconds. We choose 75% of the service requests to be to the file service, and consequently 25% to the database service.

Second, in case of multiple servers providing a service, which in our example holds for the database service, it should be specified how the accesses are distributed over the different servers. If there is going to be a single simulation, the performance modeller will have the option to define for each client the percentage of accesses to each server, but in case of a varying parameter a balanced load over all servers is enforced. There are three load balancing options. In the first option, the accesses for a particular service from a client will be uniformly distributed over all available servers providing that service. In the second option, all accesses from a client will be served by one server, the so-called *local server* of that client. In the third option, a certain percentage of the accesses of a client will be routed to the local server, while the remaining service requests are uniformly distributed over the remaining servers. When the latter option is chosen, the performance modeller should specify the percentage going to the local server (default = 0.8). The performance modelling tool will take care of establishing a balanced load of client requests to servers according to the chosen option.

For the database service in our example, we will choose the third load-balancing option with the default parameter.

6.8 Results Presentation and Interpretation

So far we have constructed a complete performance model for the OIS architecture under study, with attached to it a parameterised workload. This performance model is now ready for simulation. Before the simulation will be carried out, the performance modeller has some options with respect to the performance measures to be delivered by the simulation.

The performance modelling tool will by default present results for response times of service requests and utilisations of shared resources, but a performance modeller with some elementary knowledge of the QNAP2 algorithmic language and the structure of the constructed performance models, may easily derive additional performance measures from

the basic performance measures which QNAP2 delivers for each queue in the queueing network model.

All results may be written into a file, which implies that a performance modeller is not obliged to wait for the end of the simulation, which, if precise results are requested, may take several hours. The performance modelling tool will first show the configuration of the OIS architecture, which has been modelled, with all relevant parameters, so there will not be any confusion about the input parameters belonging to a particular set of performance measures

The performance modeller will get the option to have response times presented for each of the clients, or only have mean response times for all clients together. If all clients are identical usually the performance modeller will only be interested in the latter, but if client access patterns are not identical, response times per client will be mandatory. As in our example, all client workstations are identical, we will only present the mean response times of all clients taken together. A second option with respect to response times concerns the presentation of 95%-confidence intervals for the delivered response times. The performance modeller is free to suppress these confidence intervals. Usually a performance modeller will be interested in these confidence intervals, as they present a measure for the accuracy of the response times. If confidence intervals are not sufficiently small, simulation times may be increased. The performance modelling tool gives its user the possibility to change the simulation time per run.

With respect to utilisations, the performance modeller has similar options. He may choose to have utilisations presented for each server, or only have mean utilisations per group of servers providing the same service (e.g. mean utilisation for CPUs at database servers.). Furthermore, he has the option to either present of suppress confidence intervals for the utilisations.

As a final option the performance modeller has the option to store the results with respect to mean response times per service and mean utilisations per hardware resource of a service in a format appropriate for graphical representation.

The resulting response times of the five simulations and their corresponding confidence intervals are presented both graphically and in a table.

A number of observations may be made from these results. It may be easily seen that there is a considerable performance degradation with respect to the file service as a result of the increasing number of connected client workstations. It may therefore be concluded that the file server is saturating. This conclusion is clearly supported by the measures for the utilisations of the hardware resources, shown graphically in figure 10 and in table 2
For the database service on the other hand, the performance degradation is quite acceptable. With the increasing number of clients, the response time only shows a marginal increase, thus indicating that the database service in the considered architecture will not be a bottleneck even when the number of connected workstations is considerably increased.

From the measures for the resource utilisations, it may be concluded that network contention, reflected by the measure ptm (physical transmission medium), will not easily become a bottleneck in this OIS, since even with 90 workstations connected to the system, the utilisation stays well below 35%.

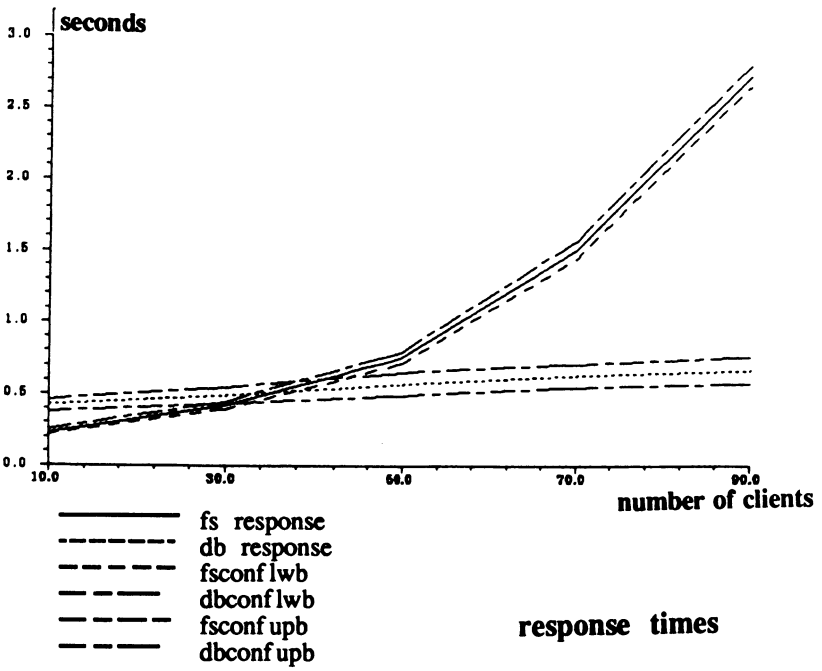


Fig. 9 Response times of file and database service.

RESPONSE TIMES		
# cl	*fs resp*	*dbs resp*
10	*0.2285*	*0.4188*
+/-	*0.01839*	*0.04091*
30	*0.4161*	*0.4856*
+/-	*0.02661*	*0.05677*
50	*0.7572*	*0.5652*
+/-	*0.04174*	*0.08129*
70	*1.5096*	*0.6272*
+/-	*0.06208*	*0.08015*
90	*2.7334*	*0.6667*
+/-	*0.07308*	*0.09348*

Table 1 Response times.

The performance measures delivered by the performance modelling study may be used to assess the number of workstations which may be connected to the LAN, while still having acceptable performance. Of course, 'acceptable performance' may be different in different environments, which will be reflected in different values in the TODOS Performance Model (TPM). Let's for example define acceptable performance in the OIS

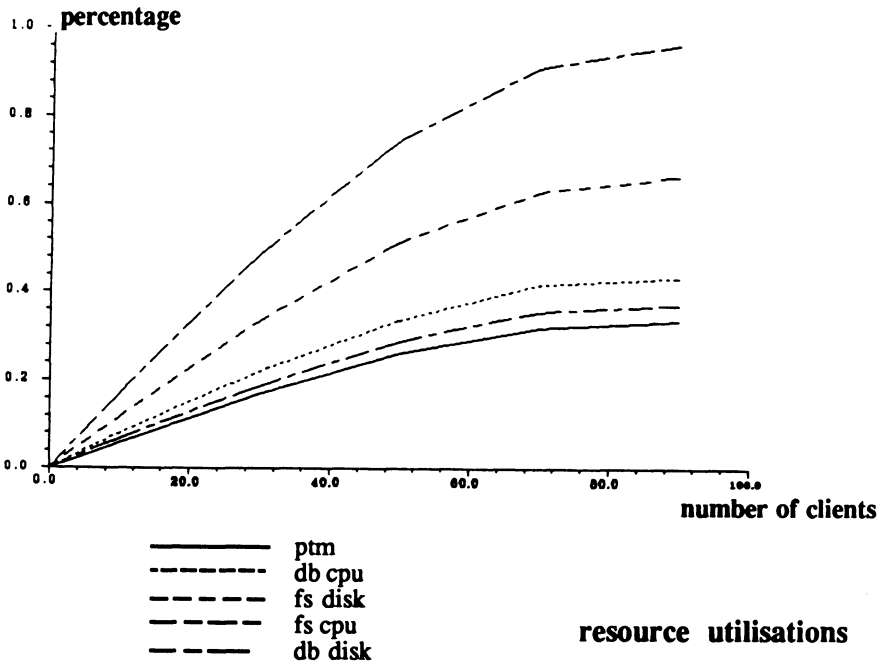


Fig. 10 Utilisations of hardware resources.

UTILISATIONS OF HARDWARE RESOURCES							
# cl	fs		dbs 1		dbs 2		ptm
	cpu	disk	cpu	disk	cpu	disk	
10	0.1661	0.1138	0.0773	0.0658	0.0783	0.0664	0.0582
+/-	0.00611	0.00487	0.00530	0.00472	0.00607	0.00500	0.00219
30	0.4873	0.3349	0.2151	0.1816	0.2248	0.1925	0.1706
+/-	0.00832	0.00731	0.00930	0.00819	0.01154	0.00952	0.00274
50	0.7479	0.5148	0.3294	0.2818	0.3447	0.2915	0.2616
+/-	0.00817	0.00615	0.01317	0.01230	0.01154	0.01143	0.00270
70	0.9107	0.6269	0.4145	0.3515	0.4204	0.3605	0.3202
+/-	0.00713	0.00612	0.01360	0.01356	0.01539	0.01336	0.00277
90	0.9618	0.6632	0.4267	0.3668	0.4409	0.3799	0.3369
+/-	0.00314	0.00331	0.01578	0.01379	0.01711	0.01419	0.00149

Table 2 Utilisations of hardware resources.

under study for the file service to deliver mean response times below 1 second. It may then be concluded from the simulation results that certainly no more than around 60 workstations should be connected to the LAN with the current number of connected servers.

Obviously, resources are not heavily utilised when only a few workstations are attached to the network. For the system under study, utilisation of all resources is still under 50% for 30 connected workstations, indicating that there will hardly be any queueing. A relatively low utilisation at all resources implies that overall performance still may be improved by reducing the service demand at any device. However, at high loads, e.g. more than 50 workstations connected to the LAN in the considered OIS architecture, the performance is totally governed by the most heavily utilised device, which in our case is clearly the CPU at the file server. It may be noticed that for 50 workstations file server CPU utilisation is already 75%, while from 70 connected workstations upwards the CPU is close to being fully utilised, causing performance to degrade rapidly with an increasing number of workstations. Now, performance may only be improved by a service demand reduction at the file server CPU. Any other modification is expected to have little or no result.

In order to improve file service performance, a number of design alternatives may be considered, such as:

- doubling of the speed of the file server CPU to 2 MIPS;
- add a second file server to the system.

It should be noted that these design alternatives may be easily derived from the original system, which in the following will be referred to as the baseline system, by some simple parameter changes. The first design alternative may be derived by indicating the file server CPU speed to be 2 MIPS when asked for it by the performance modelling tool, while the second alternative may very easily generated by answering the question with respect to the number of file servers with 2, which will lead to the construction of two file servers as part of the overall performance model.

The file service response times of these two design alternatives, compared to those in the baseline model, are graphically presented in figure 11.

It may be concluded that both design alternatives considerably improve file service performance.
Since in the baseline case the file server CPU was clearly the system's bottleneck, speeding up this CPU is an obvious modification. Clearly, it improves file system performance, though it does not entirely solve the problem of the file service saturation at high loads. From the estimates for the utilisations of the hardware resources, which are not displayed here, it may be concluded that in the modified systems the file server disk is the most utilised resource, and that this disk will consequently become the bottleneck device. Therefore, improvements to the disk subsystem, e.g. by adding a second disk, are likely to lead to better performance improvements than speeding up the file server CPU even more.

The second design alternative, in which a second file server is added to the LAN, to share the load imposed by the clients, clearly leads to the best performance improvement. It should be noted however that the qualification 'best' is only based on performace characteristics, and that cost and other aspects, which will influence the selection process between the several design alternatives, are not considered here.

The elaborated example above has illustrated that a performance modelling study, may clearly support the selection process between a number of proposed OIS architectures. A

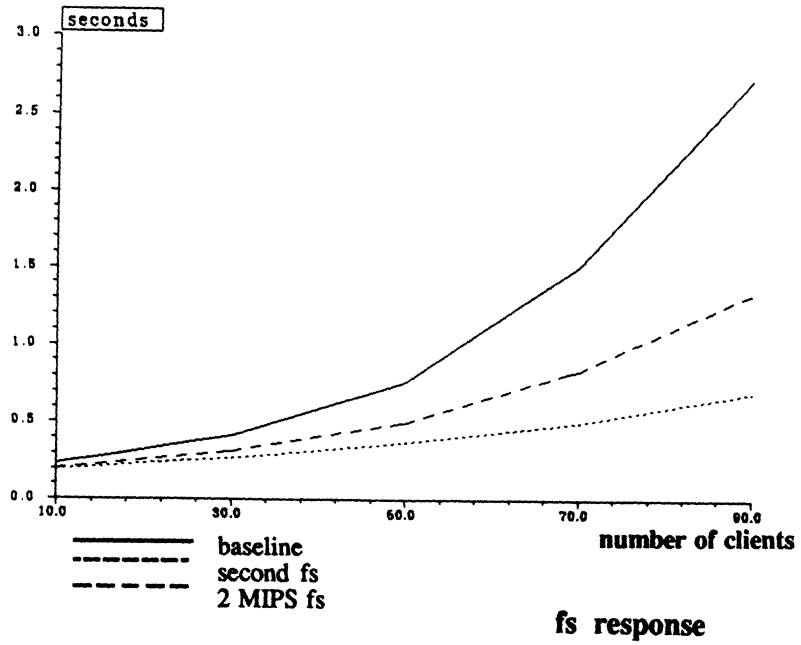


Fig. 11 File service response times for design alternatives.

number of relevant questions during this selection process may be answered more accurately after performance evaluation of the design alternatives.
Examples of such questions are:

Will expensive modifications (e.g. adding a second server, upgrading devices) be justified by an equivalent performance improvement, and a consequently increasing office workers' productivity?

Will the OIS be able to support a growing future workload (increase of the number of connected workstations, heavier use by current users)?

Chapter 6

TODOS Case study

Gerd Wolfram and Edda Pulst

1 Introduction

This chapter illustrates the application of the different tools prepared to support the TODOS methodology in a practical case study. The case study has the following objectives:

- The attainment of a natural, consistent application view including possible feedbacks from real users in order to ease further development. Testing the handling of the tools provides useful information for the user-interface. The case study demonstrates how the different tools and therefore parts of the new system can be integrated and work in an interrelated way.
- Practical use and application of the developed tools has an important impact on the evaluation of the research results attained.
- The case study demonstrates how developed functions and/or combinations of functions could be used and reused to provide the practical intersection between different design phases, each of them being concerned with a different step in the design process.

2 Background to the enterprise and case study field

The case study was chosen from a study carried out while using the FAOR methodology (Schaefer 1988). In the following section the main steps involved in applying the approach and the results are presented. First we describe some of the facts and background about the procedure and about the client enterprise situation in the case study.

2.1 The enterprise

The case study took place a **large regional bank** where the major tasks are financing, services, accounts, payments, lending and debentures. The bank has regional, national and international tasks. Special emphasis is put on international banking, portfolio management and building and loan management. The organization's branches are located at various places in Germany as well as abroad, furthermore the bank keeps intensive corresponding contacts with almost 5000 banks abroad.

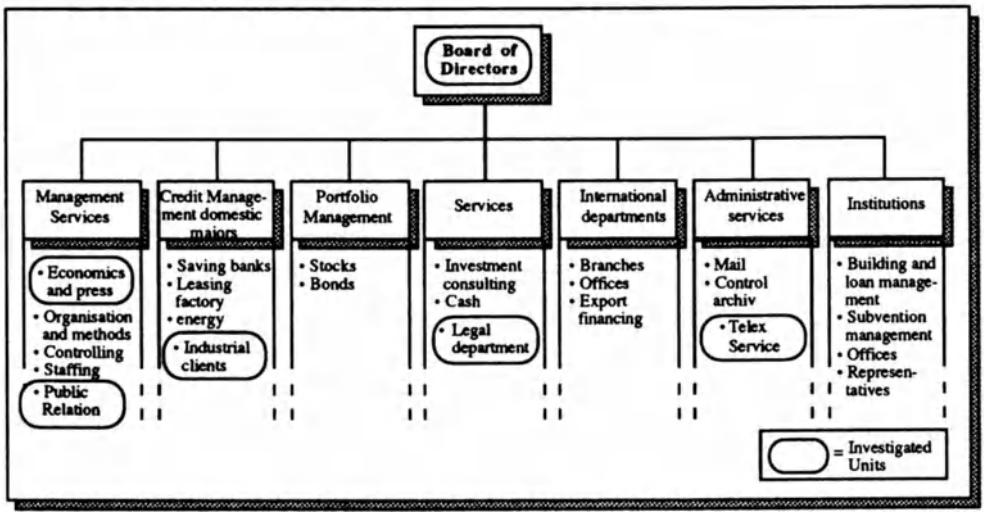


Figure 1: Organizational chart of the case study enterprise

The **banking organizational structure** may be roughly subdivided into 7 entities (Fig. 1).

The investigated units (marked) were the chosen departments where the FAOR application took place. 4.200 persons are employed in total. They can be classified into four different groups. The criteria for differentiation is the type of task or work they mainly perform (Fig. 2).

The **manager's** work consists of tasks like representation, leadership and motivation, problem-solving and decision-making. Besides, managers are often involved in technical work, e.g. in the acquisition of clients or the control of work. Managers are often outside their office. Their work consists to a large extent of communication (face-to-face or phone) and meetings. Managers have the tendency to write and read short texts. The storage of information is located at the workplace.

Experts work mainly in the service functions. They try to solve specific problems which can be characterized as innovative and difficult. Creativity and initiative are crucial qualities. Their work follows a task-orientation; special knowledge is required. Their task is to hand on information to people working in the market functions when necessary or to prepare reports on specific topics. Complex texts and documents, information storage and retrieval, communication with other knowledge workers and the usage of information systems are the main aspects of the office work.

The **clerical workers** are concerned mostly with standardized and administrative tasks in various business areas. They perform routine work. The work processes and the problems they solve are not innovative. Their work starts with folders and standardized documents.

Support tasks are fulfilled by secretaries and typists. They support the managers, experts and clerks. Their work is order-oriented.

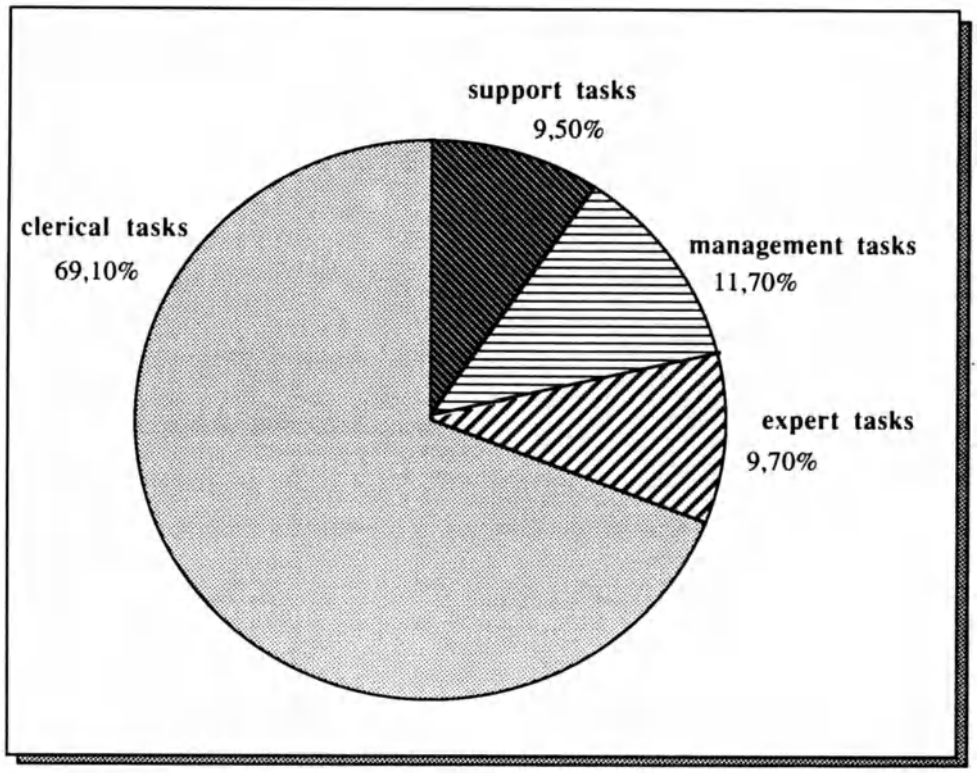


Figure 2: Classification of tasks within the case study organization

2.2 The Lending Process

The case study concentrated on the process of lending, especially to large enterprises. This process is of special interest to all types of tasks as it incorporates typical office activities ranging from unstructured to very structured ones. It is also very resource consuming in terms of office workers and equipment. Furthermore it provides a lot of functional requirements, i.e. desired speed of information transfer, quality of information and communication processes, one obtains useful hints regarding the final results and layout of each design phase.

The lending of money is one of the bank's prime business areas. Loans are given to private persons or small companies as well as to huge enterprises and trusts. While the lending business for private persons and small companies is more or less standardized, the individual contact with the client and the adaptation of the loan and its corresponding conditions to the client's specific needs is critical for the latter.

The lending procedure can be divided into **four sub-processes** (App. B) :

- **Acquisition:** During the acquisition, the first contact with the client takes place. The bank in general, its business and the lending conditions are presented. The effective marketing of the bank is critical as a basis for further and more detailed contacts with the client (e.g. advertisements in the daily newspapers).

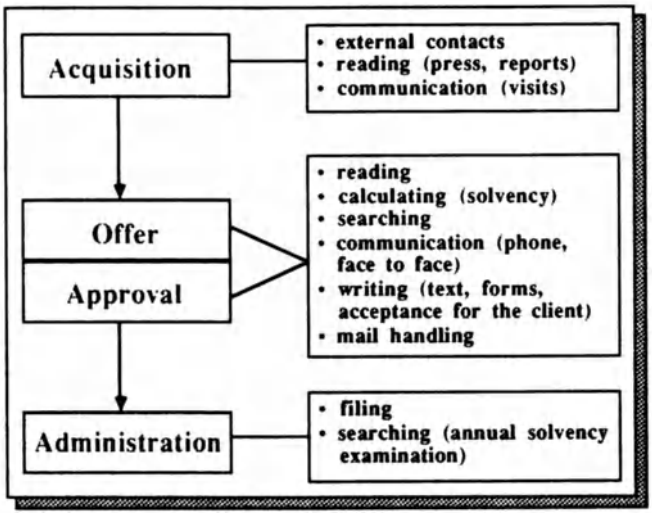


Figure 3: Phases of the lending process

- **Offering the loan**: After investigating the specific needs of the client and the possibilities open to the bank in offering a loan an individual offer is made.
- **Approval of the loan**: The main activities within this process involve verifying the documents submitted by the client and the final decision concerning the loan.
- **Administration**: The money is transferred to the client and the approved loan recorded (Fig. 3).

These sub-processes involve a lot of diverse **office activities**. The activities comprise having available external contacts and visiting the client, reading, face to face communication, calculating, filing, searching, handwriting and typewriting and mail handling. The lending process is characterized by the diversity of its documents like letters, short notes, spread sheets and forms.

The treatment of those organizations who are already clients of the bank does not differ from the treatment of potential clients even if there is the risk that the whole process is in vain and no contract will be signed in the end.

Several **organizational units** with diverse locations are involved in the various processes of lending (as indicated in Fig. 1):

- Credit department (external)
- Credit administration department (internal process)
- Economics and press department
- Public relations department
- Legal department

- Centralized typing pool
- Board of directors
- Telex service

The credit department (external) and credit administration department (internal process) are practically one and the same, the "external" section is responsible for the acquisition process involving the numerous clients' contacts whereas the "internal process" mainly concerns the internal administrative procedure within the bank. The accounting (payment of the loan) is undertaken by the accounts department located in the services department of the bank.

The public relations department is concerned with the image of the bank in public employs prospectuses, folios and advertising campaigns i.e. in newspapers and journals to promote its interests.

Generally speaking conditions of credit are always approved by the board of directors in order to allow flexible reactions to market changes (interest rates). If the acquisition manager decides to offer special (more advantageous) conditions to a client, for instance on account of his excellent solvency or long relationship with the bank, he arranges a personal meeting with the board of directors to gain acceptance for his proposal under special circumstances. This takes place before the acquisition letter is sent to the client.

The case study examined in detail the credit department, credit administration department, the economics and press department and the typing pool (Fig. 4).

The characteristic features lending process activities are due to the qualification and tasks of the people involved (Fig. 5).

3 Case study application

The present section describes the application of the TODOS methods and tools for each design phase. Each paragraph follows the same procedure: First the objectives and goals of the case study for each design phase are presented together with a short overview about the tool components and input of data. The usage of these data, their processing and the results are shown afterwards. Each paragraph concludes with a critical review of the tools and method within the design phase. Special emphasis is put on the interconnections of design phases, that is, the data exchange between design phases and their interrelated use and feedback.

3.1 Requirements collection and analysis

The TODOS tool for Requirements collection and analysis is termed "Office Data Dictionary" (ODD). As shown in Chapter 2. it provides a set of functionalities for the analysis process, e.g. storing and administrating requirements' data in a structured way. Report programs produce practical reports whether on screen or on printer for both the analysis and the judgment of architectural solutions.

The ODD is composed of several parts. The TODOS Structuring Tool (TST) acts as the focus administrating the data in the analysis. It provides the opportunity of storing and retrieving the components of the individual analysis. It contains information on the data structures and their use within the analysis. The TODOS Analysis Model (TAM) is

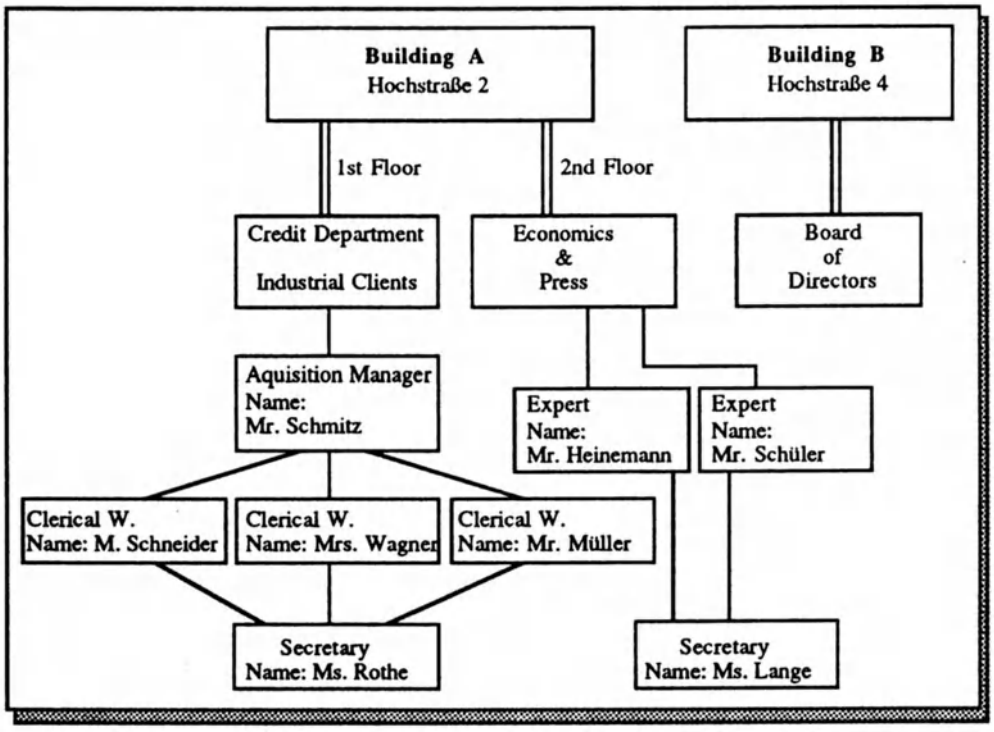


Figure 4: Location of departments involved

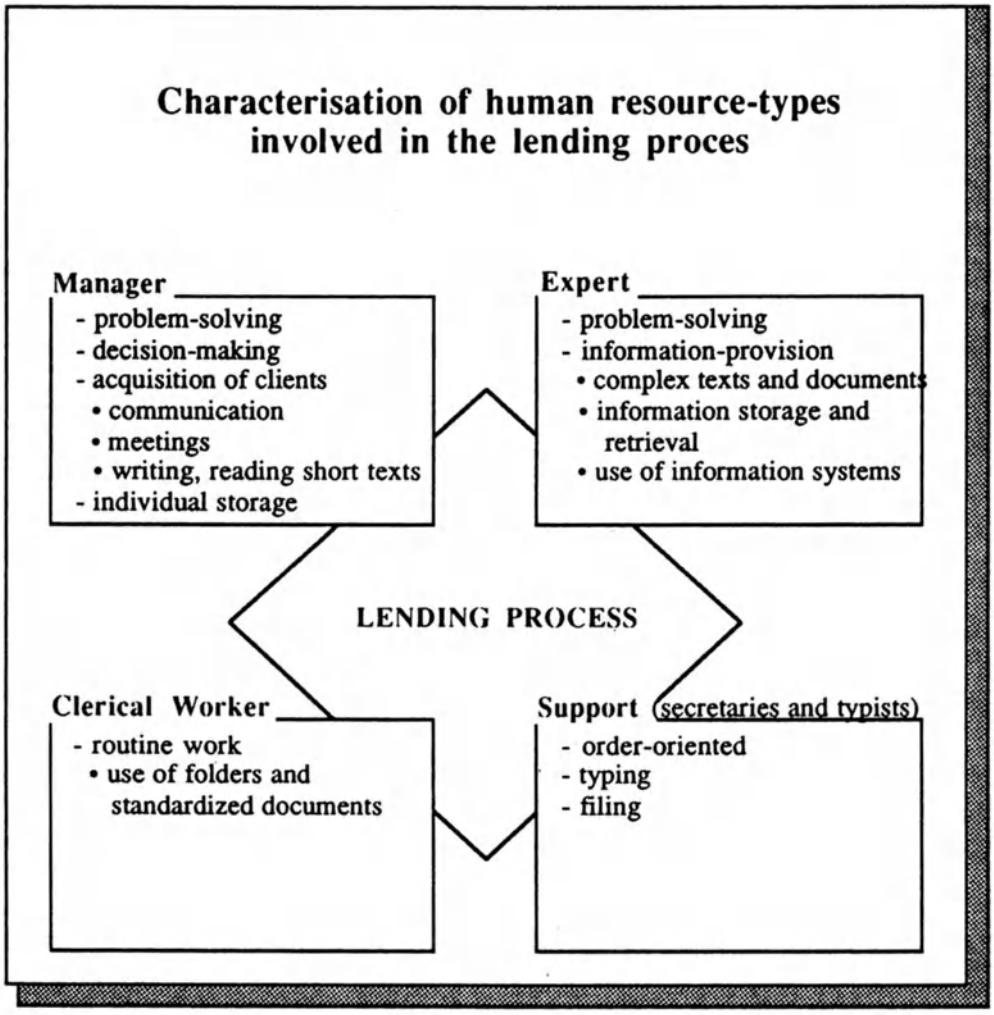


Figure 5: Human resources within the lending process

the user-interface to insert the data of the analysis into predefined screens. The TODOS Performance Model (TPM) finally is a model describing objectives and constraints to be met by the new system and therefore mainly deals with non-functional requirements like size noise, display etc. The TAM facility should be used for the feasibility study: Field data of the analysis are inserted in the predefined structure. Every data entry is checked against the specified domains property values within the TST. The concept of domain provides a consistent list of values corresponding to the terminology of the office studied. As previously described in Chapter 2 any addition of new elements of the investigated office can be easily done by enlarging the list of domains within the TST. However a lot domains do already exist in predefined libraries, each refers to a specific organization type (bank, assurance, EEC administration, ...). Within the case study it was therefore tested how easily new elements could be inserted into the ODD. The flexibility was successfully proved by inserting ideas of the users upon new functionalities which meant an expansion of the generic models of the tool. Furthermore data upon non-functional requirements (noise and workload) were inserted through the TPM (TODOS Performance Model) of the ODD. During the exploration phase (with respect to the FAOR framework which has been presented in Chapter 2) a broad understanding of the context and background of problems the client organization is facing should be obtained. Aspects which the analyst think relevant will be further explored and the findings structured and discussed with the client. Using TODOS requirements collection and analysis tool showed how the organizational preconditions could be inserted into the data dictionary for a more structured presentation. Furthermore the ODD was adjusted to the specific client situation.

As discussed in Sect. 2., the application of the case study data was restricted to 3 organizational units situated in various locations:

- Credit Department industrial clients
- Economics and Press Department
- Board of Directors

The organizational structure was inserted via the 1st screen of the TODOS Analysis Model (Fig. 6) indicating the name of the units involved as well as their addresses and overhanging units.

Units and their tasks (mission) were related to the people working in the organization via an appropriate mask which adds details of the professional status occupied by each person (Fig. 7).

Loans are given huge enterprises and trusts; this process is characterized by a balanced mixture of standardization and unstructured work.

Within TAM the sub-processes of the lending process were defined in the form 3 product-objectives: acquisition, definition and approval of the offer, administration in order to be inserted via the "Product-objective-screen" of TAM (Fig. 8). A first evaluation of the various processes (formal and support level) provided useful indicators for the analysis phase.

Significantly, the case study provided many different office activities which comprise among other features external contacts with the client, a number of internal contacts and meetings, activities like reading, face-to-face-communication, calculating, searching,

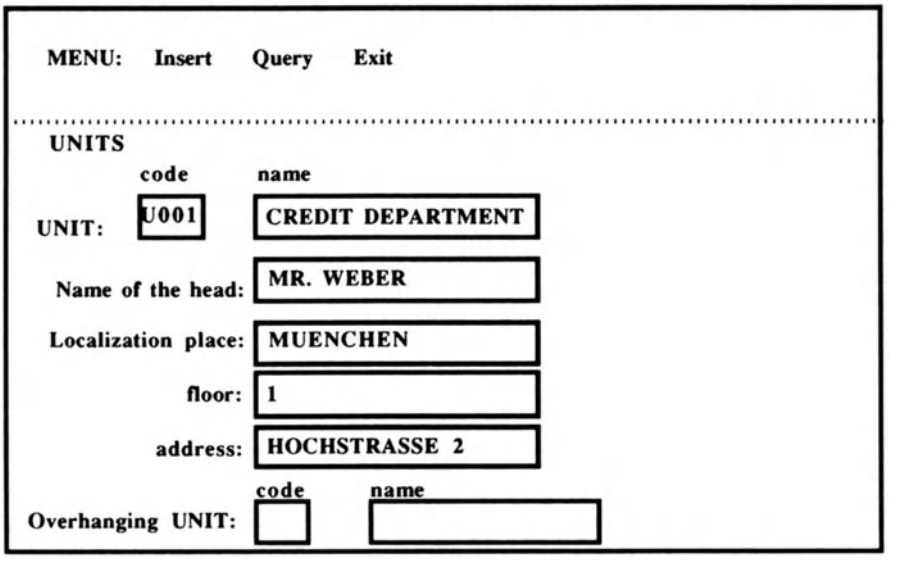


Figure 6: Input of departments involved

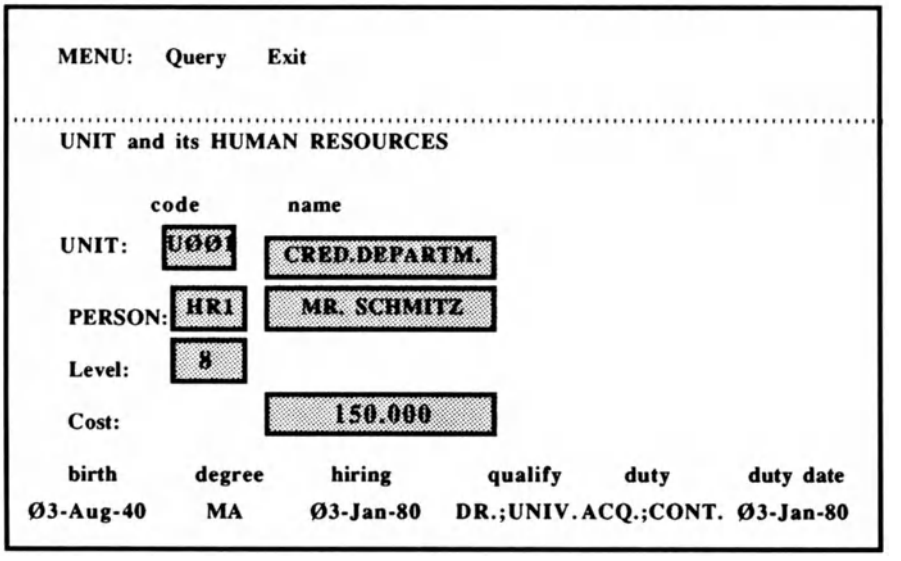


Figure 7: Professional details of human resources involved

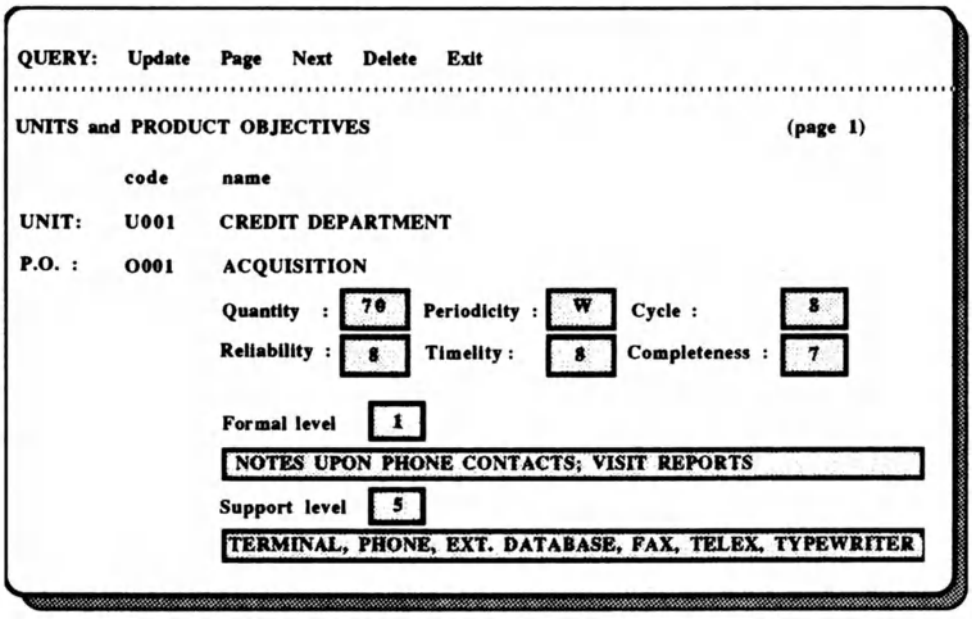


Figure 8: Product-Objective-Screen in TAM

filing, handwriting, typewriting and mail handling. These activities are executed with the help of instruments. During the exploration phase the instruments in general use and their characteristic features were also defined (Fig. 9).

As mentioned in Chapter 2. TAM provides a defined analysis scheme, which the analyst may use as long as it is suitable for his investigation field. In case of any further information going beyond that schema he may easily extend the generic models of the Office Data Dictionary with the help of TST (TODOS Structuring Tool).

Within the case study the TST was further enlarged by specifying new properties (new class of equipment) and domains on the one hand. On the other hand specific attributes for the office itself and and the persons, archives and equipment (current technology, incompatibility of technology) were added.

In the subsequent method tailoring phase new attributes and domains which completed the relationships of office and information categories were added to the ODD. The ODD proved to be a helpful support within the analysis phase itself. This involved data gathering guided by several instruments like the function analysis instrument, communication analysis instrument, information analysis instrument, and the user needs analysis instrument. This data were inserted into the ODD. The final outcome of this phase was the requirements specification. Those requirements also entered the ODD.

The application of the FAOR information analysis instrument was detailed data upon archive being a central archive and departmental and personal archives (Fig. 10).

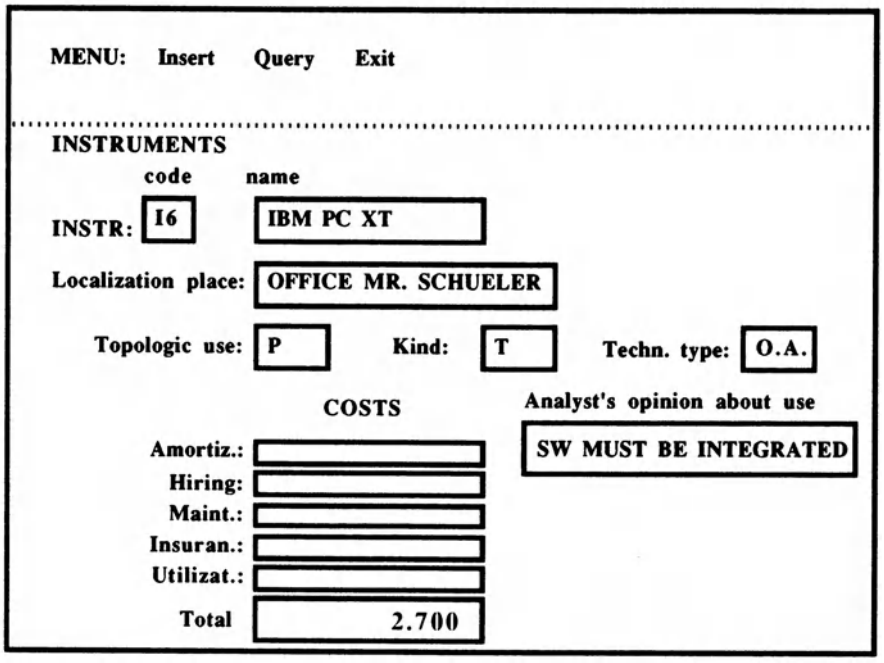


Figure 9: Instruments used in the organization

Archives: Structure and Access

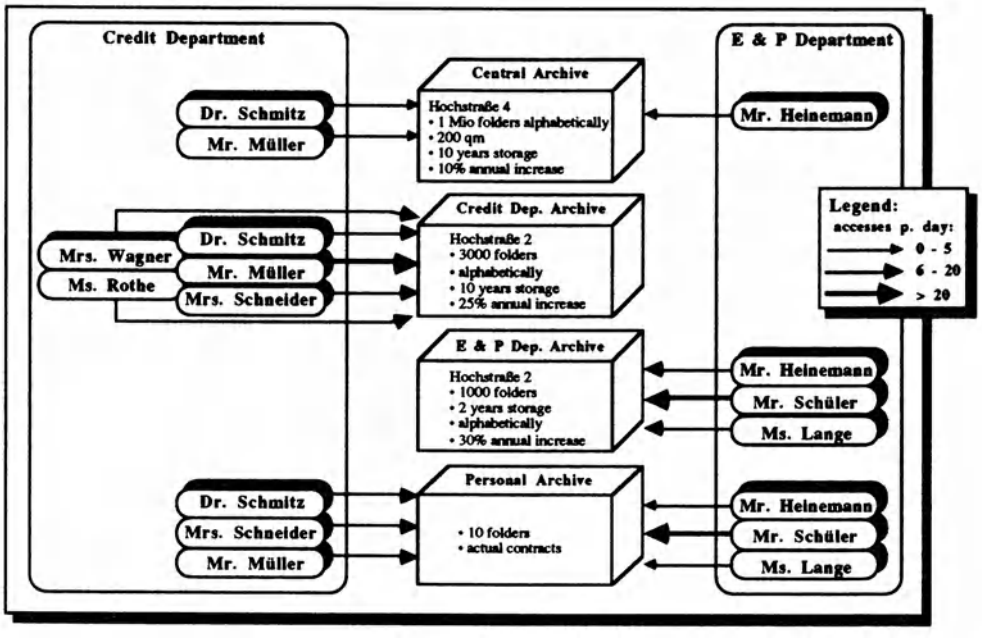


Figure 10: Types of archives

Personal archives are therefore those folders which are usually stored in either the credit departmental or the economics & press departmental archive and which can be taken out using borrowing slips. The credit departmental archive only contains data on established and potential clients whereas the E & P departmental archive also stores information on firms which as yet have no contact with the bank. Information identical to both archive comprises:

- Balance sheets of established and potential clients
- Hard copy versions of the external database CREDITREFORM
- Newspaper articles (original in E & P; copy in credit d. archive)

Therefore the credit departmental archive (Fig. 11) consists of folders in alphabetical order containing client's files classified according to date. Unsuccessful contracts are stored for 1 year after which they are deleted from the records. Successful contracts are kept in the credit departmental archive and accessed once a year by a clerical worker for solvency until they are repaid and thus terminated.

Folders belonging to the E & P departmental archive are also arranged alphabetically, each containing press and other information on various firms. The economics and press departmental archive has the largest annual increase of 30%.

The central archive contains only former folders of the economics and press departmental archive consisting of terminated contracts which are under a legal obligation to be stored for 10 years. Storage is generally in alphabetical order. Activities performed in the office were detailed and specified with flow charts (please refer to App. B) to demonstrate in detail the diverse office activities and human resources involved in producing the objectives within the lending process. They also show the use of documents and archives, data processing equipment and therefore give useful hints with respect to the future office information system. The type of documents used was also specified.

TAM and TST provided the possibility both of storing all features of the relevant archives and relating the archives to the persons involved so that full consistency with the above mentioned features was attained.

The FAOR functional analysis instrument (part of Analysis in FAOR Activity Framework) interprets the office as a functional system within the organization, that means in relation to the broader system and the task environment of the office (Fig. 12).

The analysis process was responsible for producing functional and

non-functional requirements for future performance of the system. Queries as to the features of product-objectives provided useful assistance in indicating functional requirements (Fig. 13): It could be easily observed and analyzed how actual deficiencies of information media were related to the business goals (critical success factors).

The TODOS Performance Model (TPM), (see Chapter 2, Section 2.2), has been specifically devised for gaining insight into the non-functional requirements of the new Office Information System. Its main use is in the evaluation phase of the FAOR Activity Framework. The TPM stores requirements for a number of user-oriented performance measures, which the future OIS has to meet. The idea of the TPM is to deliver design parameters actually during the course of the requirements analysis in order to gain both a swift user feedback and data for the configuration of the system during Architectural Design (see Chapter 5). Typical classes of requirements to be stored in the TPM are

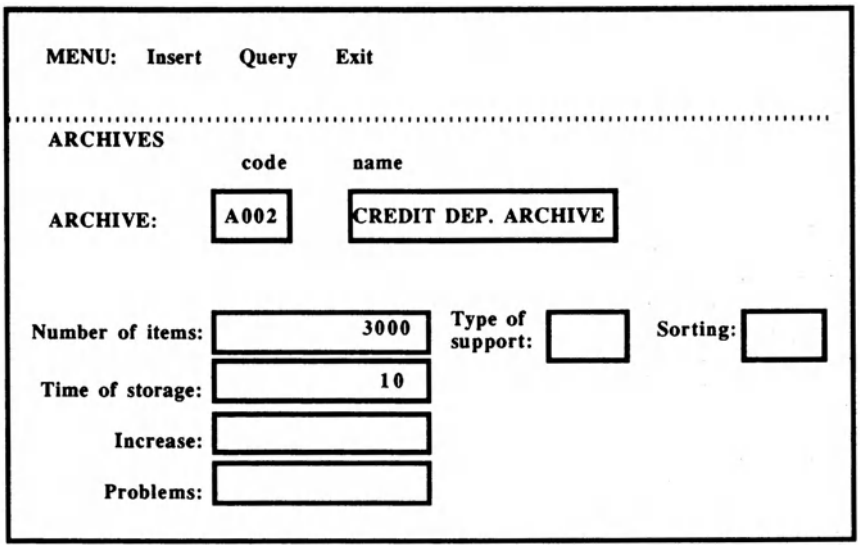


Figure 11: Insertion of archive features into ODD

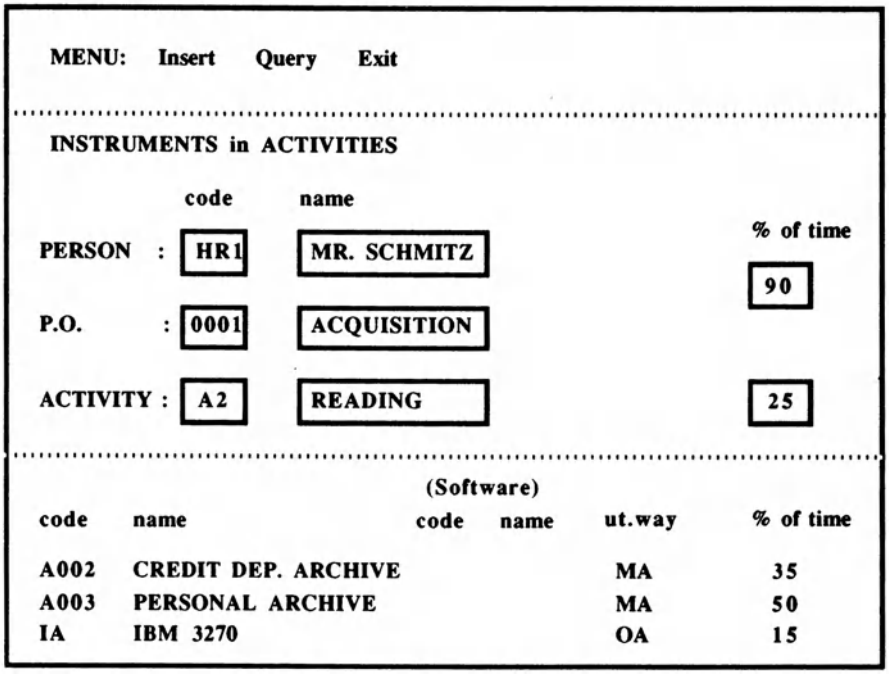


Figure 12: Functional analysis with TODOS

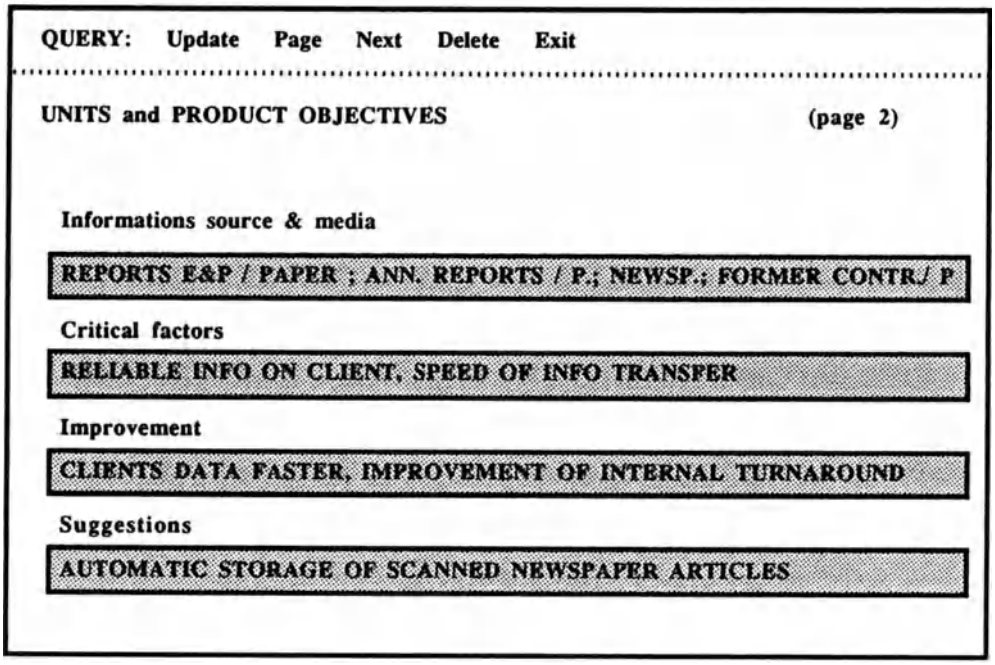


Figure 13: Functional requirements within TODOS

non-functional aspects like size, noise, display quality, color, ergonomy, help functions and response time constraints. By means of the TPM the analyst is able to derive various solutions for evaluating the impact of the future office information system including the organizational measures accompanying it by analyzing the interrelationships existing between the elements of the system (Fig. 14).

Following the application of the case study data, it was concluded that substantial insight through intensive use of the ODD had been obtained. The TAM/TSM integration had been verified. Requirements collection and analysis is also useful in initializing the design tasks in that it helps to identify constraints and guidelines via the TPM for the future organization. So the analyst can rapidly obtain a rich picture of a future situation enabling him to discuss both with the management of the organization investigated and the designers of the system. For this task diverse reports can be used.

Applying the ODD of WP1 within a real life example showed both the functionalities of the tool and the TODOS/FAOR intersection.

Interface with other TODOS Design Phases

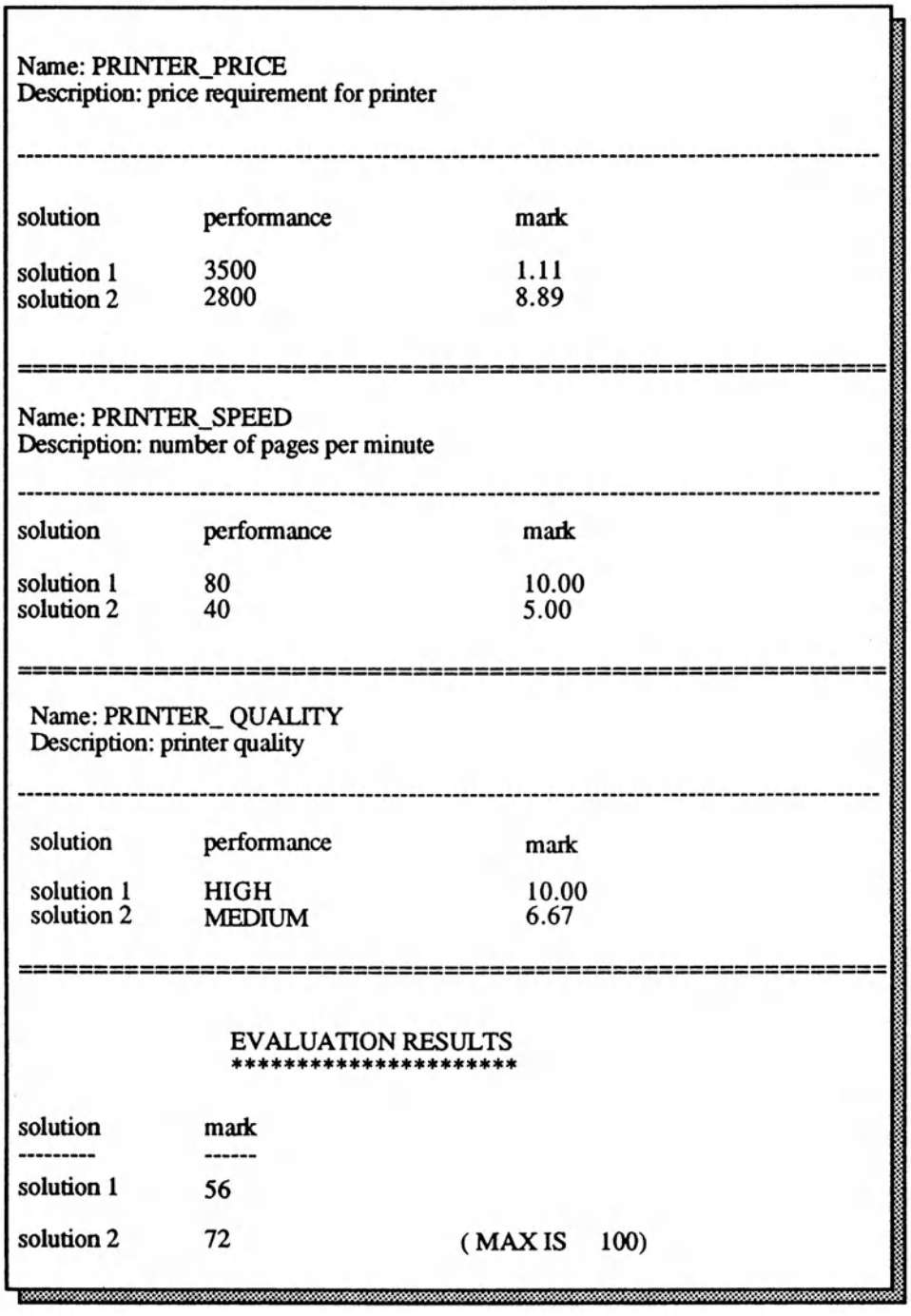
The interface to the Conceptual Design and Rapid Prototyping Tools was realized via the transmission of data concerning human resources, diversified by their name and code as well as the complete information on archives and documents. During the two design phases, the designer could gain easy access to information by querying the appropriate reports derived from the ODD (Fig. 15).

For Architectural Design the architectural features of the TPM were interesting: The architectural designers referred to the Office Data Dictionary in order to check which technical constraints were related to the proposed architectures. In parallel they gave their performance evaluation to analysts (test of printer speed for proposed servers) in order to check this against individual marks and weights given by responsible people during the analysis (analyst, manager etc.).

3.2 Conceptual design of the future Office Information System

The aim of the Conceptual Design in TODOS is to create a conceptual scheme of the future office information system. This conceptual scheme is a description of the elements in the office in terms of concepts of the TODOS Conceptual Model (TCM). The scheme should be complete, correct, and consistent on delivery without containing undefined elements. The principal purpose of the method for logical design is to structure the production of the formal specifications in a set of design steps. Partitioning the design task is obligatory when large projects are involved.

The C-TODOS tool has been developed to support office conceptual design. It provides the computer-based support to facilitate and improve the quality of office conceptual designers. In the case study, input into the conceptual design phase consists of the functional requirements collected and analyzed during the first phase in the functional design cycle. Such requirements were stored in the Office Data Dictionary (ODD) (see App. A). The portion of the ODD utilized during conceptual design contains information about the principal documents in the office, the goals of the office and the main activities required to attain these goals.



Name: PRINTER_PRICE
Description: price requirement for printer

solution	performance	mark
solution 1	3500	1.11
solution 2	2800	8.89

Name: PRINTER_SPEED
Description: number of pages per minute

solution	performance	mark
solution 1	80	10.00
solution 2	40	5.00

Name: PRINTER_ QUALITY
Description: printer quality

solution	performance	mark
solution 1	HIGH	10.00
solution 2	MEDIUM	6.67

EVALUATION RESULTS
****

solution	mark
solution 1	56
solution 2	72

(MAX IS 100)

Figure 14: Evaluation of non-functional requirements and solutions

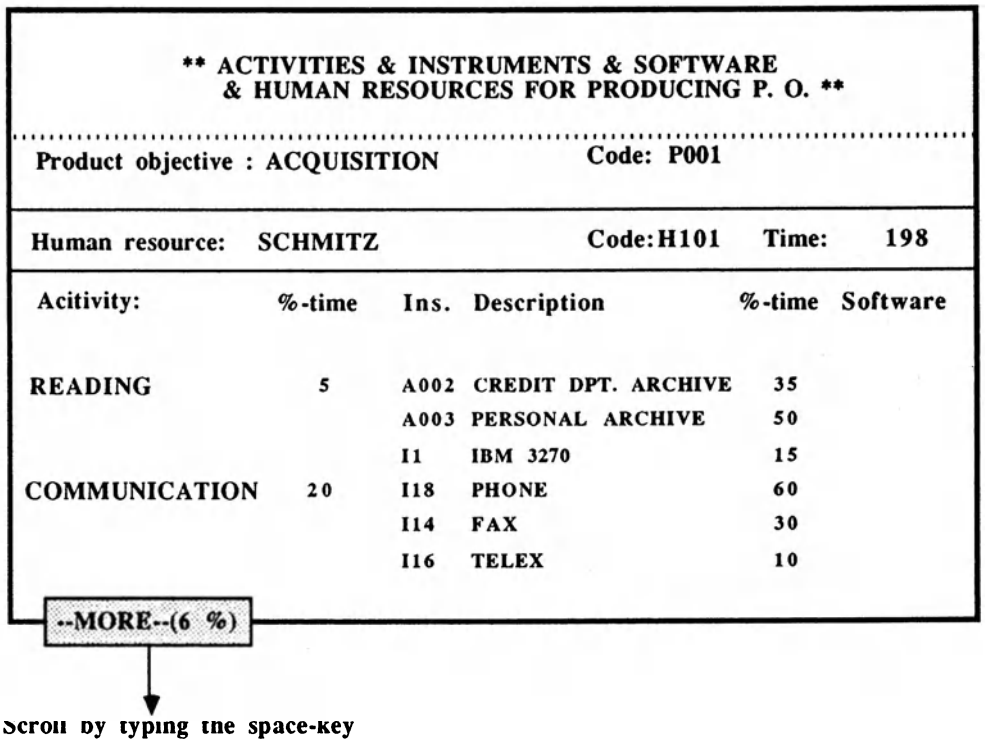


Figure 15: Example for a preparation of collected data

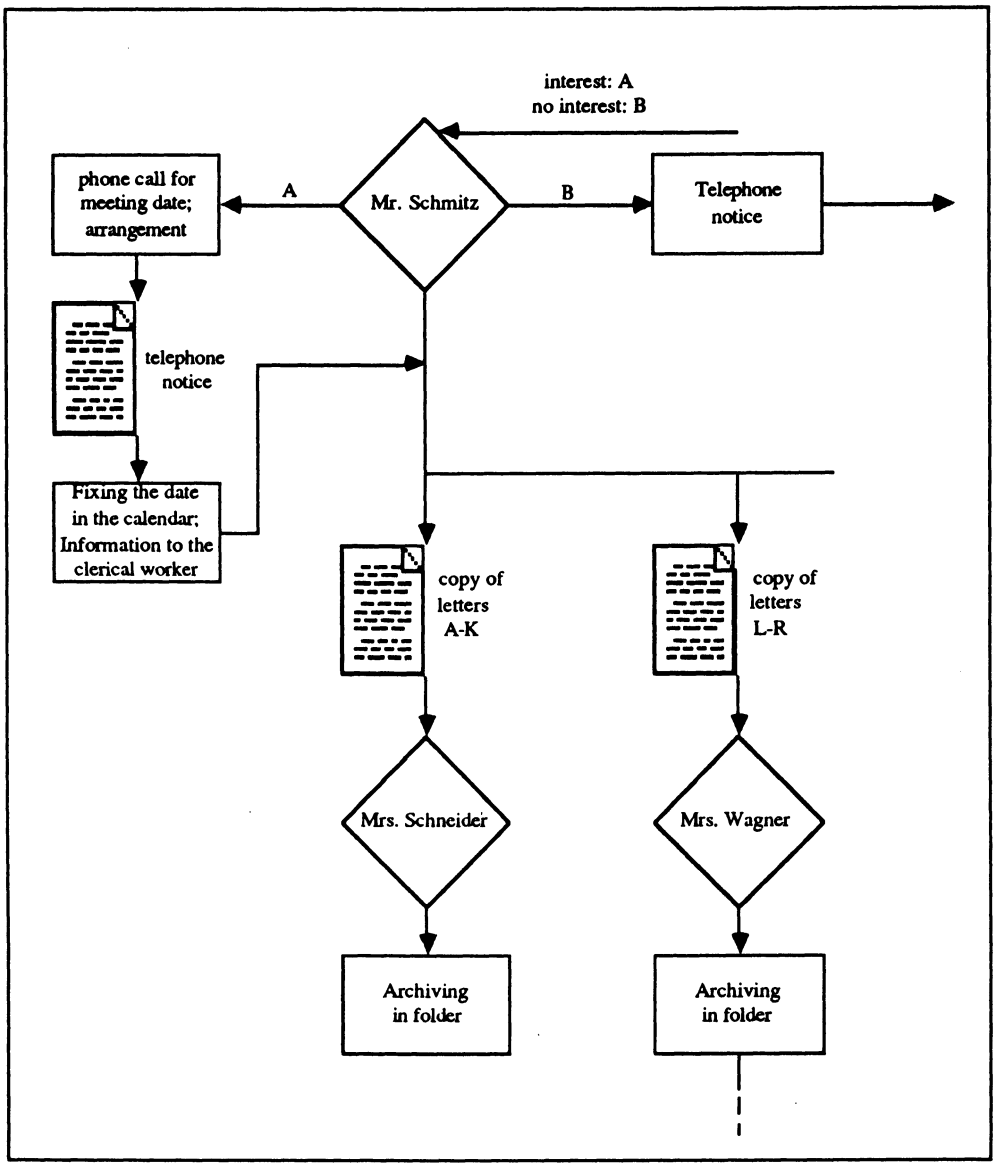


Figure 16: Examples of principal activities in the case study

Specification of the conceptual office scheme proceeds from weakly structured requirements, specified in the requirements collection and analysis phase and stored in the ODD, towards the formal definition of the conceptual scheme. The functional requirements in the ODD are expressed in terms of units, human resources, product-objectives and archives within the descriptive model:

- Organizational charts in the case study describe the departments in the bank which are involved in the lending process: Credit Department, Legal Department, and so on. Human resources contain the office workers' roles within the different units (e.g., "manager", clerical worker").

- Product-objectives contain a description of office goals. The main goal of the lending procedure is partitioned into four phases (such as "Acquisition", Offer, Approval and Administration. shown in Fig. 3): For each phase, the activities performed to achieve these objectives and the agent roles involved in them were described using a flow chart. Fig. 16 shows how actions in the office were represented in the ODD, in terms of office workers performing them, documents handled and decision points. Special attention was paid to the fact that the treatment of potential new clients doesn't differ at all from those who are in the stable clientele.

- The description of the archives' content was provided in the form of its documents with reference to the most important types, e.g. "contract" (Fig. 17). The aim of the logical design phase is to elaborate this description into a TCM scheme describing the functionalities of the future office information systems. The content of the ODD was elaborated via queries and report generation to obtain the TCM scheme and based on the following assumptions:

- The concepts of "documents" and "human resources" and "units" in the ODD are compatible with those of documents and agents in TCM.

- The "phase" concept in the ODD describes both a set of actions and its connection with other phases. Actions in the ODD can be mapped onto actions in TCM. Connections can be transformed in events, thus obtaining a first version of a TCM scheme (TODOS TR 1.1.2.7).

The conceptual schema for the case study contained 97 entities (45 static entities and 52 dynamic entities). In this chapter, an abridged example of the process of production of the TCM schema, based on the C-TODOS tool, is presented. Only a small part of the case application is shown while leaving out minor details of the process.

The case study data application began with the specification of the static part of the office scheme, i.e., the definition of documents. For instance, let us consider the "Contract" document the layout of which is presented in Fig. 17.

In C-TODOS specification can be entered using the graphical editor or in TSL text. If the graphical editor is used, the procedure is automatically transformed into TSL text (TSL generation). The graphical representation of the static entity "document" in TCM is shown in Fig. 18. The corresponding TSL text in Fig. 19.

Successively, the dynamic entity specifications are inserted. Let us consider contract preparation within the Offer and Approval phases of the lending process. The corresponding flow chart is shown in Fig. 20.

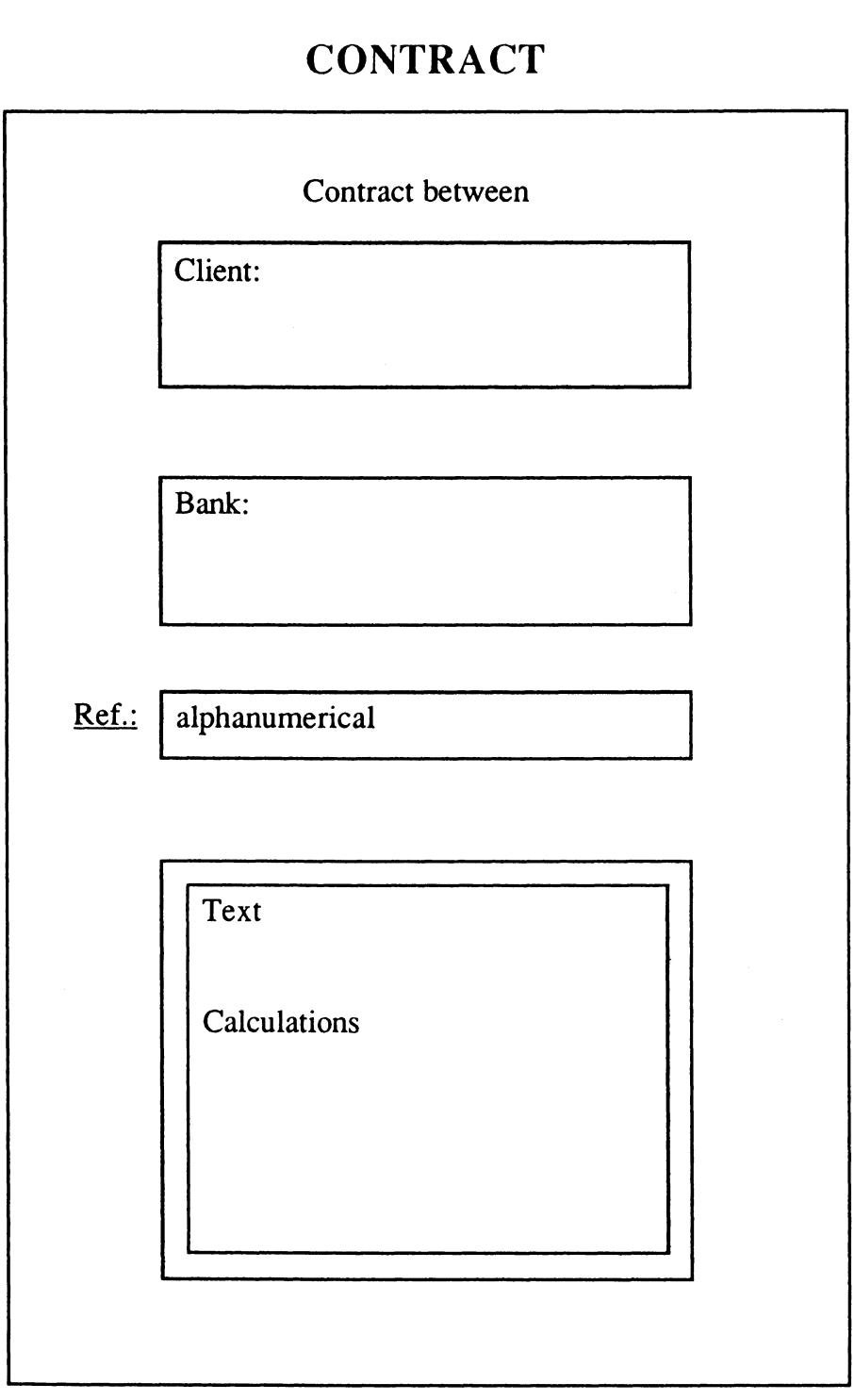


Figure 17: Static part of the office scheme

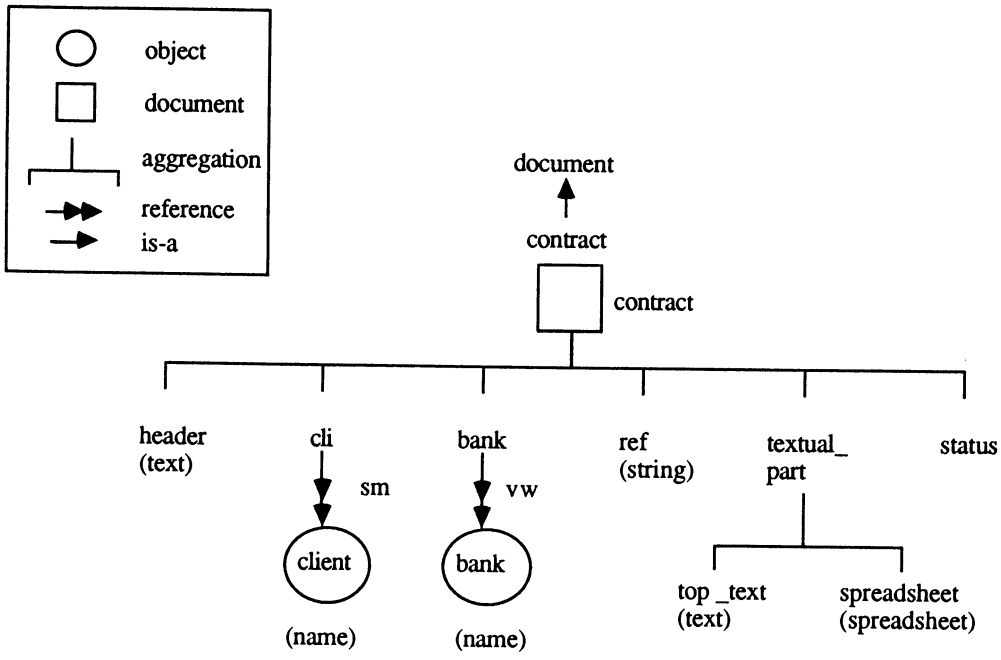


Figure 18: Graphical representation of the contract document

```
<contract> is-a document;
    { aggregation-of
        { header: text;
        cli : same-as client with { name };
        bank : view-of bank with { name };
        ref : string (10);
        textual_part : aggregation-of
            { top_text : text;
            spreadsheet : spreadsheet
        };
        status : ("new", "modified", "ready", "accepted");
        values : aggregation-of
            { header : {"Contract between"
                }
            }
        }
}
```

Figure 19: TSL text generation for document type "contract"

Once created, the contract is repeatedly modified by agents in the Credit Department. When prepared the contract is sent to the Legal Department for verification. Successive modifications may be requested. When approved by the Legal Department, the contract is sent to the client for acceptance. When accepted, the document listing all due payments for the client is prepared.

Fig. 21 shows the TCM schema derived in the conceptual design phase using the C-TODOS tool.

The TSL textual description of some of the dynamic elements in Fig. 21 is shown in Fig. 22.(Fig. 22 a to 22 d):

The "is-a hierarchy" graph showed the specialization chains of the basic office elements: documents, objects, messages, agents, events, actions, predicates, conditions and factors. This graph provided an overview of the generalization/specialization relationships among all defined elements.

During the design process, some of the entities stayed undefined. It was possible to ask for the list of undefined entities using the query module of C-TODOS. When an undefined entity is successively defined, the tool checks to ensure that the property existence rule is satisfied (for instance, after definition of the static entity "contract" as in Fig. 19-20, "bank" must have a property "name").

During the office conceptual scheme analysis phase, the following tasks were performed with the help of C-TODOS:

- check constraints: no violation should occur
- check accuracy rules: for instance,some actions may not have a definition of input or output parameters.

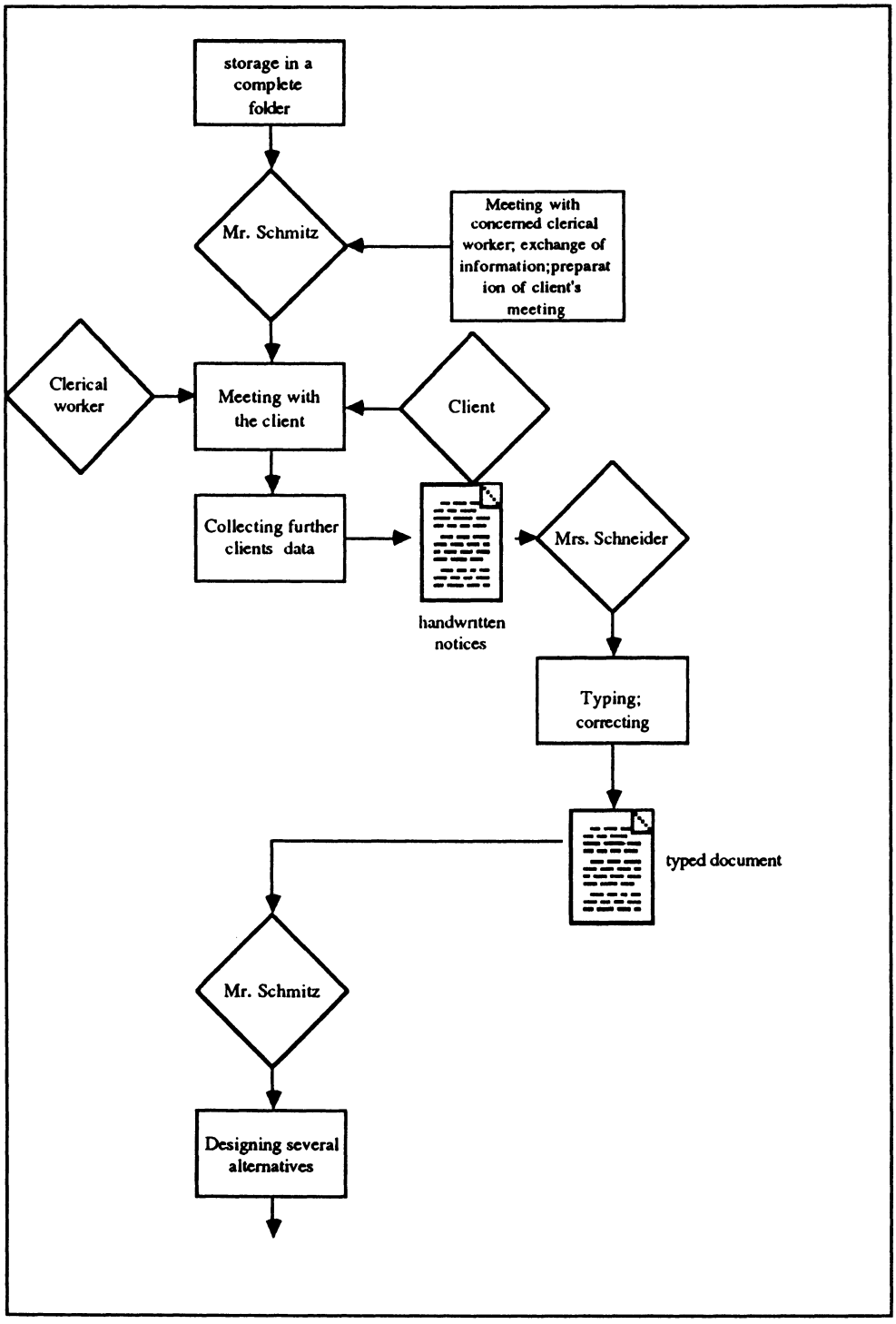


Figure 20: Contract preparation in the lending process (part 1)

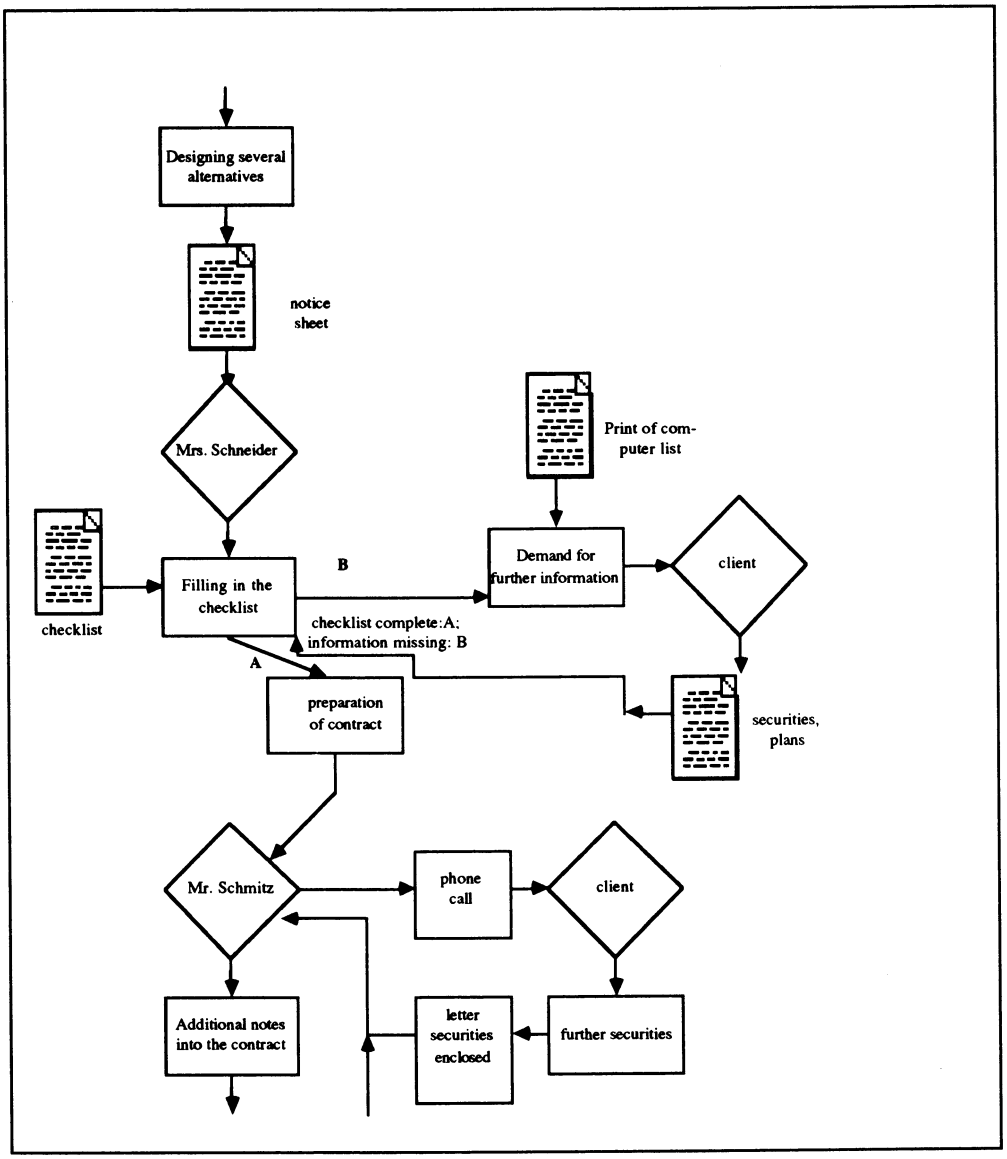


Figure 20: b) Contract preparation in the lending process (part 2)

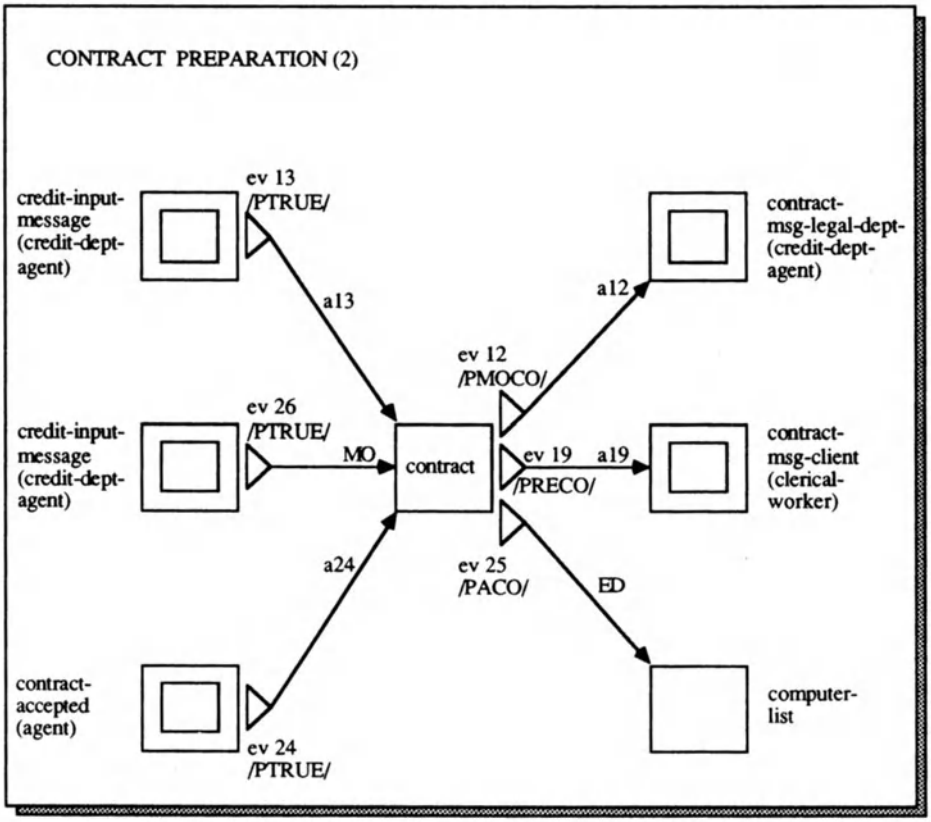


Figure 21: Dynamic part of the specification process

```
<ev19> is-a modification_event;
  { aggregation-of
    { values : aggregation-of
      { comments : "send contract to client for approval";
       ev-entity : contract;
       predicate : PRECO;
       triggers : {( a19) };
       follows : @ev26@;
      ev-agents : @acquisition_manager@
```

Figure 22: a) TSL definition of event 'ev19'

```
<PRECO> is-a predicate;
 { aggregation-of
   { values : aggregation-of
     { comments : "contract is ready to be sent to client";
       expression : "$co.status='ready'"
     };
    in : aggregation-of
     { co : ref-to contract
```

b) TSL definition of predicate 'PRECO'

```
<ev26> is-a arrival_event;
 { aggregation-of
   { values : aggregation-of
     { comments : "contract is ready";
      ev-entity : credit_input_message;
      predicate : PTRUE;
       triggers : {( MO ( $co,"$co.status:='ready'" )) };
      follows : @ev11@;
      ev-agents : @acquisition_manager@
     };
    var : aggregation-of
     { co : ref-to contract
     };
    properties : aggregation-of
     { empty : ref-to empty
```

c) TSL definition of event 'ev26'

```
<sendcontrtcocli> is-a action;
 { aggregation-of
   { values : aggregation-of
     { comments : "send contract to client";
       act-entity : contract_msg_client;
       steps : ( PR ( $co ); CR ( $cmcl ); TX ( $cmcl ) );
      act-agents : @SYSTEM@};
    in : aggregation-of
     { co : ref-to contract
     };
    out : aggregation-of
     { cmcl : ref-to contract_msg_client
```

d) TSL definition of 'sendcontrtocli' action

Figure 22: TSL definitions (cont'd)

The dynamic process of preparing the data for potential credit clients within the acquisition process has been modeled (Refer to App. C).

Interface with other design phases

The output of conceptual design consists of a conceptual description of the office system. Such a description is given by the TCM office scheme, describing the office system supporting office work to achieve the goals listed in the ODD. The TCM scheme models the functionalities of the system to be implemented. Document production is modeled and office procedures formally described; exceptions and anomalies in office work are also accommodated and supported. The output of conceptual design is simultaneously an input to the rapid prototyping phase and to the phase of specification of architecture requirements.

The production of an office prototype allows user's evaluation of the system before its final implementation. Modifications and suggestions are reported back to the functional requirement collection and analysis phase. The whole cycle can be iterated until the conceptual design produces a set of specifications that are complete, correct, and accepted by the user. The functional design cycle interacts with the non-functional design cycle The goal of the non-functional design cycle is to choose an architecture for the OIS. To achieve this result, two elements have to be considered: the non-functional requirements and the functional specifications, i.e., which types of documents and activities are used and performed in the office. Therefore, the conceptual office scheme forms a part of the input included in the office architecture design. The results of the architecture design phase, in turn, affects the functional design activities. For instance, cost or technical limitations could have an impact on the possibility of handling or transmitting multimedia documents.

Experiences with the design process for the case study

To produce the specifications shown in App. C, three iterations of the functional design cycle were needed. At the end of the first delivery session, a complete office scheme was prepared. After prototyping, some modelization errors were found. For instance, in the first version of the scheme both "modify_contract_msg" and "contract_ready_msg" were modeled as "credit_dept_msg". This was a design error, since the semantics of the two messages was different, and hence the messages triggered different activities. The error was identified in the prototyping phase. Another request was for renaming some activities and messages. The numbering style chosen in the beginning (i.e. a1, a2, and so on for activities) had to be modified, as the significance of entity names was important in the automatic generated prototype.

Much help could be gained from the tool during the task of specifying office requirements. The tool supported documentation production, and provided features to ensure that the specifications were consistent. In particular, it was possible to query the specifications for examining them, and to retrieve information about undefined entities. Moreover, most consistency checks were automatically performed by the tool. In the described process, much was left to the designer in terms of what to automate and how.

3.3 Rapid prototyping

This section describes the case study scenario in the rapid prototyping phase. The design goals of a prototype are quite different from those of the final product.

- A prototype is always constructed to illustrate the feasibility of new ideas or design. Unlike the final product, prototype design and construction often begin with incomplete specifications.
- Prototyping should create a quick software package representing an office system application considering man-machine interaction, with layout and masks-generation, and the functional flow, with respect to one single working place at a time. Layouts for the static entities, especially of documents and messages, provide a visual description of the system, as well as giving the possibility to change views of layouts under space optimizing aspects. Each user should have the possibility to configure his user interface at the beginning of his work with the prototyping tool.

Input to the prototyping phase was made up on one hand of the collections of functional requirements resulting form the analysis of office work by the Requirements Collection and Analysis Phase The most important information arising from these analyses encompassed the description of real instances. Further information were layout examples for documents (Fig. 23).

The functional specifications defined in the conceptual design phase were also taken into consideration. The TSL definitions were used by the prototyping tool. The prototyping support tool is started by the process "make new prototype". The case study prototype was named "credit_department". The process of transferring the TSL definitions into an executable prototype knowledge base was composed of three phases:

- *Creating units*: For every entity defined in the TCM conceptual model, a KEE unit was created, and the TSL description was stored into a special slot of this unit.
- *Parsing*: For every unit, the text (in that special slot) was translated into a list structure, suitable for the third phase.
- Making the knowledge base *executable*: by providing the necessary information for a scheduler module which controls the execution flow of the (office) prototype.

The output of the prototyping phase was twofold: Creation of documents, object and message layouts on the one hand, implementation of office prototype on the other hand. The functionalities of the latter had to be tested by office workers.

- The **primitive editor** was used used to describe a manual process, which was interpreted for the prototyping phase as an automatic process.
- During the **basic instantiation** a description of the "credit_department" was inserted into the "credit_department" knowledge base. The instances were office workers, who should be allowed to create their own layouts. For example, the agents Schneider, Schmitz, Mueller, Lange and Rothe were instantiated. They

TELEPHONE NOTICE

Date: _____

Call from:

Name _____

Company _____

Telephone _____

Subject:

_____

_____

_____

_____

Recall: ☐ yes

☐ no

Notice:

_____

_____

_____

_____

Figure 23: Documents' layout

were placed in their respective classes. It was decided that an instance was necessary for the "credit-prospects" entity. The designer first created a layout for this entity and than created an instance to that entity via this basic instantiation component.

- For the **layout creation** the interface generator was used. It is an important part of the prototyping tool as it allows to a certain extent each user to tailor his own interface to the office prototype. The information about this interface is stored in a separate knowledge base which is specific to the user and to the prototype, and is loaded automatically when the user logs into the prototype. The layout creation for a certain class is in the first phase completely guided by the system, which offers in the form of menus all the properties necessary to create windows in the layout. The designer/user clicks through the menus until there are more windows to create. He has then in a second phase the chance of modifying the layout, e.g., by changing/removing titles and borders of windows, moving windows etc. Moreover, he can at any time modify existing layouts using this interface generator. Following our own experience, it should not be difficult for the designer and/or user to create layouts for the static entities and messages concerning the user.

- For the **prototype execution** the TSL grammar might be extended to allow completely automatic transformation. TSL in the state as it is now makes it possible to develop rather quickly a TCM for a given office. Fixing the details is then left to the prototyping phase, where one can experiment with different transactions. The price for this advantages is that the semantic equivalence between TCM and prototype cannot be guaranteed; another design might come up with a different translation for a primitive comment. As a consequence, the TCM alone is not enough as a specification of the functional requirements to the end system.

  Running the prototype mere general experience were made: The implicit parameter passing concept of TSL works quite fine, if one always follows the course of events as defined by the "follows" property. But this means that if one office worker has finished a task and produced some output that is needed by another worker, this other worker must first perform his task, using the first worker's output, before the first one can go on working. Fortunately, TSL also offers explicit parameter passing. So one can for this first version use implicit parameter passing (to get the first prototype very quickly) and then refine the TCM to create a more realistic prototypes.

- For the prototype execution one has to identify oneself by a login procedure; the main purpose of this is to tell the system which messages this user can send and receive. After logging in, a menu pops up, presenting all the message types the user may send, and in addition two options: "view arrived messages" and - of course - "exit". Normally, one chooses a message to send to the system. This action, then, causes the system to look for the consequences of this message and to trigger them; the system now remains active as long as its actions possess consequences. When it stops, the original menu pops up again. The prototype "credit department" was started by logging in Mrs. Schneider (an instance of "clerical_worker" class). She selected the "telephone_notice" message type. The

prototype hereafter creates an instance of this class (which is documented in the control window on the right side) and displays Mrs. Schneider's layout for "telephone_notice" (Fig. 28), allowing her to edit the contents. After she has finished editing, she points on the message frame to send the message to the system. The system now recognizes event "ev1" and triggers its consequences (for event names refer to dynamic graphs presented in App. C). Testing provided useful information on the user interface by prototyping the future system components. The output of the prototyping phase contained the creation of documents, object and message layouts on the one hand, implementation of office prototype on the other hand. The system works fine if the user behaves meaningfully. The implicit parameter passing concept of TSL works quite fine, if one always follows the course of events as defined by the "follows" property.

An instance of a client object was created and displayed, so that Mrs. Schneider could edit it. Finally the status property of the client instance was set to "interested". This implied the identification of the "ev4". "ev4" created a meeting instance and set its status to "new". A reference was established from this meeting instance to the client instance. The next event whose predicate is true (status is "new") was "ev5". Therefore an "arrange_meeting" message to "clerical_worker" was constructed, provided with a reference to meeting, and was transmitted, which meant in this case displayed. In parallel the "credit_prospects" instance, which had previously been built during "basic instantiation", is retrieved and its contents copied into the meeting object. Now Mrs. Schneider logs out via choosing the "exit" of the main menu.

To show *interconnections between agents*, the "acquisition-manager" Mr. Schmitz logged in. He chose the "start_meeting_msg", which was connected to the arrival event "ev7". So the meeting object got the status "start", which makes "ev9" occur. The activity "prmeetdoc" printed the "doc" slot of meeting and produced a "cli_doc_msg" with a reference to the meeting and transmitted it into the message-KB, since Mr. Schmitz did not belong to the receiver class ("clerical_worker") of "cli_doc_msg". Mr. Schmitz told the system about the end of the meeting by sending an "end_meeting_msg", which caused the status of meeting to become "end". This again pushed "ev10", triggering an action that set the status of client to "met" and allowed Mr. Schmitz to update client's data. At this point, Mr. Schmitz interrupted his work and Mrs. Schneider logged in to view the arrival messages. She found (stored in the message-KB common to all agents) instances of "cli_doc_msg" created during the activity of Mr. Schmitz. A new log in again as Mr. Schmitz was done in order to arrive at the contract preparation. Mr. Schmitz chose from his menu the class "credit_approval_form". Mr. Schmitz filled in his and Mrs. Schneider's name and gave the client a credit_status of "excellent". The consequence of this was that a reference was made from the client object to the newly created instance of "credit_approval_form", and the status of client became "approved", so that the predicate of "ev11" is true. "ev11" triggered the creation of a new "contract" document. Subsequently this document could be edited. The client name property (of the contract) was filled automatically with the client's name. Finally the contract was awarded the status "new". At this point, Mr. Schmitz completed the contract preparation (by sending some more messages) or could leave this task to another credit department agent.

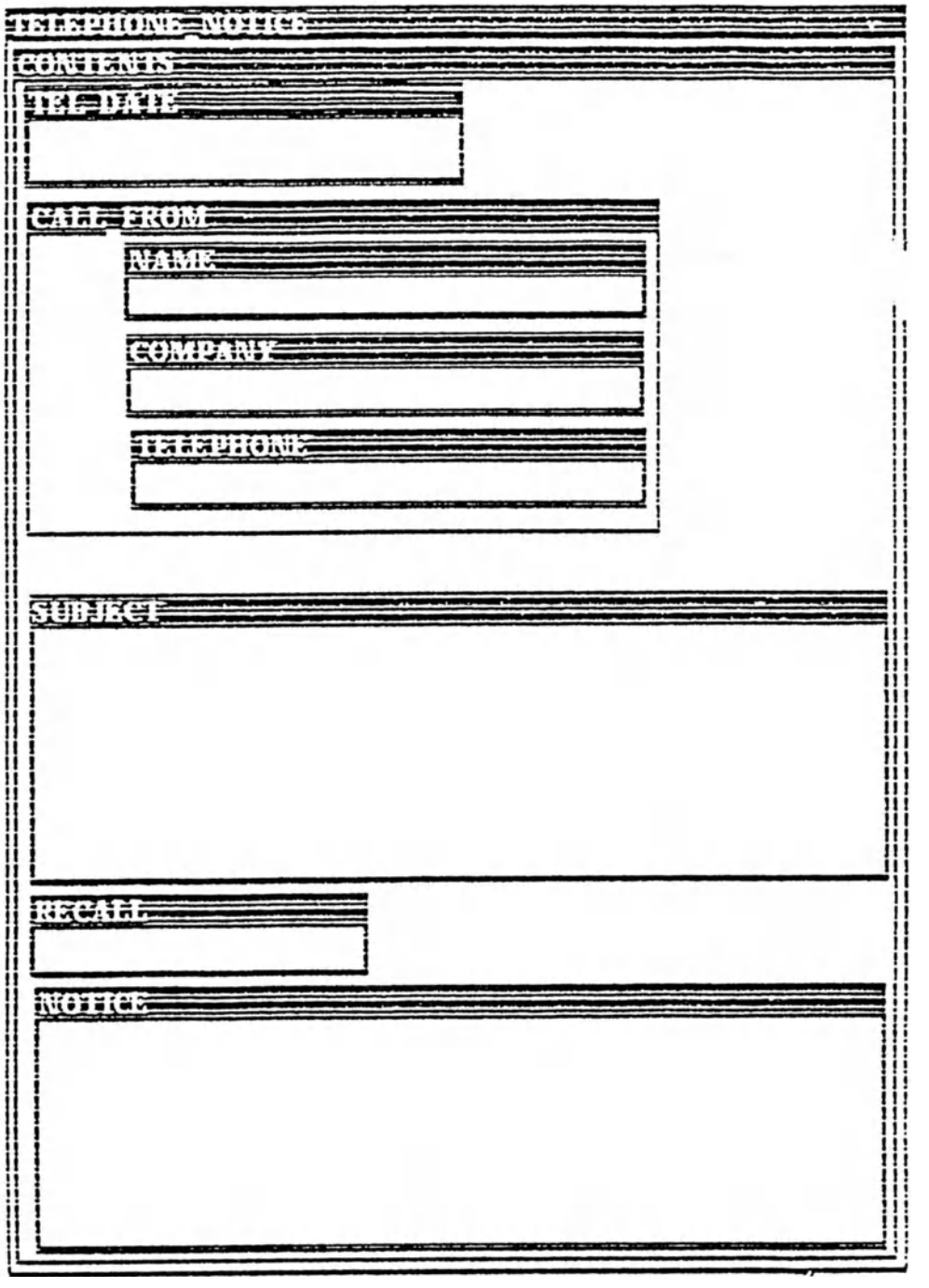


Figure 24: Telephone notice created by a clerical worker

Therefore messages sent to the system triggered activities which then led to the occurrence of more events triggering more actions, until the system came to a halt.

Interface with other TODOS design phases

A feed back to requirements collection and analysis should be the participation of the analysis team in the prototype test run. There the analysts can see then realization of the process he described in the ODD (later modeled in the TCM). The work with the prototyping support tool gives the designer and office agent a comfortable environment to build the man-machine interface to an office prototype, based upon the conceptual of an office system. The interface to conceptual modeling consists on an experience report coming from the rapid prototyping phase, giving the results and difficulties back to the logical design phase. Further comments are derived from the analysis team evaluation of the prototype, following the functional design cycle.

3.4 Architecture Design

While the design phases presented above are congruent with any traditional software development methodology, the architecture design phase is a specific TODOS component with respect to office information system design. The aim of the architecture modeling phase is the identification of an architecture that realizes the office information system being designed. Architecture comprises any set of interconnected hardware components and available software packages. Thus rather than implementing the office information system by developing software, TODOS focuses on the use of existing software packages and emphasizes the design of the architecture that will support the information system.

Therefore Architecture Design is subdivided into two steps (see Chapter 5):

- In the first step, which is the architecture generation step a number of alternative architectures, suitable for realizing the office information system, is identified (see Ch. 5, Part I and II).
- In the second step, the most appropriate architecture, among those identified in the first stage, is selected. To this end, each candidate architecture is transformed into an equivalent queuing network model, on which transactions simulating the office activities are run. The performance of the architecture is evaluated in this way, so that the most appropriate architecture can be selected on the basis of a cost-benefit tradeoff (see Ch. 5, Part III).

The input to the architecture generation is quantitative and qualitative information about the office activities. The qualitative information is provided in form of a conceptual schema describing the data objects manipulated by the office activities and the functions abstracting such activities.

The quantitative information is provided by the requirements collection analysis phase by integrating the conceptual schema with data such as the frequency of activities and size of the involved data. Architectures are generated by the interaction of the office system designers with the architecture generation tool, which assists the designer in the specification of architectures satisfying the requirements given as input. In the case study constraints on the technical layout (due to the present situation) were specified in the requirements collection and analysis phase (Fig. 25).

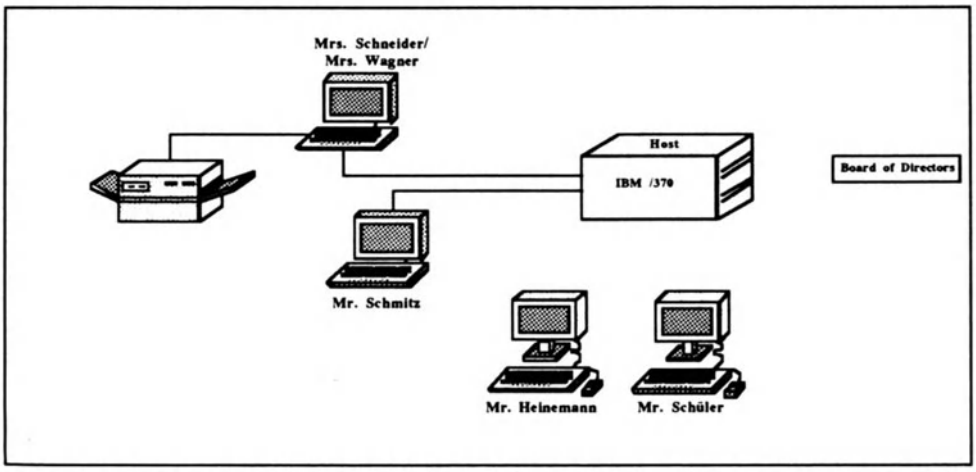


Figure 25: Requirements for the architecture specification

** ACTIVITIES & INSTRUMENTS & SOFTWARE
& HUMAN RESOURCES FOR PRODUCING P. O. **

Product objective : ACQUISITION Code: P001

Human resource: SCHMITZ Code: H101 Time:

Ac tivity:	%-time	Ins.	Description	%-time	Software
EXTERNAL CONTACT	50				
INTERNAL MEETING	10				
READING	5	A002	CREDIT DPT. ARCHIVE	35	
		A003	PERSONAL ARCHIVE	50	
		I1	IBM 3270	15	
COMMUNICATION	20	I18	PHONE	60	
		I14	FAX	30	
		I16	TELEX	10	
WRITING	5	I001	IBM 3270	100	WORD STAR
SEARCHING	5	A007	CREDIT DPT. ARCHIVE	40	
		A010	SCHMITZ'S ARCHIVES	40	
		I001	IBM 3270	20	
MAIL HANDLING	5	I009	FAX	60	
		I010	TELEX	40	

Human resource: SCHNEIDER Code: H102 Time:

Acitivity:	%-time	Ins.	Description	%-time	Software

Figure 26: Workload characterization in the ODD

Sharing information and resources (storage devices, printers, scanners, etc.) was a basic requirement in the future office information system of the case study. This requirement was met by establishing a communication network restricted to Local Area Networks (LANs). Consequently, the resources were distributed over the LAN, each point-to-point connected with a host machine.

Another source of input for the performance evaluation in the Office Data Dictionary was the TODOS Performance Model (TPM), where requirements for a number of user-oriented performance measures, which have to be met by the OIS under design, were stored. TODOS Analysis Model (TAM) as well was used as a source for information as it contained a characterization of the office workload, like the access of each person to the various archives. An appropriate characterization of the workload imposed on the OIS was very important, since the performance of any system could only be determined with respect to the work being supported (Fig. 26).

The following assumptions were made:

- Workstations and PCs with no or limited storage capacity,
- file servers providing archiving facilities with large storage capacity like disks, drums and tape drives,
- print servers that enable the printing of documents on connected (laser) printers,
- database servers that enable access to shared data bases,
- mail servers that enable distribution of documents among office workers,
- local area networks (LAN) for communication between the various machines of an office system,
- gateway servers that enable communication with other networks and possibly links to a central computer (mainframe).

The personal workstations under consideration are SUN (-like) workstations and IBM (-compatible) PCs, while the servers are based on (SUN-) workstation technology. The LANs are based on Ethernet.

The appropriate level of workload characterization for the performance evaluation was in terms of:

- file handling,
- document printing,
- document mailing, etc.

The case study dimension was, for demonstration purposes, adjusted in order to derive OIS architectures. If only the current situation had been taken into consideration with probably only 8 personal workstations (or PC) connected to a LAN, performance evaluation should have hardly been worthwhile.

As far as the workload on the OIS is concerned, "meaningful" assumptions were made whenever necessary, sometimes based on (empirical) performance studies found in literature.

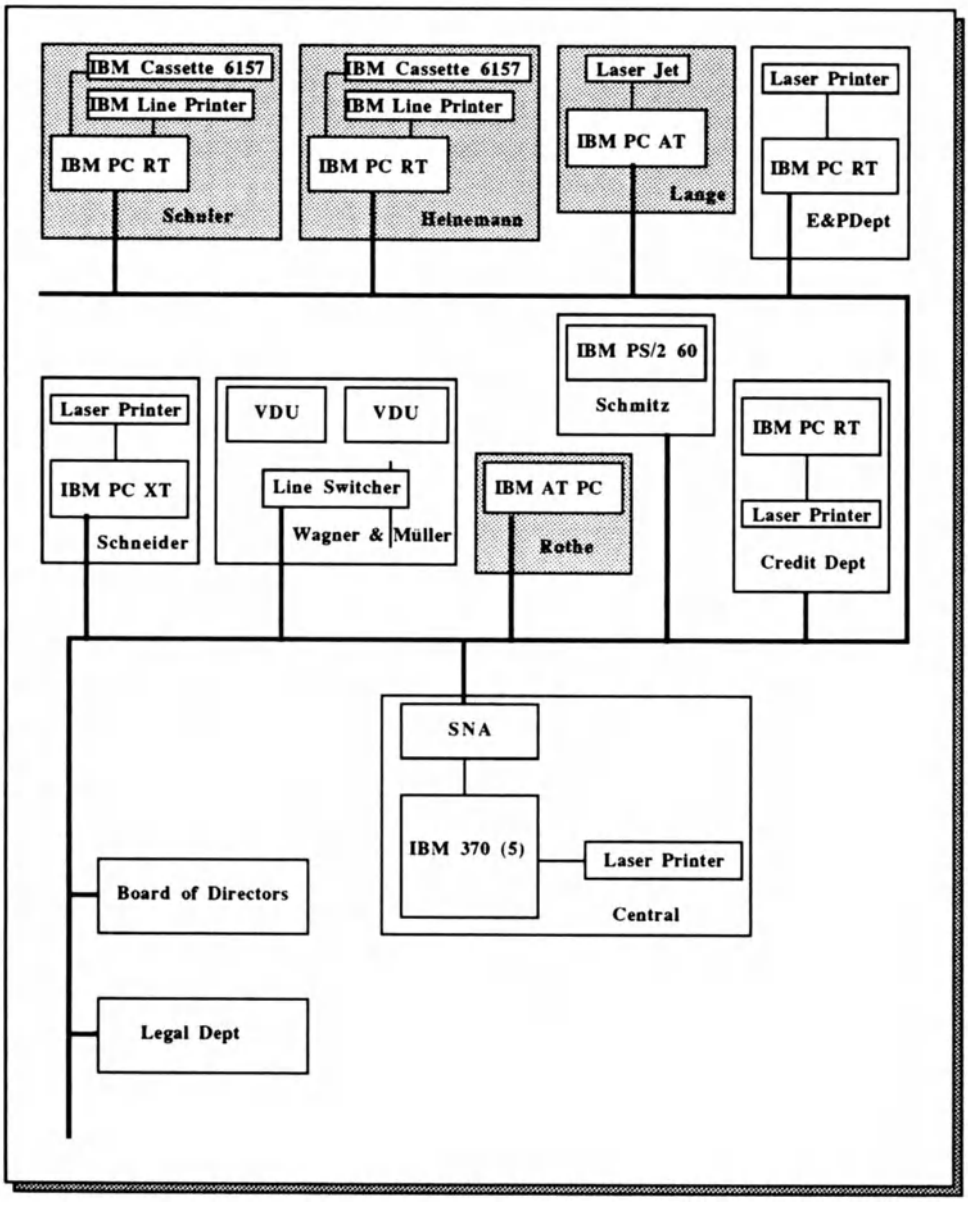


Figure 27: Architecture Solution

Experience with the design process for the case study

Architecture Design applied to the case study was divided into 2 different steps: the architecture specification and the architecture performance evaluation.

The delivered solution took into consideration the imposed constraints (Fig. 27). The implementation language PSN (please refer in detail to Chapter 5, I+II) provided the mechanism for representing and manipulating the different objects and their relationships and implementing the Architecture Specification System (ASPES). A graphical interface for ASPES was implemented as a C program running under UNIX.

Alternative computer system architectures were then derived from the requirements collection analysis and the conceptual model through a stepwise refinement process. Successive transformations of the office description within the case study were derived from the ODD (TST) and the database produced by the logical design phase(TQL) of TODOS through interactions with the designers.

The distribution requirements, the reusability of already existing hardware, suggestions for the hardware as well as requirements on the integration of the already existing software were all taken into account. The distribution of software for the above mentioned architecture mainly corresponded to the requirements of the case study.

Mr. Schüler:	Graphics Spreadsheet Modula II™ DB III™ PC-Text 4™ PC-Office™
Mr. Heinemann:	Graphics Spreadsheet Modula II™ DB III™ PC-Text 4™ PC-Office™
Ms. Lange:	PC-Office™ PC-Text 4™
Mr. Schmitz:	PC-Text 4™ PC-Office™ DB III™
Ms. Rothe:	PC-Text 4™ PC-Office™
Mrs. Schneider:	Graphics PC-Text 4™ PC-Office™
Board of Directors:	PC-Text 4™ PC-Office™
IBM 370:	Graphics DB III™ DISOSS/370™ Office/370™ Text/370™

The second step of Architecture Design concerned the performance evaluation of the proposed architecture.

The SNA communication architecture has a similar hierarchical structure to the OSI model, and the IBM mainframe in the case study definition was viewed as one big server to be replaced by several smaller servers. The file access performance of diskless workstations accessing a remote file server over a LAN was also modeled.

The reviewed OIS architectures comprise a number of client workstations (from which remote file accesses are initiated) and one or more file server(s) which handle these requests. Both clients and server(s) are connected to a LAN and rely for communication on the transport service provided via the LAN.

Since a human user of the system does not access files continuously, but pauses after a period of time, no constant stream of data is assumed. The length of the human pause is determined on the basis of further measurements, which show that on average 4K per second is transferred. The corresponding pause is included in the model as an interactive think time with a length of 0.85 seconds.

It was started with a first model which is assumed as the baseline case. Afterwards, several design alternatives were applied to the model in order to improve the performance of the system. The case was investigated on the basis of an increasing number of client workstations attached to the LAN. The number of workstations was increased from 10 to 50, and in each of the five corresponding simulations the response time per 4K was transferred in order to estimate the utilizations of the hardware resources at the server site. The resources at each client site are not relevant to performance degradations since they were not overloaded. The following choices were made for the baseline system:

Single file server
Same CPU for clients and file server with a speed of 1 Mips
Priority scheduling discipline with pre-emption distance 2 for all CPUs
LAN's packet size of 1 Kbytes
Transfer rate of 10 Mbps
Transport layer window of 8 for the transport service
Connection-oriented data link layer with data link window 4
No acknowledgement accumulation
Disk subsystem as the client uses a block size of 4 Kbytes
Mean access time is 20 ms

The results of the five simulations were presented graphically (Fig. 28).

It was concluded that network contention would be unlikely to produce a bottleneck, since even with 50 workstations attached to the system, the network utilization is only

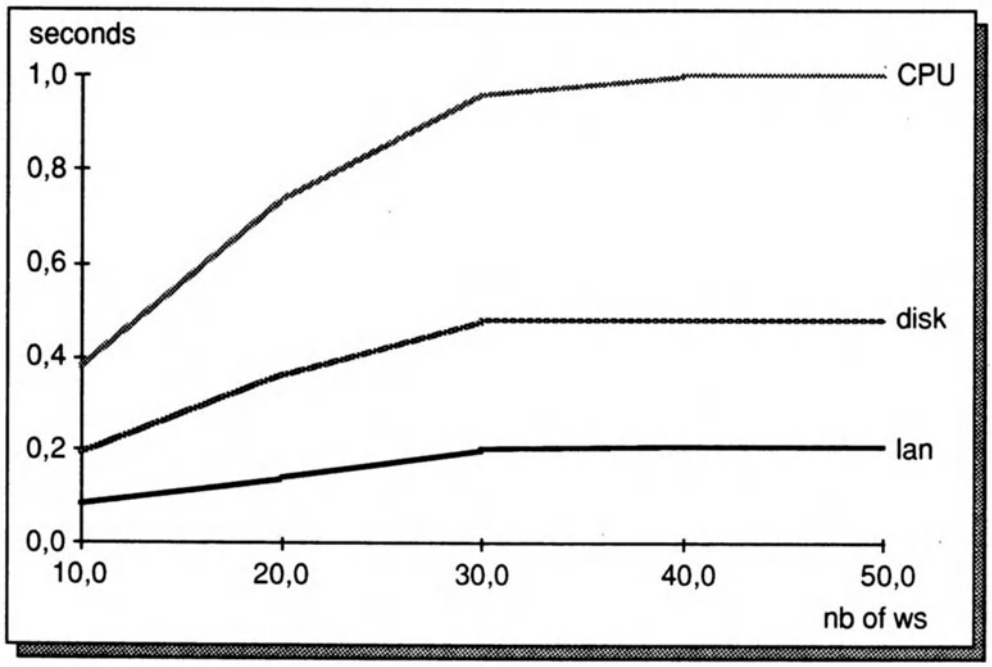


Figure 28: Utilization of the hardware resources

25 %. The measures of utilizations of the shared hardware resources and the confidence intervals for these measures were also presented in a table (Fig. 29).

Obviously the resources were not heavily utilized when only a few workstations were attached to the network. This implied that performance could be improved by reducing the service demand of any device. However, at high loads, e.g. more than 30 workstations, the performance was totally governed by the most heavily utilized device and, as expected, the system's bottleneck in the file service modeled here was the server cpu. Already with 30 workstations, the server cpu was close to being fully utilized, causing the performance to degrade rapidly in accordance with the increasing number of workstations. Thus performance could only be improved by a service demand reduction at the server cpu and any other modification was expected to have little or no result.

In order to show the flexibility of the performance modeling techniques employed, four design alternatives were modeled, all of which may be derived from the baseline case by simple parameter changes. The four cases were:

- increase the disk block size to 8Kbytes
- increase the LAN-packet size to 4Kbytes
- double the speed of the server cpu to 2 Mips

UTILISATIONS OF SERVER HARDWARE RESOURCES			
# ws	cpu	disk	network
10	0.3729	0.1879	0.8863e-01
+/-	0.1201e-01	0.8802e-02	0.1821e-02
20	0.7348	0.3682	0.1758
+/-	0.1993e-01	0.1065e-02	0.4402e-02
30	0.9611	0.4826	0.2304
+/-	0.9196e-02	0.7639e-02	0.1942e-02
40	0.9983	0.4929	0.2385
+/-	0.1071e-02	0.6031e-02	0.1832e-02
50	0.9999	0.5044	0.2412
+/-	0.2489e-03	0.1181e-01	0.2045e-02

Figure 29: Utilization of server hardware resources

- add a second file server to the system

From these four cases, the response times of the file access requests were obtained and presented (Fig. 30).

The alternatives were evaluated separately.

1. The increase of the disk block size from 4 to 8 Kbytes had two effects. First, the effective disk access time was reduced, since the frequency of access requests and disk acknowledgments diminished: both a request and an acknowledgment now covered 8Kb instead of 4Kb.

2. By increasing the size of the packets that were transmitted over the medium from 1 to 4 Kbytes, the cpu service demands were reduced since the number of packets to be transmitted over the medium diminished: transmitting 4K of data could be performed with one single data packet, rather than with four. Fig. 42 shows that this systems's performance is comparable to the performance of the system with 8K disk blocks.

   The common factor in these first two cases was that the amount of work to be executed is amortized over larger volumes of data. The conclusion could be drawn that these volumes, the disk block size and the LAN packet size, should be made

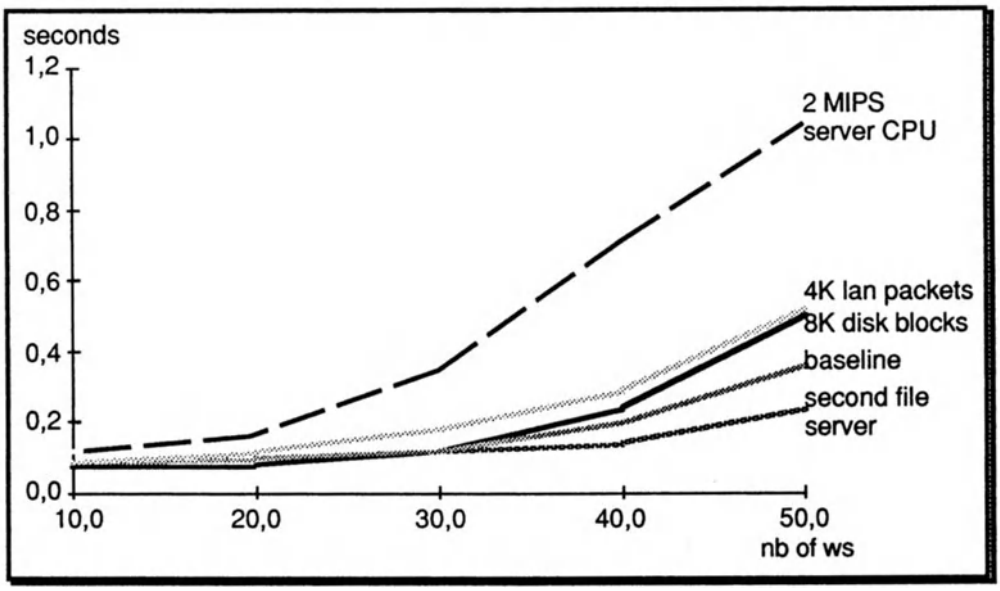


Figure 30: Response times of the four alternatives

as large as possible to reduce the cpu processing time per kilobyte. However, there were limits to the amortization. For instance, once the disk block size exceeded the size of a particular file, no further reduction in access times to that file could be obtained by further increasing the block size. The same applies to the packet size over the medium. Another consideration may be that some costs were proportional to the amount of data processed, so the marginal benefit of amortization decreases.

3. Since the server cpu constituted by a long way the system's bottleneck, an obvious modification to improve performance was to speed up the processor. From the estimates for the utilizations of the hardware resources in this case it could be concluded that the utilizations of the server cpu and the disk were almost identical. Therefore, speeding up the processor even further would not lead to an equivalent performance improvement as the disk itself would then become the bottleneck. Consequently, further improvements to the processor at the server site would also require improvements to the disk subsystem, e.g. by adding a second disk in order to increase profitability.

4. Another approach to improve file service performance at high loads would be by adding a second identical file server to the network to share the load imposed by the clients. The results showed that this would be the best improvement to the baseline case. It should however be noted that the above conclusions only reflect performance characteristics and do not consider cost aspects.

Appendix A - ODD Contents

	A	B	C
1		**Requirements**	**Test Case Material**
2			
3	**Unit**	**Code**	UØØ1
4		**Name**	Credit Department - Industrial Clients
5		**Localization**	Building A
6		**Floor**	1st floor
7		**Address**	Hochstraße 2
8		**Headclerk**	Mr. Weber
9		**Higher unit**	Domestic Credit Management
10		**OA sensivity comment**	No experience with oa; no idea
11		**OA sensivity quote**	medium
12		**OA viability comment**	High expectation and motivation
13		**OA viability quote**	high
14		**Surface of the unit?**	300 m2
15		**Costs of the premises**	DM 400 000/year
16		**Rooms**	100.000 DM
17		**Furnishing**	50.000 DM
18		**Material**	40.000 DM
19		**Energetic**	40.000 DM
20		**External**	10.000 DM
21		**Instruments**	100.000 DM
22		**Software**	60.000 DM
23		**Mission of the unit**	Acquisition and administration of credits
24	**Objective**	**Code of the objective**	OØØ1
25		**Name of the objective**	Acquisition
26		**Description of the objective**	getting high amount-credits and/or a large number of clients
27		**Quote of value**	high
28		**Indicator of result**	Number of Credits/Amounts of Credits
29		**Priority of the objective**	9
30		**Target for the objective**	100 per year
31		**Quantity of P-O**	70 per year
32		**Periodicity of P-O**	weekly
33		**Length of P-O-cycle**	2 months
34		**Reliability**	8
35		**Timelity of P-O**	8
36		**Completeness of P-O**	7
37		**Fomal level of P-O productive cyle**	1
38			-short informal notes about telephone contacts
39			- reports about visits
40		**Instruments & support**	Phone
41			Information system /media = Terminal(electronically)
42			Reports of E&P/media = paper
43			Annual reports/ media = paper
44			Newspaper articles/ media = paper
45			former contracts/ media = paper
46			information about client referred from ext. database/ media = print

	A	B	C
47			Fax
48			Telex
49			Type Writer
50			Word Processing Machines
51		**Comments on critical factors**	Reliable information about client, speed of inf. transfer
52		**Comments about improvements of P-O**	Client's data faster, improve the internal turnaround
53	**Objective**	**Code**	OØØ2
54		**Name of the objective**	Formalization and definition of the offer
55		**Description of the objective**	procedures concerning the content and layout of the final contract
56		**Quote of value**	9
57		**Indicator of result**	Complete contract
58		**Priority of the objective**	medium - high
59		**Target for the objective**	100 contracts per year
60		**Quantity of P-O**	70 contracts per year
61		**Periodicity of P-O**	weekly
62		**Length of P-O-cycle**	3 months
63		**Reliability**	9
64		**Timelity of P-O**	9
65		**Completeness of P-O**	9
66		**Fomal level of P-O productive cyle**	7
67			Standardized procedures are followed (preperations of documents)
68			forms (contracts) must be filled
69			standardized calculations (solvency, rates etc.)
70			
71		**Instruments & support**	Contract/media= paper
72			Information system/Media= Terminal(electronically)
73			Mail/ media = paper
74			Balance sheets/Media = paper
75			Credit approval form/ media = paper
76			Checklist/media = paper
77			Computer lists about client's status/ media = paper
78			Phone
79			Fax
80			Telex
81			PC
82			IBM Workstation
83			Type writer
84			Word Processing machines
85	**Objective**	**Code**	OØØ3
86		**Description of the objective**	Administration
87		**Quote of value**	9
88		**Indicator of result**	optimal retrieval of clients' data for examination of annual solvency
89		**Priority of the objective**	medium
90		**Target for the objective**	good
91		**Quantity of P-O**	300 per year
92		**Periodicity of P-O**	daily

	A	B	C
93		Length of P-O-cycle	1 week
94		Reliability	8
95		Timelity of P-O	9
96		Completeness of P-O	9
97		Fomal level of P-O productive cyle	9
98			Standardized procedures of filing and retrieving
99			Standardized calculation of solvency and return on capital
100		Instruments & support	sheets for retrieval/media = paper
101			Prefabricated calculation sheets/ media = paper
102			Balance sheets
103			Annual reports
104			Newspaper articles
105			Information services
106			MA= 40%
107			DP= 18%
108			OA = 2%
109			CE = 40%
110		Media of information sources	Phone
111			Information system/IBM 3270
112			Fax
113			Telex
114			PC
115			Type Writer
116			Word Processing Machines
117			Mail
118	Archives		
119		Central Archive	IAØØ1
120		name	finished clients' contracts after successful repayment
121		Localization	Building B, Hochstraße 4
122		Number of items	1 million folders / 200 m2
123		Type of sorting	year/alphabetically
124		Time of storage	10 years (legally)
125		Annual increase of archives	10%
126		Analysts description of problems	80% of documents ought to be stored in a paper version because of legal reasons
127			
128			
129		Departamental Archive	IAØØ2
130		name	clients' contracts in work/information about clients i.e. from E&P
131		Number of items	3000 folders, each containing at most 10 different clients' files (at least 30 pages)
132		Type of sorting	alphabetically, within the same client by date
133		Time of storage	at most 10 years
134		Annual increase of archives	25%
135		Analysts description of problems	Redundant data which are also stored in Economics&Press Departamental Archive
136			
137		Personal Archive	IAØØ3-999
138		name	actual contracts = part of the departamental archive/correspondence

	A	B	C
139		Number of items	10 folders at most
140		Type of sorting	alphabetically
141		Time of storage	4 months at most
142			
143		Unit&Product Objective	
144		Acquisition	50%
145			
146		Offer Definition	30%
147			
148		Administration	20%
149			
150	Units & P-O used	P-O coming from E&P	
151		- completeness	90%
152		- reliability	90%
153		- timelity	80%
154		P-O coming from Board of directors	too late
155		- completeness	90%
156		- reliability	100%
157		- timelity	70%
158			
159	Human Resource	Name	Mr. Schmitz
160		Code	Acquisition manager
161		Level	Dr., University
162		Cost	DM150.000
163		Duty	Acquisition, contacts
164		Duty starting date	6.Jan 80
165		Birth Date	3.Aug 40
166			Terminal
167	Instrument	Name	IBM 3270
168		Topologic use	P
169		Technical type	DP
170		Localization	desk
171		Costs(Maintenance, insurance etc.)	DM 15.000 including operating expenses
172		Operating expenses	see above
173		Analyst's opinion about the use	Work station would be preferable
174	HR and archives	Average number of accesses	central= 2 per year
175			Credit departamental archive = 15 per day
176			personal = 20 per day
177		Periodicity of accesses	hourly
178		Frequency of unsuccessful accesses	10%
179		Average borrowing time	2 months from departamental and central archive
180		Access purpose	Information about already existing contracts and financial situation of the client
181			
182	Units, P-O, Human R.	Acquisition	90%
183		% of time -external contacts	50%
184		-internal meeting	10%

	A	B	C
185		-reading	5%
186		-communication	20%
187		-writing	5%
188		-searching	5%
189		-mail handling	5%
190	Instruments in activities		
191		Communication	
192		Telephone	60%
193		Fax	30%
194		Telex	10%
195		Reading	
196		VDU	15%
197		Credit Departamental Archive	35%
198		Personal Archive	50%
199		Writing	
200		VDU	100%
201		Searching	
202		VDU	20%
203		Credit Departamental Archive	40%
204		Personal Archive	40%
205		Mail Handling	
206		Fax	60%
207		Telex	40%
208			
209		Offer Definition & Approval	10%
210		% of time -external contacts	10%
211		-internal meeting	20%
212		-reading	10%
213		-communication	30%
214		-writing	10%
215		-searching	10%
216		-mail handling	10%
217	Instruments in activities	Communication	
218		Telephone	10%
219		Fax	50%
220		Telex	40%
221		Reading	
222		VDU	15%
223		Credit Departamental Archive	65%
224		Personal Archive	20%
225		Writing	
226		VDU	100%
227		Searching	
228		VDU	10%
229		Credit Departamental Archive	50%
230		Personal Archive	40%

	A	B	C
231		Mail Handling	
232		Fax	50%
233		Telex	50%
234			
235	Human Resource	Name	Mrs. Schneider
236		Code	clerical worker
237		Level	High school
238		Cost	DM 60 000
239		Birth Date	2.Dez 52
240		Duty	Credit administration
241		Duty starting date	1.Mär 73
242	Instrument	Name	Terminal 3270
243		Localization	desk
244		Technical type	DP
245		Topologic use	U
246		Costs(Maintenance, insurance etc.)	DM 6.000 including operating expenses
247		Operating expenses	
248		Analyst's opinion about the use	Work station necessary
249	HR and archives	Average number of accesses	central= 0
250			Credit departamental=20 per day
251			personal = 10 per day
252		Periodicity of accesses	hourly
253		Frequency of unsuccessful accesses	10%
254		Average borrowing time	4 months from departamental archive
255		Access purpose	Entering of mail, further information, actual calculations
256			
257	Units, P-O, Human R.	Acquisition	5%
258		-communication	10%
259		-searching	90%
260	Instruments in activities	Communication	
261		Telephone	80%
262		Fax	10%
263		Telex	10%
264		Searching	
265		VDU	10%
266		Credit Departamental Archive	55%
267		Personal Archive	35%
268			
269		Offer Definition & Approval	85%
270		-communication	10%
271		-writing/calculating	60%
272		-searching	20%
273		-mail handling	10%
274	Instruments in activities	Communication	
275		Telephone	50%
276		Fax	25%

	A	B	C
277		Telex	25%
278		Writing	
279		VDU	100%
280		Searching	
281		VDU	10%
282		Credit Departamental Archive	60%
283		Personal Archive	30%
284		Mail Handling	
285		Fax	50%
286		Telex	50%
287			
288		Administration	10%
289		-searching	100%
290	Instruments in activities	Searching	
291		Credit Departamental Archive	70%
292		Personal Archive	30%
293			
294	Human Resource	Name	Mrs. Wagner
295		Code	Credit administration
296		Level	Secondary School
297		Cost	DM 50 000
298		Birth Date	2.Feb 60
299		Duty	Credit administration
300		Duty starting date	3.Jan 80
301			
302	Instrument	Name	Uses terminal of Mrs. Schneider
303		Topologic use	U
304		Technical type	DP
305		Localization	no distance between the 2 desks
306		Costs (Maintenance, Insurance etc.)	see line 246
307		Operating expenses	
308		Analyst's opinion about the use	Own terminal
309	HR and archives	Average number of accesses	central = 0
310			Credit departamental = 25 per day
311			personal= 0
312		Periodicity of accesses	hourly
313		Frequency of unsuccessful accesses	5%
314		Access purpose	entering of actual contract additionals
315			
316			
317		Average borrowing time	4 months
318			
319	Units, P-O, Human R.	Acquisition	5%
320		-searching	100%
321	Instruments in activities	Searching	
322		Credit Departamental Archive	100%

	A	B	C
323			
324		Offer Definition & Approval	90%
325		-writing	50%
326		-searching	10%
327		-communication	10%
328		-mail handling	30%
329	Instruments in activities	Communication	
330		Telephone	80%
331		Fax	10%
332		Telex	10%
333		Writing	
334		VDU	100%
335		Searching	
336		Credit Departamental Archive	100%
337		Mail Handling	
338		Fax	50%
339		Telex	50%
340			
341		Administration	5%
342		searching	100%
343	Instruments in activities	Searching	
344		Credit Departamental Archive	100%
345			
346	Human Resource	Name	Mr. Müller
347		Code	clerical worker
348		Level	Secondary school
349		Cost	DM 60 000
350		Birth Date	6/24/1936
351		Duty	Credit administration
352		Duty starting date	6.Jan 65
353			
354	Instrument	Name	typewriter
355		Localization	desk
356		Topologic use	P
357		Costs (Maintenance, Insurance etc.)	DM 500 including operating expenses
358		Operating expenses	-
359		Analyst's opinion	Own workstation/Terminal
360	HR and archives	Average number of accesses	Central = 5 per year
361			Credit Departamental = 30 per day
362			personal = 10 per day
363		Periodicity of accesses	hourly
364		Frequency of unsuccessful accesses	10%
365		Average borrowing time	3 months
366		Access purpose	adding of calculation sheets, retrieval for annual solvency checking
367			
368	Units, P-O, Human R.	Offer Definition and Approval	100%

	A	B	C
369		-communication	10%
370		-writing	50%
371		-searching	10%
372		-mail handling	30%
373	Instruments in activities	Communication	
374		Telephone	20%
375		Fax	40%
376		Telex	40%
377		Writing	
378		Typewriter	100%
379		Searching	
380		Central Archive	10%
381		Credit Departamental Archive	70%
382		Personal Archive	20%
383		Mail Handling	
384		Fax	60%
385		Telex	40%
386			
387	Human Resource	Name	Ms Rothe
388		Code	secretary
389		Level	elementary school
390		Cost	DM 40 000
391		Birth Date	1/23/1959
392		Duty	Typing, Filing (Credit Administration)
393		Duty starting date	3.1.85
394	Instrument	Name	Typemachine/memory
395		Localization	desk
396		Topologic use	P
397		Costs(Maintenance, insurance etc.)	DM 600 including operating expenses
398		Operating expenses	-
399		Analyst's opinion	needs an individual terminal or PC
400		Average number of accesses	Credit departamental = 15 per day
401	HR and archives	Periodicity of accesses	hourly
402		Frequency of unsuccessful accesses	15%
403		Average borrowing time	1 hour
404		Access purpose	Adding copies of letters
405			
406	Units, P-O, Human R.	Acquisition	5%
407		-writing	100%
408			
409		Offer Definition & Approval	85%
410		-writing	75%
411		-searching	10%
412		-mail handling	10%
413		-communication	5%
414	Instruments in activities	Writing	

	A	B	C
415		Typewriter	100%
416		Searching	
417		Credit Departamental Archive	100%
418		Mail Handling	
419		Typewriter	50%
420		Fax	30%
421		Telex	20%
422		Communication	
423		Telephone	60%
424		Fax	30%
425		Telex	10%
426			
427		Administration	10%
428		-writing	50%
429		-searching	50%
430	Instruments in activities	Writing	
431		Typewriter	100%
432		Searching	
433		Credit Departamental Archive	100%
434			
435	Unit	Code	?
436		Name	Economics and Press
437		Localization	Building A
438		Floor	2 nd floor
439		Address	Hochstraße 2
440		Headclerk	Mr. Schwarze
441		Higher unit	Management Services
442		OA sensivity comment	Experience with PCs
443		OA sensivity quote	high
444		OA viability comment	high
445		OA viability quote	9
446		Surface of the unit?	100 m2
447		Costs of the premises	154.000 DM
448		Rooms	40.000 DM
449		Furnishing	13.000 DM
450		Material	12.000 DM
451		Energetic	14.000 DM
452		External	10.000 DM
453		Instruments	50.000 DM
454		Software	15.000 DM
455		Mission of the unit	Economics and Press, Service for the provision of specific data about clients.
456	Objective	Code of the objective	?
457		Name of the objective	Organizing credit (solvency) information, actual and common info about potential clients
458		Priority of the objective	High
459		Indicator of result	Sufficient data about every client
460		Target for the objective	Distribution of required information within 1 day

	A	B	C
461		Quantity of P-O	
462		Periodicity of P-O	daily
463		Length of P-O-cycle	3 days
464		Reliability	90%
465		Timelity of P-O	80%
466		Completeness of P-O	sufficient
467		Fomal level of P-O productive cyle	medium
468		Level of support	MA = 42%
469			DP= 18%
470			OA = 20%
471			CE = 20%
472		Media of information sources	Phone
473			Reports
474			Letters
475			Newspaper articles
476			Credit Information institutions (CREDITREFORM, EBIL), media = external databases
477		Comments on critical factors	Insufficient data
478		Comments about improvements of P-O	Client's data faster
479	Archives		
480		Departamental Archive	
481		name	informations about clients (newspaper articles, balance sheets etc.)
482		Number of items	1000 folders
483		Type of sorting	alphabetically
484		Time of storage	2 years
485		Annual increase of archives	15%
486		Analysts description of problems	Redundant data with Credit Department
487			
488		Personal Archive	
489		name	actual clients
490		Number of items	10 folders with short notes, newspaper articles
491		Type of sorting	alphabetically, within the client by date
492		Time of storage	1 month
493			
494	Unit&Product Objective	Acquisition	
495		- completeness	90%
496		- reliability	90%
497		- timelity	70%
498		Offer Definition	
499		- completeness	90%
500		- reliability	90%
501		- timelity	70%
502		Administration	0
503	Units & P-O used	P-O coming from Credit Department	ok.
504			
505	Human Resource	Name	Mr. Heinemann
506		Code	expert

	A	B	C
507		Level	University
508		Cost	DM 95 000
509		Duty	Manager assistant
510		Duty starting date	9.Jan 82
511		Birth Date	2.Jul 54
512			
513	Instrument	Name	IBM PC AT
514		Localization	Own office, see above
515		Technical type	OA(including Communication Software for ext. databases)
516		Topologic use	P
517		Costs(Maintenance, insurance etc.)	DM 2.700 including operating expenses
518		Operating expenses	-
519		Analyst's opinion about the use	SW must be integrated, integration with inf. systems
520			
521	Software	Code	
522		Name	Open Access
523		Release	-
524		Version	A1.01
525		Memory required	360
526		Cost	1500
527			
528		Name	Text 4
529		Version	version1.05
530		Memory required	360
531			
532	HR and archives	Average number of accesses	central= 2 per year
533			Economics & Press departamental archive = 20 per day
534			personal archive = 5 per day
535		Periodicity of accesses	hourly
536		Frequency of unsuccessful accesses	10%
537		Average borrowing time	1 week
538		Access purpose	Retrieval of previous information about the clients
539		Analyst's opinion	Redundant data between Credit Departamental and E&P departamental archive
540			
541	Units, P-O, Human R.	Acquisition	100%
542		-communication	10%
543		-reading	45%
544		-writing	25%
545		-searching	10%
546		-mail handling	10%
547			
548	Instruments in activities	Communication	
549		Telephone	80%
550		Fax	10%
551		Telex	10%
552		Reading	

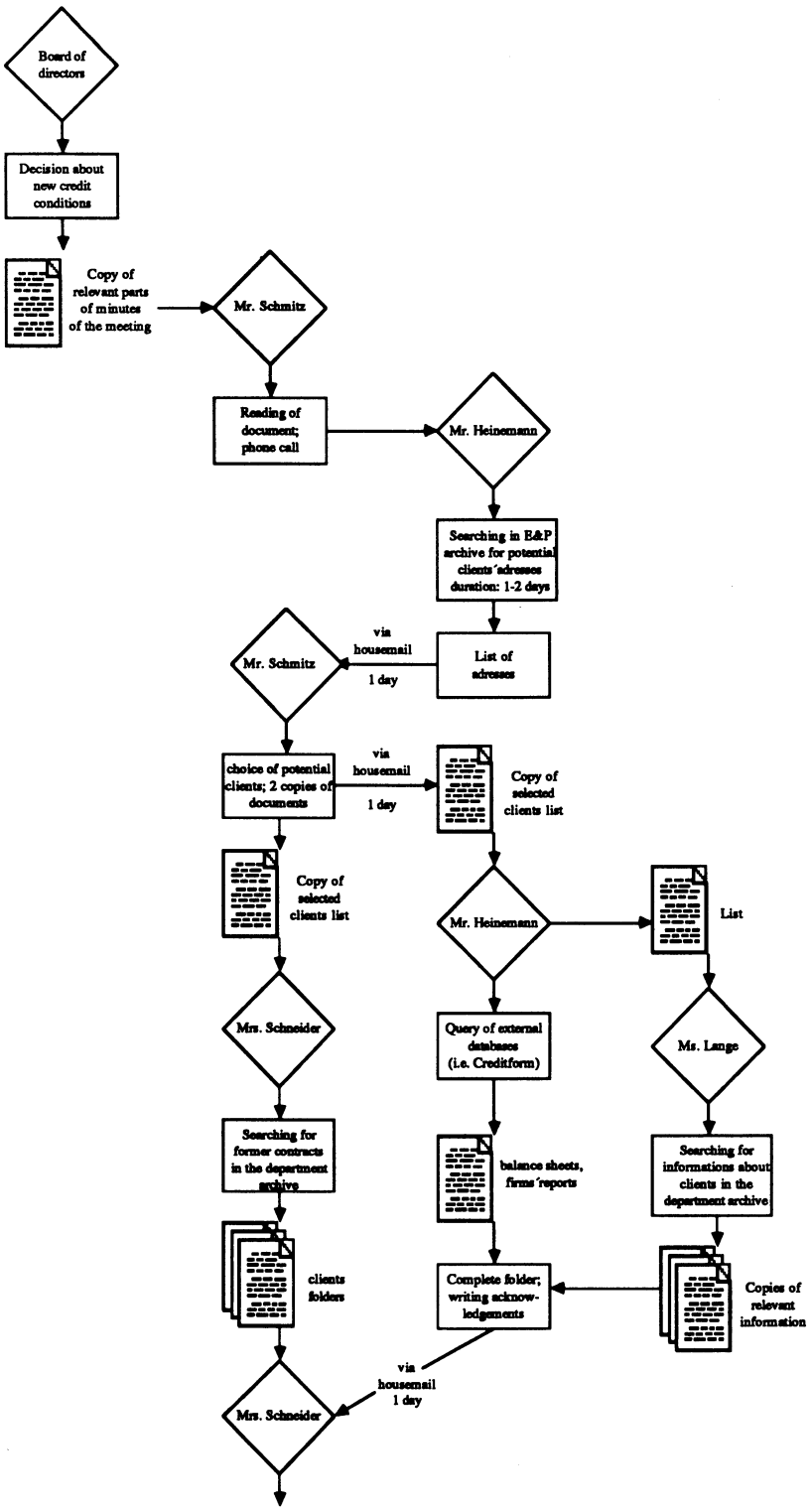
	A	B	C
553		E&P Archive	90%
554		Personal Archive	10%
555		Writing	
556		IBM PC	100%
557		Searching	
558		E&P Archive	70%
559		Credit Departamental Archive	4%
560		Central Archive	8%
561		Personal Archive	18%
562		Mail Handling	
563		Fax	10%
564		IBM PC	80%
565		Telex	10%
566			
567	Human Resource	Name	Mr. Schüler
568		Code	expert
569		Level	University
570		Cost	DM 70 000
571		Birth Date	1.Mär 79
572		Duty	Manager assistant
573		Duty starting date	1.Mär 79
574			
575	Instrument	Name	IBM PC XT
576		Topologic use	P
577		Technical type	OA
578		Localization	
579		Costs(Maintenance, insurance etc.)	DM 2.700 including operating expenses
580		Operating expenses	none
581		Analyst's opinion about the use	SW must be integrated
582			
583	Software	Code	
584		Name	Open Access
585		Release	?
586		Version	A 1.01
587		Memory required	360
588		Cost	1500
589			
590		Name	Text 4
591		Version	version1.05
592		Memory required	360
593			
594	HR and archives	Average number of accesses	
595			E&P departamental archive = 70 per day
596			Personal archive = 30 per day
597		Periodicity of accesses	hourly
598		Frequency of unsuccessful accesses	5%

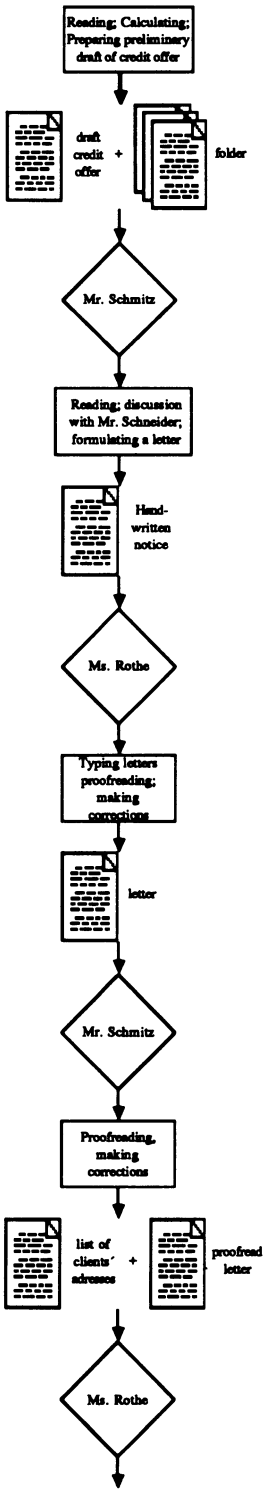
	A	B	C
599		Average borrowing time	1 week
600		Access purpose	Retrieval of previous information about the client, adding new information (balance sheets)
601			
602	Units, P-O, Human R.	Acquisition	100%
603		-communication	10%
604		-reading	55%
605		-writing	15%
606		-searching	10%
607		-mail handling	10%
608			
609	Instruments in activities	Communication	
610		Telephone	80%
611		Fax	10%
612		Telex	10%
613		Reading	
614		E&P Archive	60%
615		Personal Archive	40%
616		Writing	
617		IBM PC	100%
618		Searching	
619		E&P Archive	68%
620		Credit Departamental Archive	2%
621		Central Archive	2%
622		Personal Archive	28%
623		Mail Handling	
624		Fax	10%
625		IBM PC	80%
626		Telex	10%
627		Analyst's opinion	
628			
629	Human Resource	Name	Ms Lange
630		Code	secretary
631		Level	elementary school
632		Cost	DM 40 000
633		Birth Date	3/30/1958
634		Duty	Typing, Filing/Credit Administration
635		Duty starting date	3.Jan 80
636	Instrument	Name	Typemachine/memory
637		Localization	desk
638		Topologic use	P
639		Costs(Maintenance, insurance etc.)	DM 600 including operating expenses
640		Operating expenses	-
641	HR and archives	Average number of accesses	E&P departamental archive = 15 per day
642			personal archive = 3 per day
643		Periodicity of accesses	hourly
644		Frequency of unsuccessful accesses	10%

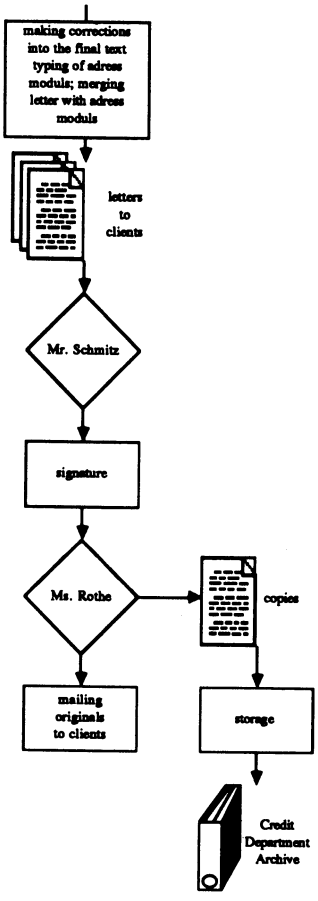
	A	B	C
645		Average borrowing time	1 day
646		Access purpose	Finding folders indicated by mr. Heinemann or Mr. Schüler
647			
648	Units, P-O, Human R.	Acquisition	100%
649		-writing	80%
650		-searching	10%
651		-mail handling	10%
652	Instruments in activities	Writing	
653		Type Machine	100%
654		Searching	
655		E&P Departamental Archive	78%
656		Personal Archive	20%
657		Credit Departamental Archive	2%
658		Mail Handling	
659		Type Machine	80%
660		Fax	10%
661		Telex	10%
662			
663		Code	?
664	Unit	Name	Number 2
665		Localization	Building B
666		Floor	2nd floor
667		Address	Hochstraße 4
668		OA sensivity comment	no experience with OA
669		OA sensivity quote	medium
670		OA viability comment	highly motivated
671		OA viability quote	high
672		Surface of the unit?	high
673		Costs of the premises	200m2
674		Costs of the furnishings	200.000 DM/per year
675		Costs of consumed materials	40.000 DM/per year
676		Costs of heating, electricity	20.000 DM/per year
677		Costs of external services	7.000 DM/per year
678		Mission of the unit	Board of directors
679	Objective	Code of the objective	
680		Name of the objective	Approval of Credits
681		Priority of the objective	10
682	@	Indicator of result	amount of credit/ number of clients
683		Target for the objective	100 per year

Appendix B - Office Activities Definition

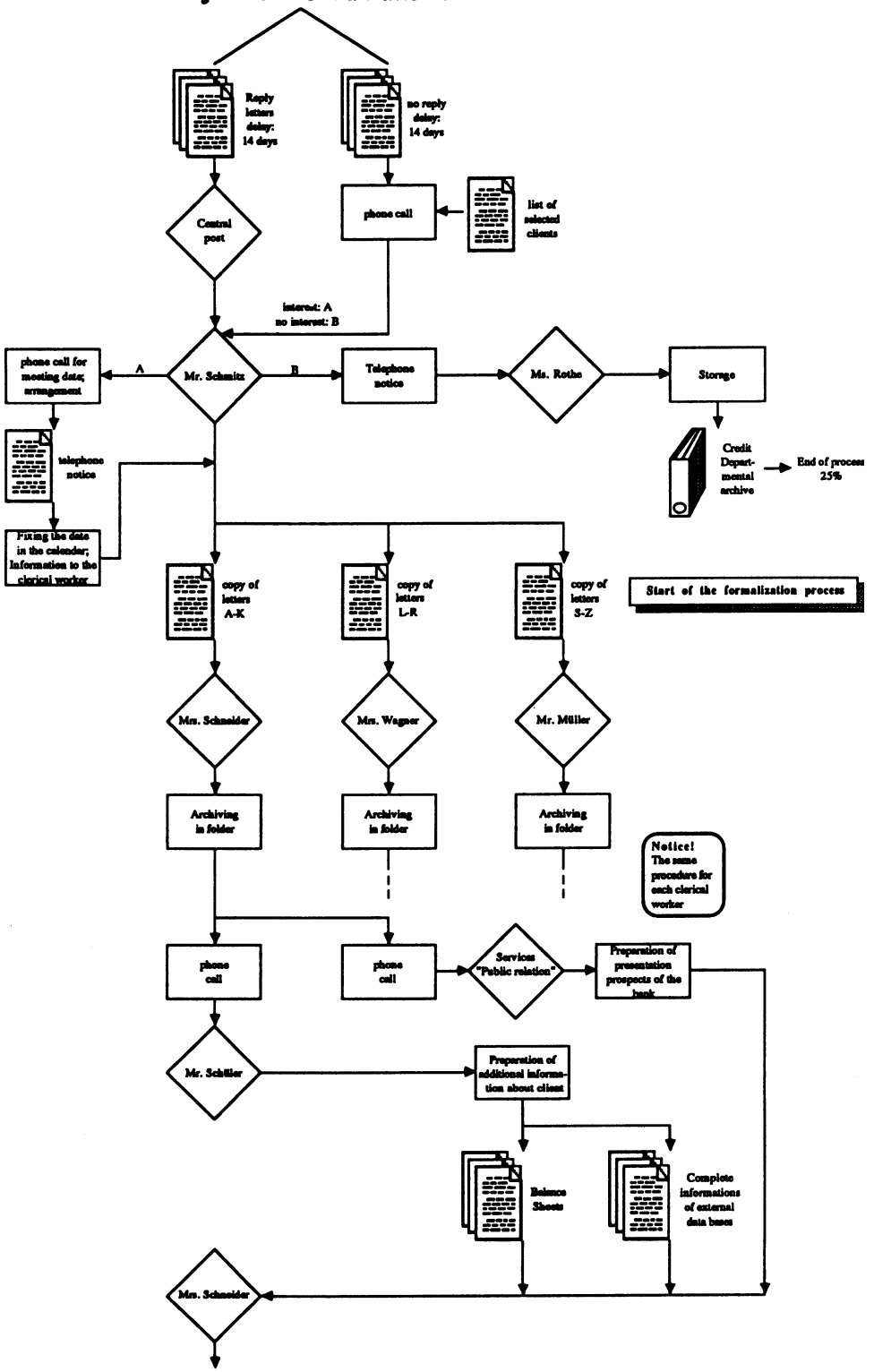
Product-Objective "Acquisition"

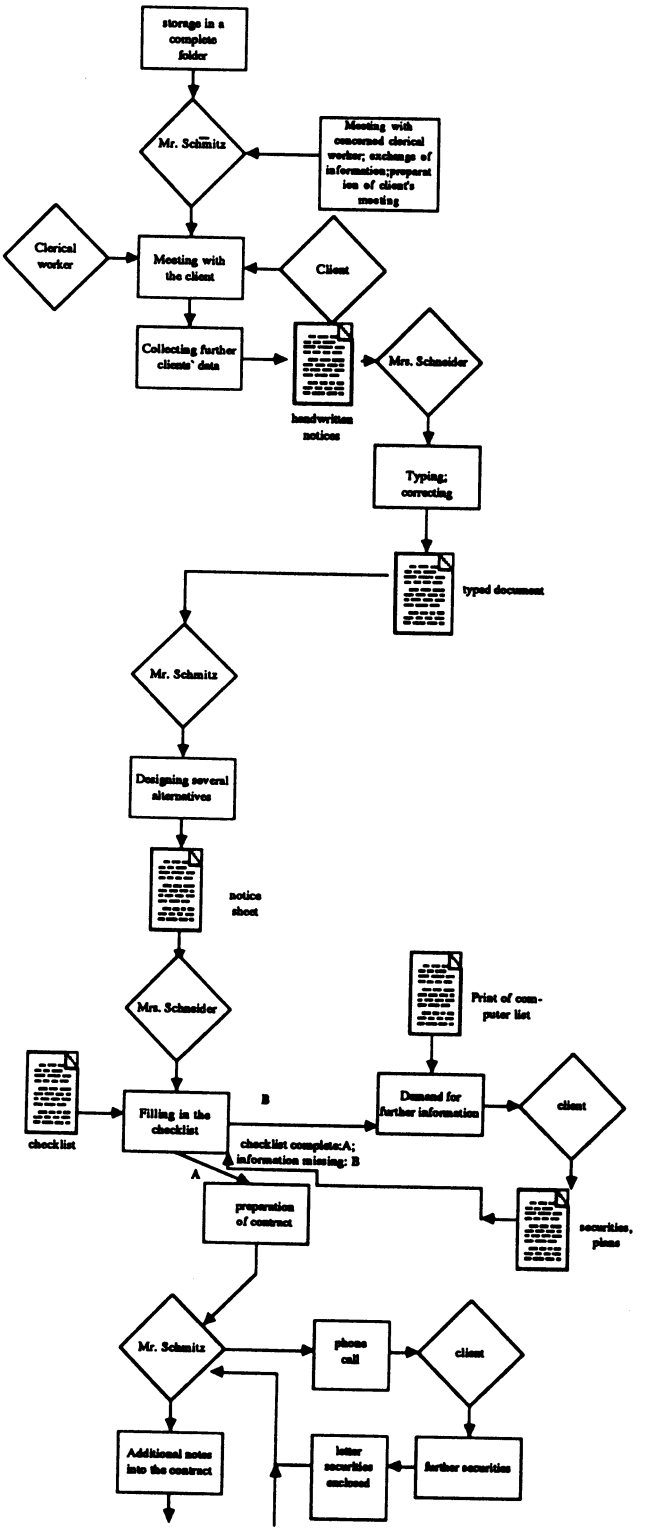



Reading; Calculating; Preparing preliminary draft of credit offer
draft credit offer
+
folder
Mr. Schmitz
Reading; discussion with Mr. Schneider; formulating a letter
Hand-written notice
Ms. Rothe
Typing letters proofreading; making corrections
letter
Mr. Schmitz
Proofreading, making corrections
list of clients' adresses
+
proofread letter
Ms. Rothe

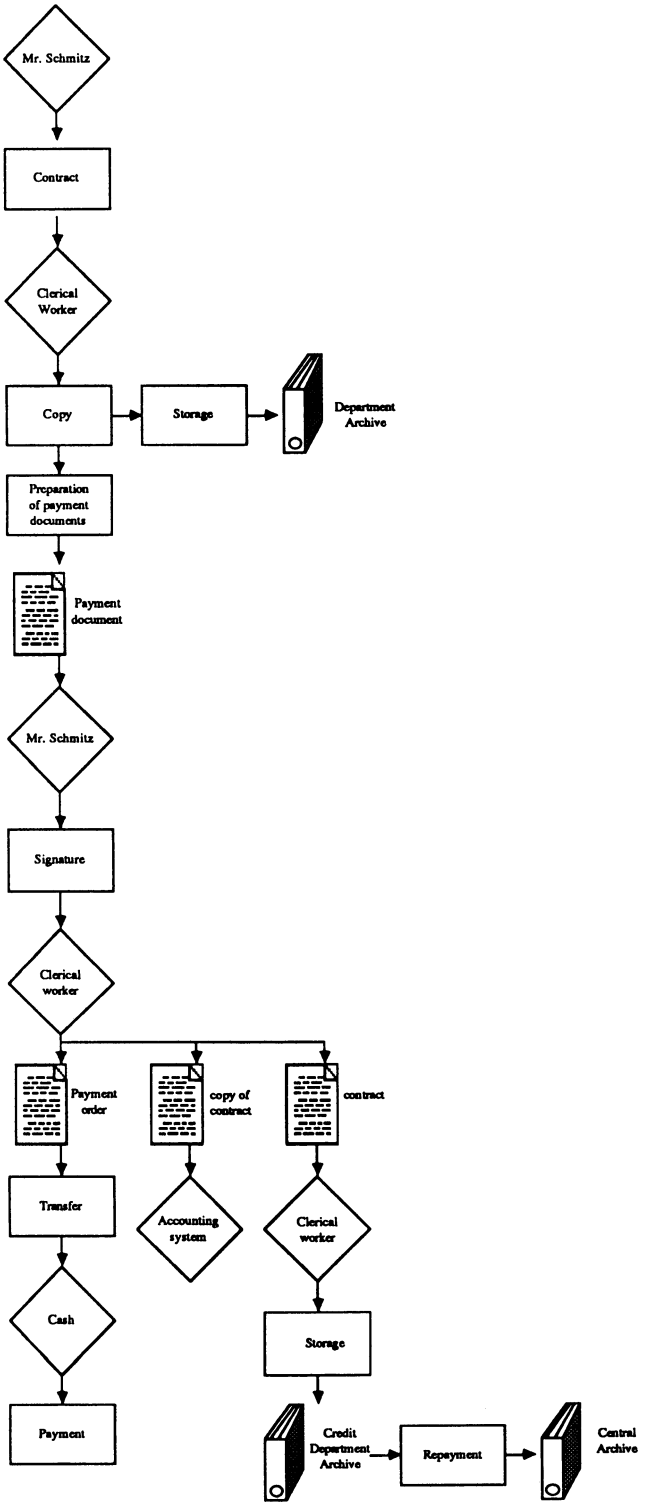

making corrections into the final text typing of adress moduls; merging letter with adress moduls
letters to clients
Mr. Schmitz
signature
Ms. Rothe
copies
mailing originals to clients
storage
Credit Department Archive

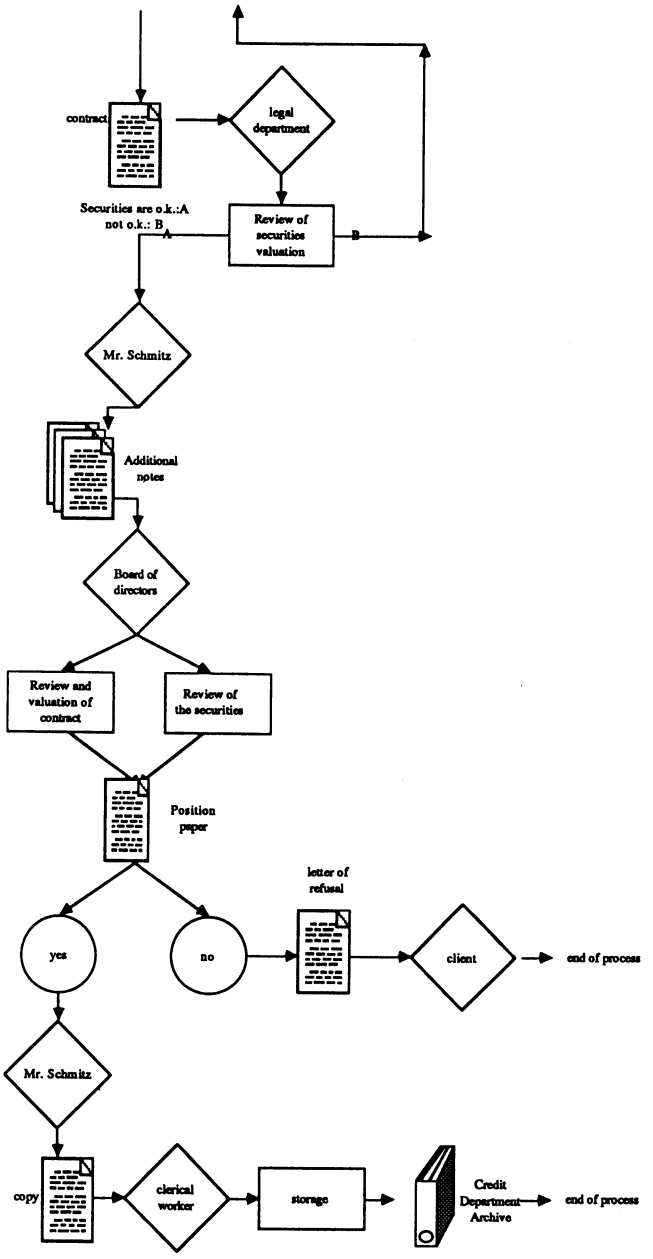
Product-Objective "Formalization and Definition of the Offer"




storage in a complete folder
Mr. Schmitz
Meeting with concerned clerical worker; exchange of information;preparation of client's meeting
Clerical worker
Meeting with the client
Client
Collecting further clients' data
handwritten notices
Mrs. Schneider
Typing; correcting
typed document
Mr. Schmitz
Designing several alternatives
notice sheet
Mrs. Schneider
Print of computer list
checklist
Filling in the checklist
B
Demand for further information
client
checklist complete:A; information missing: B
A
preparation of contract
securities, plans
Mr. Schmitz
phone call
client
Additional notes into the contract
letter securities enclosed
further securities

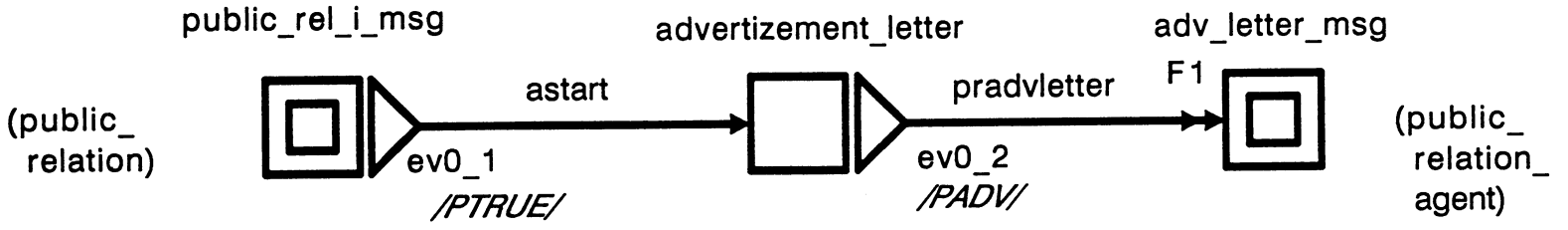
Product-Objective : Administration of the contract



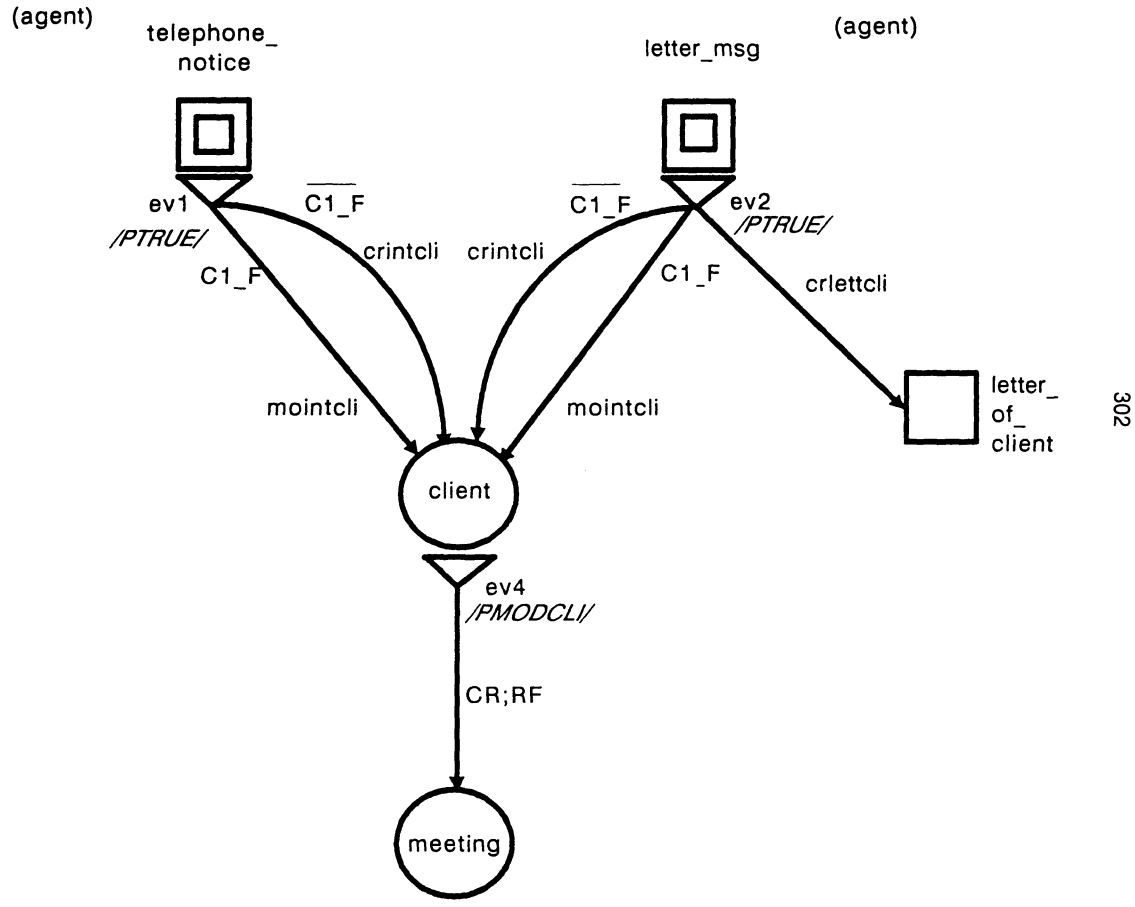

contract
legal department
Securities are o.k.:A
not o.k.: B
A
Review of securities valuation
B
Mr. Schmitz
Additional notes
Board of directors
Review and valuation of contract
Review of the securities
Position paper
letter of refusal
yes
no
client
end of process
Mr. Schmitz
copy
clerical worker
storage
Credit Department Archive
end of process

Appendix C - TCM dynamic graph

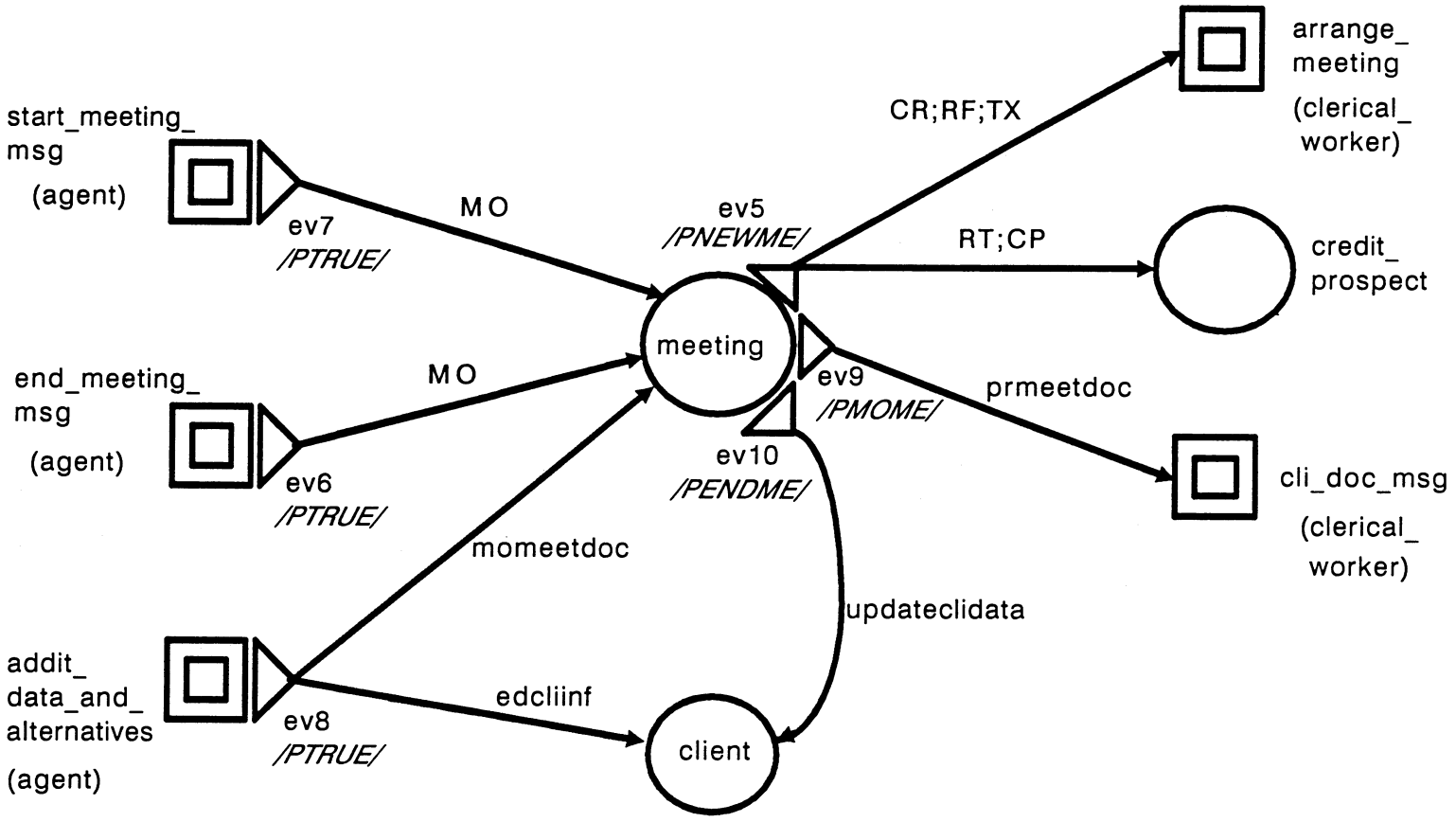
ADVERTIZEMENT



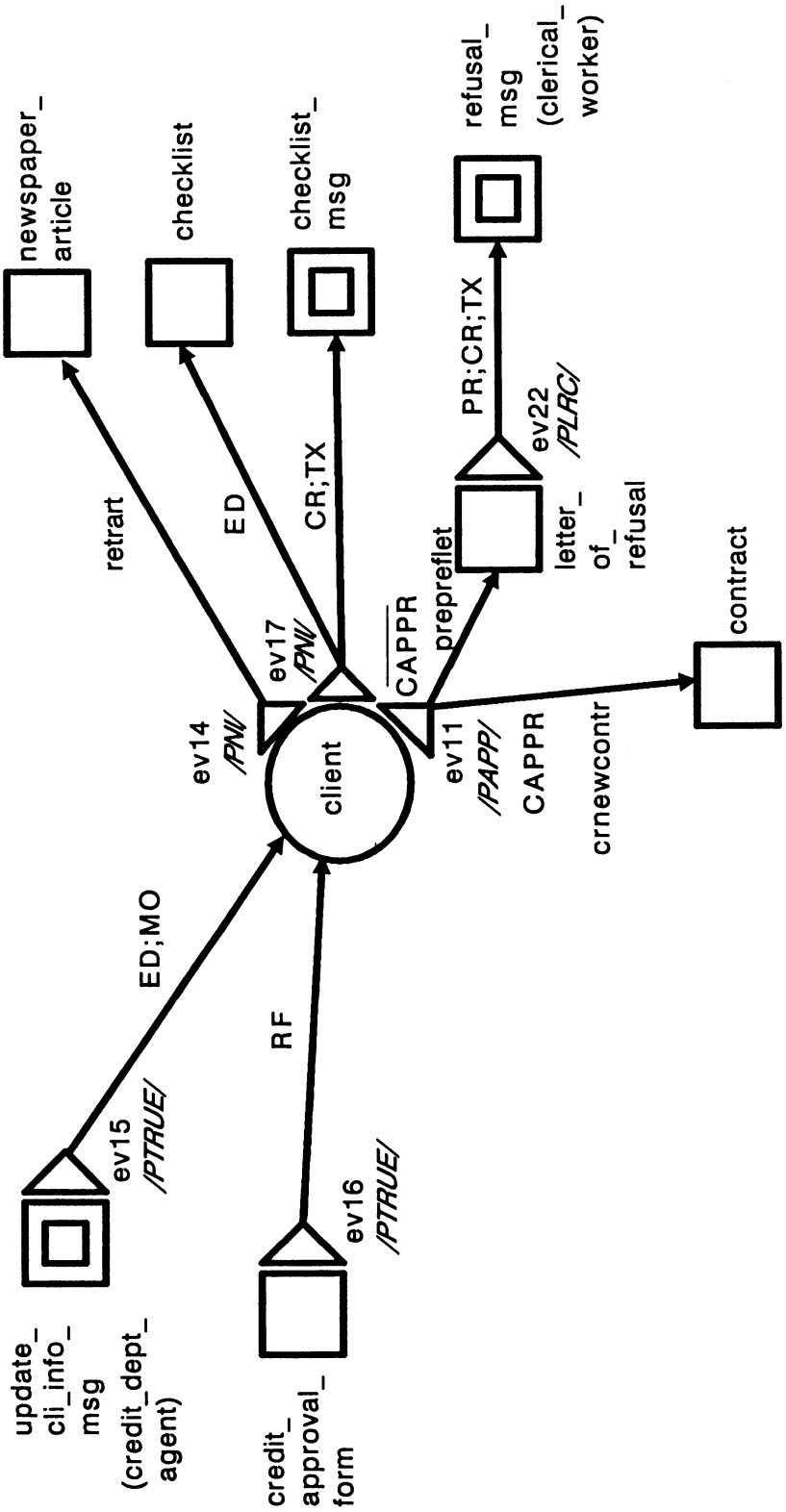
NEW CLIENT


(agent)
telephone_
notice
(agent)
letter_msg
ev1
/PTRUE/
C1_F
crintcli
C1_F
mointcli
C1_F
crintcli
C1_F
mointcli
ev2
/PTRUE/
crlettcli
letter_
of_
client
client
ev4
/PMODCLI/
CR;RF
meeting

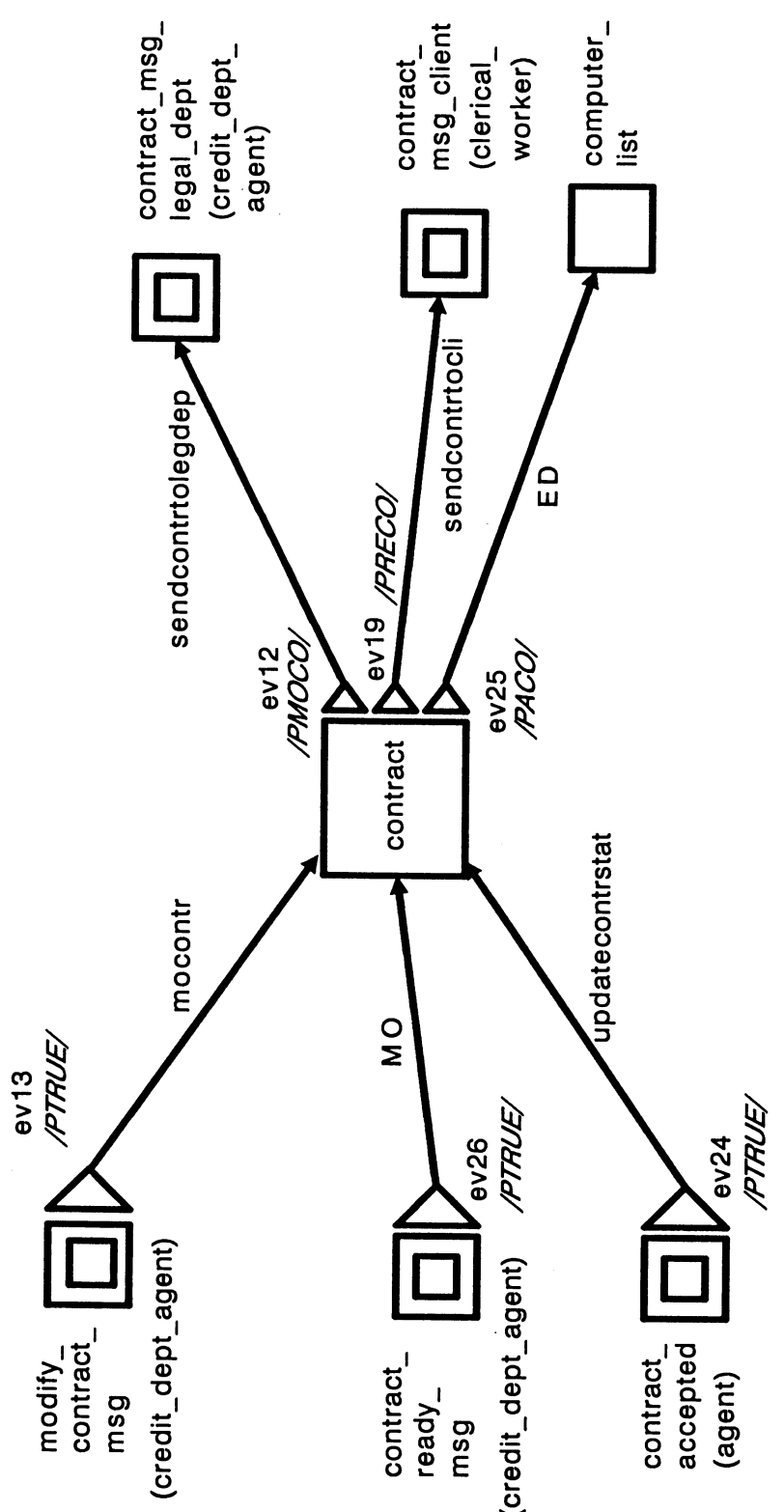
MEETING


start_meeting_
msg
(agent)
ev7
/PTRUE/
MO
end_meeting_
msg
(agent)
ev6
/PTRUE/
MO
addit_
data_and_
alternatives
(agent)
ev8
/PTRUE/
momeetdoc
edcliinf
meeting
client
ev5
/PNEWME/
CR;RF;TX
RT;CP
ev9
/PMOME/
prmeetdoc
ev10
/PENDME/
updateclidata
arrange_
meeting
(clerical_
worker)
credit_
prospect
cli_doc_msg
(clerical_
worker)

CONTRACT PREPARATION (1)



CONTRACT PREPARATION (2)



Chapter 7

Concluding Remarks and Future Work

Barbara Pernici, Colette Rolland, and the TODOS Team

Computer Aided Software Engineering (CASE) is gaining more and more popularity in the software development community. While its importance has become evident to handle design documentation and several tools present on the market provide such a functionality, still very limited functionalities are provided to support design decisions and to suggest possible design choices to the developer. The main aim of the TODOS approach is to provide a set of tools with powerful functionalities to support the designer in office information systems design.

TODOS has focused on the following aspects of office information systems design:

- *supporting analysis of requirements and of conceptual schemas*: the tools described in Chapters 2 and 3 help the designer in collecting information about the application under development, and provide powerful functionalities to analyze the consistency of the collected information. Moreover, it is possible to analyze collected information, under different perspectives, and some facilities are provided to complete missing information automatically.

- *rapid prototyping*: tools are provided to realize a prototype of the office system under development from the conceptual schema. Prototyping is very useful to support design decisions, in particular to gather users' evaluation of design choices, and to refine the design of user interfaces.

- *architecture selection*: architecture design is particularly important in the TODOS design life cycle. In fact, the design of the office system architecture is developed with the goal of reusing available software and hardware components. In applications such as office and information systems the development of original components for a given application should be limited as much as possible.

An original aspect in TODOS is the use of *information bases* to support the development process. Databases have been proposed in TODOS for the Requirements Collection and Analysis phase and in the Conceptual Design phase. Knowledge bases are used to support rapid prototyping and architecture selection. A future extension of the TODOS approach should provide an integrated information base to support the different phases of office systems development.

Other research projects are focusing on the use of information bases to support information systems design: the DAIDA ESPRIT project (Jarke and DAIDA Team

1988) proposes the use of a knowledge base, ConceptBase, to support design decisions, and the ITHACA (Integrated Toolkit for Highly Advanced Computer Applications) ESPRIT II project proposes the use of a Software Information Base to support the design phases and the reuse of software components (Proefrock et al. 1989).

An *office system development methodology* is proposed in TODOS. The methodology is based on a functional development life-cycle and a non-functional development life-cycle evolving in parallel. The office system is developed iterating design phases, interacting with the user, and refining design choices until the system is considered complete and consistent. The results of a project developed in TODOS are the functional specifications of the system, the definition of the user interfaces, and a hardware and software architecture. Detailed design is not covered by the TODOS methodology, since for this phase the developer can base his work on classical system development methods.

Some issues have to be considered in connection with the TODOS framework. First of all, the principal problem, particularly in requirements collection and analysis and in conceptual design, is the risk of automating existing procedures, including inefficiencies and paths that could be avoided using a computer based support environment in the office. While the goal of some office systems is to have a paperless office, a design that automates all existing office activities will produce the same amount of paper as in the non-automated office, or even more. The analysis of document flows and information exchange in the office is still a research issue. Exceptions in the office procedures should be considered, if they are important for correct functioning of the office. Not considering them could cause the realization of an inflexible system; inflexibility of office support systems is one one of the major causes of failure in the realization of information systems, and therefore also for an office system. On the other hand, taking into account all possible exceptions is a very heavy task in design. Therefore, models of exceptions should be a predefined part of the office scheme and automatically invoked when an exception arises. For instance, if the agent responsible for a given action is missing, a substitute should be automatically considered for performing the action. Intelligent support for the designer could be one solution to the above mentioned problems. Another important issue is that the style of work and the organizational structure can change rather often. Therefore, the support to system development and maintenance should be integrated with the system itself, in order to support system evolution: the phase of system development and system operation should not be considered in isolation, but supported in an integrated way.

These considerations are being addressed in current research projects. Research on information systems development environments is continuing in the second phase of the ESPRIT Programme of the Commission of the European Communities (ESPRIT II), in particular within the ITHACA and HECTOR projects. The ITHACA (Integrated Toolkit for Developing Highly Advanced Applications) project has the aim of developing an integrated application development and support environment, based on the object-oriented programming approach; within ITHACA, the goal of reusing software components is particularly important (Proefrock et al. 1989). Methodological issues have been considered within HECTOR (Harmonized European Concepts and Tools for ORganizational Information Systems), studying a reference framework for existing methods and methodologies based on requirements of a number of application areas.

The TODOS project was completed in December 1988. Tools developed within TODOS and described in this book have been engineered by industrial partners after the end of the project and are being used in office development projects.

For further details on work performed within TODOS, the authors of the chapters describing the different TODOS tools can be contacted (see the Authors' Address List).

References

Ahlsen M., Bjornerstedt, A., Briffs, S., Hulten, C., Soderlund, C.: An architecture for object management in OIS. ACM Trans. on Office Information Systems, *2* (3) (July 1984)

Avison, D.E.: Information Systems Development: A Data Base Approach. Oxford et al., 1985

Balser, R., Cheatham, T., Green, C.: Software technology in the 1990's: using a new paradigm. IEEE Computer (1983)

Barber, G.: Supporting organisational problems solving with a workstation. ACM Trans. on Office Information Systems, *1* (1), 45-67 (1983)

Barstow, D.R.: Domain-specific automatic programming. IEEE Trans. on Software Engineering *11*, November 1985

Baskett, F., Chandy, K.M., Munts, R.R., Palacios, F.G.: Open, closed and mixed networks of queues with different classes of customers. Journal of the ACM *22* (2), 248-260 (1975)

Beilner, H.: Workload characterisation and performance modelling tools. University of Dormund, Informatik IV, 1985

Borgida, A., Greenspan, S., Mylopoulos, J.: Knowledge representation as the basis for requirements specifications. IEEE Computer *18* (4), 82-91 (1985)

Borgida, A., Mylopoulos, J., Wong, H.K.T.: Methodological and computer aids for interactive information system development. In H.-J. Schneider, A.I. Wasserman (eds.): Automated Tools for Information System Design. North-Holland, Amsterdam 1982, pp. 109-124

Bracchi, G., Fugini, M.G.: Database support to office system design. Proc. 6th British Conf. on Databases (BNCOD 6), Cardiff, July 1988, pp. 47-67

Bracchi, G., Fugini, M.G.: Advanced design techniques for office system design. Proc. 11th IFIP Conf, Invited Paper, S. Francisco, CA, Aug. 1989, pp. 751-752

Bracchi, G., Pernici, B.: The design requirements of office systems. ACM Trans. Office Information Systems *2* (2) 151-170 (1984)

Bracchi, G., Pernici, B.: SOS: A conceptual model for office information systems. Data Base *15* (2), 11-18 (Winter 1984)

Bracchi, G., Pernici, B.: Trends in office modeling. In A.A. Verrijn-Stuart, R.A. Hirschheim (eds.): Proc. IFIP Conf. on Office Systems, Helsinki, North-Holland (1986), pp. 77-97

Brodie, M.: Automatic database design and development. ACM Conf. on Management of Data (SIGMOD 87), Tutorial, 1987

Brodie, M., Mylopoulos, J., Schmidt H-J.: On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer-Verlag, Berlin 1984

Brodie, M.L., Silva E.: Active and Passive Component Modelling: ACM/PCM. In T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (eds.): Information Systems Development Mehodologies: a Comparative Review. North-Holland, Amsterdam 1982, pp. 41-92

Carey, T.T., Mason, R.E.A.: Information system prototyping: techniques, tools, and methodologies. INFOR *21* (3) (1983)

Carmo, J.: The INFOLOG branching logic of events. In A. Sernadas et. al. (eds.): Information Systems: Theoretical and Formal Aspects. Elsevier Science Publishers B.V., North-Holland, Amsterdam - The Netherlands, 1985, pp. 159-174

Castelli, D., Meghini, C., Musto, D.: A knowledge representation approach to architecture specification in the office information system design. Proc. IFIP Working Conference on Office Information Systems: The Design Process, Lins, August 1988, North-Holland, 1989, pp. 95-107

Cassola, F., Galli, M., Pernici, B.: Diagnosis and correction of office system communication, accepted for publication on the Journal Decision Support Systems (1990)

Ceri, S. (eds.): Methodology and Tools for Database Design, North-Holland, Amsterdam 1983

Ceri, S., Tanca, L.: ELAND: An expert system for the configuration of local area networks applications. Proc. 11th Conference on Local Computer Networks, Minneapolis, Oct. 1986

Chang, S.K., Chan, W.L.: Transformation and verification of office procedures. IEEE Trans. on Software Engineering *11* (8), 724-734 (1985)

Conrath, D.W., De Antonellis, V., Simone, C.: A comprehensive approach to modeling office organisation and support technology. In B. Pernici, A.A. Verrijn-Stuart (eds.): Office Information Systems: the Design Process, North-Holland, Amsterdam 1989.

Cook, C.L.: Streamlining office procedures - An analysis using the Information Control Net model. Proc. AFIPS National Comp. Conf., May 1980

Croft, B., Lefkowits, L.S.: Task support in an office system. ACM Trans. on Office Information Systems, *2* (3), 197-212 (July 1984)

De Antonellis, V., Zonta, B.: A disciplined aprpoach to office analysis. IEEE Trans. on Software Engineering, Aug. 1990.

De Marco, T.: Structured Analysis and System Specification. Yourdon Press, New York 1979

Denning, P.J., Busen, J.P.: The operational analysis of queuing network models. ACM Computing Surveys *10* (3), 225-261 (1978)

Dolk, D.R., Kirsch, R.A.: A relational information resource dictionary system. Comm. of the ACM *30* (1), 48-61 (1987)

Dunnion, J., Harper, D.J., Amker-Moeller, B., Bogh, T., Sherwood-Smith, M., Van Rijsbergen, C.J.: Minstrel - ODM: a basic office data model, Proc. ESPRIT Technical Week '85, North-Holland, Brussels, Sept. 1985

Ellis, C.A.: Information Control Nets - A mathematical model of office information flow. Proc. ACM Conf. on Simulation, Measurement, and Modelling of Computer Systems, Boulder, Colorado, Aug. 1979, pp. 225-239

Ellis, C.A., Naffah, N.: Design of Office Information Systems. Springer-Verlag, Berlin 1987

Eriksson, D.: The conceptual representation in the RAMATIC tool. SYSLAB WP 91, SYSLAB-G, Systems Development Laboratory, Gothenberg, 1984

Ferrari, D.: Considerations on the insularity of performance evaluation. IEEE Trans. on Software Engineering *SE-12* (6), 678-683 (1986)

Flores, J., Ludlow, J.J.: Doing and speaking in the office. In R. Sprague (ed.): DSS: Issues and Challenges. Pergamon Press, London 1981

Floyd, C.: A systematic look at prototyping. In R. Budde et al: Approaches to Prototyping. Springer-Verlag, Berlin 1984

Fugini, M.G., Pozzi, S.: An approach to distributed conceptual design of office systems. In B. Pernici, A.A. Verrijn-Stuart (eds.): Office Information Systems: the Design Process, North-Holland, Amsterdam 1989, pp. 109-132

Goldberg, A.: Introducing the Smalltalk-80 System. Byte, *6* (8) (Aug. 1981)

Greif, I., Ellis, C.A. (eds.): Special issue on computer-supported cooperative work. ACM Trans. on Office Information Systems *5* (2), 113-211 (1987)

Gustafsson, M.R., Karlsson, M.R., Bubenko, J.A. jr.: A declarative approach to conceptual information modeling. In T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (eds.): Information Systems Development Mehodologies: a Comparative Review. North-Holland, Amsterdam 1982, pp. 93-142

Heidelberger, P., Lakshmi, M.S.: A performance comparison of multimicro and mainframe database architectures. IEEE Trans. on Software Engineering *SE-14* (4), 522-531 (1988)

Heijmink, F., Henry, P., Kieback, A., Rames, J.R.: Development of tools for designing OIS. In H.-J. Bullinger et al. (eds.): Information Technology for Organisational Systems. Elsevier Science Publishers B.V., North-Holland, Brussels - Luxembourg 1988, pp. 66-73

Hewitt, C.: Office are open systems, ACM Trans. on Office Information Systems, vol. 4, no. 3, pp. 271-287 (1986).

Hirscheim, R., Schaefer, G.: Requirements analysis: a new look at an old topic. Unpublished article, 1987

Hull, R., King, R.: Semantic database modeling survey, applications, and research issues. ACM Computing Surveys *19* (3), 201-260 (1987)

IEEE Software, Issue on CASE Technologies. (Mar. 1988)

ISO: Concepts and Terminology for the Conceptual Schema and the Information Base. J.J. van Griethuysen ed., ISO TC97/SC5/WG3 (1982)

ISO: Open Systems Interconnection - Basic Reference Model. IS 7498, 1983

Intellicorp: Kee Software Development System User's Manual. 1986

Jackson, M.: System Development. Prentice-Hall, Series in Computer Science, 1983

Jarke M., DAIDA Team: The DAIDA environment for knowledge-based information system development. In ESPRIT'88: Putting the Technology to Use, North-Holland, 1988, pp. 405-422

Jeffery, J.R., Laurence, M.: Systems Analysis and Design. Prentice-Hall International, Sydney (1984)

Johansson, L.A.: Use aspects of RAMATIC. SYSLAB WP 87, SYSLAB-G, Systems Development Laboratory, Gothenberg, 1984

Kaye, R.A., Karam, G.M.: Cooperating knowledge-based assistants for the office. ACM Trans. on Office Information Systems *5* (4), 297-326 (1987)

Kieback, A., Kerber, W.: Prototyping as an automatic tool for designing office systems. Proc. ESPRIT Technical Week '87, North Holland, 1987

Konsynski, B.R., Bracker, L.C., Bracker, W.E.: A model for specification of office communications. IEEE Trans. on Communication *COM-30* (1), 27-36 (1982)

Lasowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: File access performance of diskless workstations. ACM Trans. on Computer Systems *4* (3), 238-268 (1986)

Leavitt, H.J.: Applied organisation change in industry: structural, technical and human

approaches. In W.W. Cooper, H.J. Leavitt, M.W. Shelley (eds.): New Perspectives in Organization Research. Wiley, New York 1964

Levesque, H., Mylopoulos, J.: A procedural semantics for semantic networks. In N. Findler (ed.): Associative Networks. Academic Press, 1979

Lundeberg, M., Goldkuhl, G., Nilsson, A.: A systematic approach to information systems development: I. Introduction. In: Information Systems *4* (1979)

MacNair, E.A., Sauer, C.H.: Elements of Practical Performance Modelling. Prentice-Hall, 1985

Malone, T.W. et al.: Semistructured messages are surprisingly useful for computer-supported cooperative work. ACM Trans. on Office Information Systems *5* (2), 115-131 (1987)

Mark, L., Roussopoulos, N.: Metadata management. IEEE Computer (1986)

Meister, B., Janson, P., Svoboda, L.: File transfer in Local Area Networks: a performance study. Proc. of the 5th IEEE Intl. Conf. on Distributed Computer Systems, 1985, pp. 338-349

Mendelson, E.: Introduction to Mathematical Logic. Van Nostrand Company, New York 1964

Mumford, E.: Designing Human Systems. Manchester Business School, Manchester (1983)

Navathe, S., Kerschberg, L.: Role of data dictionaries in information resource management. Information and Management *10* (1986)

Newman, W.: Office models and office systems design. In Naffah N. (ed.): Integrated Office Systems. North-Holland, Amsterdam 1980, pp. 3-10

Nutt, G.J., Ricci, P.A.: Quinault: An office modeling system. Computer *14* (5), pp. 41-58 (1981)

Olle, T.W., Verrijn-Stuart, A.A, Bhabuta, L. (eds.): Computerized Assistance during the Information Systems Life Cycle. North-Holland, Amsterdam 1988

Ousterhout, J.K., Da Costa, H., Harrison, D., Kuntze, J.A., Kupfer, M., Thompson, J.G.: A trace-driven analysis of the UNIX 4.2BSD File System. Proc. of the 10th ACM Symp. on Operating Systems Principles, 1985, pp. 15-24

Panko, R.R.: 38 offices: analysing the needs in individual offices. ACM Trans. on Office Information Systems *2* (3), 226- (1984)

Pernici, B.: Supporting OIS design thorugh semantic queries. Proc. ACM-IEEE Conf. on Office Systems, Palo Alto, CA, March 1988, pp. 276-283

Pernici, B., Vogel, W.: An integrated approach to OIS development. In ESPRIT '86 - Results and Achievements, North-Holland, Amsterdam 1987, pp. 835-844

Pernici, B., Barbic, F., Fugini, M.G., Maiocchi, R., Pernici, B., Rames, J.R., Rolland, C.: C-TODOS: An automatic tool for office system conceptual design. ACM Trans. on Information Systems *7* (4), 378-419 (1989)

Potier, D.: New users' introduction to QNAP2. INRIA Technical Report No. 40, 1984

Proefrock, A.K., Tsichritzis, D., Mueller, G., Ader, M.: ITHACA: An integrated Toolkit for Highly Advanced Computer Applications. In Office and Business Systems Results and Progress of ESPRIT Projects in 1989, DG XIII, CEC (1989)

Ramamoorthy, C.V., Garg, V., Prakash, A.: Programming in the large. IEEE Trans. on Software Engineering *12* (7) (1986)

Reiter, R.: Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos, J.W. Schmidt (eds.): On Conceptual Modelling. Springer Verlag, New York 1984, pp. 191-233

Rolland, C., Proix, C.: An expert system approach to information system design. In H-J. Kugler (ed.): Information Processing '86. North-Holland, Amsterdam 1986, pp. 241-250

Rolland, C., Richard, C.: The Remora methodology for information systems design and management. In T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (eds.): Information Systems Development Mehodologies: a Comparative Review. North-Holland, Amsterdam 1982, pp. 369-426

Rolland, C., et al.: The RUBIS system. In (Olle et al. 1988), pp. 193-240

Schaefer et al.: Functional Analysis of Office Requirements. John Wiley and So. Publ., 1988

Schneider, H.J., Wasserman, A.I. (eds.): Automated Tools for Information Systems Design. North-Holland, Amsterdam 1982

Sirbu, M., Schoichet, S., Kunin, J., Hammer, M.: OAM: An office analysis methodology. Behaviour and Information Technology (1984)

Smith, J.M., Smith D.C.P.: Database abstractions: aggregation and generalisation. ACM Trans. on Database Systems, *2* (2), 105-133 (June 1977)

Sommerville, I.: Software Engineering. Addison Wesley Publ. Co., 3rd ed., Workingham 1989

Stalling, W.: Local Networks. ACM Computing Surveys *16* (1), (1984)

Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., Tatar, D.: WYSIWIS revised: Early experiences with multiuser interfaces. ACM Trans. on Office Information Systems *5* (2), 147-167 (1987)

Taylor, F.W.: Scientific Management, 1911

Teichroew, D., Hershey, E.A.: PSL/PSA: A computer- aided technique for structured documentation and analysis of information processing systems. IEEE Trans. on Software Engineering *3* (1) (1977)

Tsichritzis, D.: OFS: An integrated form management system. Proc. 6th Intl. Conf. on Very Large Data Bases, 1980, pp. 161-166

Tueni, M., Li, J., Lariviere, F., Juredini, G., et al.: The Activity Manager System environment. ESPRIT Project IWS Tech. Rep., May 1988

Veran, M., Potier, D.: QNAP2: A portable environment for queuing systems modelling. In D. Potier (ed.): Modelling Techniques and Tools for Performance Analysis, North-Holland, 1985, pp. 5-24

Wasserman, A.I.: Automated tools for information system design. In H.-J. Schneider, A.I. Wasserman (eds.): Automated Tools for Information System Design. North-Holland, Amsterdam 1982, pp. 1-10

Woods, W.: What's in a link: Foundations for semantic networks. In D.G. Bobrow and A.M. Collins (eds.): Representation and Understanding. Academic Press, New York 1975

Yau, S.S., Tsai, J.J.-P.: A survey of software design techniques. IEEE Trans. on Software Engineering *12* (6) (1986)

Zimmermann, H.: OSI Reference Model - The ISO model of architecture for Open System Interconnection. IEEE Trans. on Communications *COM-28* (4), 425-432 (1980)

Zisman, M.D.: Representation, Specification and Automation of Office Procedures. University of Pennsylvania, in Dissertation Abstracts International *38* (11), March 1978

Zloof, M.: Office-By-Example. IBM System Journal *21* (3), 272-304 (1982)

TODOS Technical Reports

Aksit, M., Bledoeg, E., van Dorsselaer, E., Heijmink, F., Meghini, C., Musto, D.: Architecture components analysis. ESPRIT TODOS Technical Report 4.1, 1987

Barbic, F., Fugini, M.G., Maiocchi, R., Pernici, B., Rames, J.R., Rolland, C.: TODOS conceptual model and specification language. ESPRIT TODOS Technical Report 2.2., issue A, December 1986

Bassanini, G., Di Stefano, F., Lunghi, G.: TODOS Analysis Model overview. ESPRIT TODOS Technical Report 1.2.2.1, December 1987

Bassanini, G., Lunghi, G., Temporin, M.: TODOS Analysis Model results and evaluation. ESPRIT TODOS Technical Report 1.3.5, 1988

Castelli, D., Meghini, C., Musto, D.: Architecture Specification Language: Design and Implementation. ESPRIT TODOS Technical Report 4.2, June 1988

Castelli, D., Meghini, C.: Application of the Mapping Methodology to the TODOS Test Case. ESPRIT TODOS Technical Report 4.3.4, December 1988

van Dorsselaer, E., Heijmink, F.: Performance evaluation within TODOS. ESPRIT TODOS Technical Report 4.2.6, September 1988

van Dorsselaer, E., Heijmink, F., Peters, H.: Modular construction of performance models for office information systems. ESPRIT TODOS Technical Report 4.2.5, 1988

Fugini, M.G., Maiocchi, R., Pernici, Possi, S., Stanga, M., Zecca, U.: Specification Database Design: the C-TODOS tool. ESPRIT TODOS Technical Report 2.3.1, April 1988

Heijmink, F., Meghini, C., Bledoeg, E., van Dorsselaer, E., Aksit, M., Musto, D.: Architecture Components Analysis. ESPRIT TODOS Technical Report 4.1, March 1987

Heijmink, F.: Definition of Office Primitives. ESPRIT TODOS Technical Report 4.3.1, September 1987

Henry, P.: Requirements Definition. ESPRIT TODOS Technical Report 1.1.2.6, May 1987

Henry, P.: Overview of the TODOS Structuring Tool. ESPRIT TODOS Technical Report 2.1.1, September 1987

Henry, P., Malherbe, H.: Overview of the TODOS Performance Model. ESPRIT TODOS Technical Report 1.2.3.1, 1988

Kieback, A.: Development of Basic Office Primitives. ESPRIT TODOS Technical Report 3.4, September 1987

Kieback, A., Mischke, G.: Languages and tools for rapid prototyping of OIS. ESPRIT TODOS Technical Report 3.2, 1987

Meghini, C., Musto, D.: Architecture Specification Language. ESPRIT TODOS Technical Report 4.2.1, January 1987

Musto, D.: Evaluation of TODOS Conceptual Model and study of mapping techniques. ESPRIT TODOS Technical Report 4.3.3, December 1988

Rames, J.R., Rolland, C.: G-TODOS: the C-TODOS graphical interface. ESPRIT TODOS Report 2.4.1, Nov. 1987

Wolfram, G., Pulst, E., TODOS Test Case Definition, ESPRIT TODOS Technical Report 5.5.1/5.5.2, July 1988

Wolfram, G., Pulst, E.: Test Case Execution. ESPRIT TODOS Technical Report 5.5.2, Dec. 1988

Glossary

AGEM	Architecture GEneration Methodology
ASL	Architecture Specification Language
ASPES	Architecture SPEcification System
ASL	Architecture Specification Language
AST	Architecture Specification Theory
LAN	Local Area Network
ODD	Office Data Dictionary
OIS	Office Information System
PE	Performance Evaluation
SDB	Specification Database
TAM	TODOS Analysis Model
TCM	TODOS Conceptual Model
TODOS	Automatic TOols for Designing Office Systems
TODQueL	TODOS Query Language
TSL	TODOS Specification Language

Authors' Address List

Gianluca Bassanini
Italtel
piazzale Zavattari, 12
I-20149 Milano MI, Italy
present address: Data General S.p.A., Via Mecenate 90, Milano

Donatella Castelli
Istituto di Elaborazione della Informazione -
Consiglio Nazionale delle Ricerche,
via S. Maria 46,
I-50126 Pisa PI, Italy

Fabio Di Stefano
Systems and Management
via Zezon 5
I-20124 Milano MI, Italy

Etienne L.M.E. Van Dorsselaer
Océ - Nederland B.V.
st. Urbanusweg 43, Venlo Holland
P.O.Box 101
NL-5900 MA Venlo - The Netherlands

Mariagrazia Fugini
Università di Brescia
Politecnico di Milano,
piazza Leonardo da Vinci, 32
I-20133 Milano MI, Italy

Frank J.M. Heijmink
Océ - Nederland B.V.
st. Urbanusweg 43, Venlo Holland
P.O.Box 101
NL-5900 MA Venlo - The Netherlands

Pascal Henry
Sema Group
56 rue Salengro
F-94126 Fontenay Sous Bois
present address: Bull Ingénierie, Rue de Paris 12, F-78230 LE PECQ. BP 59

Antoinette Kieback
Dornier GmbH
Postfach 1420
D-7990 Friedrichafen 1 - West Germany

Giancarlo Lunghi
Italtel
piazzale Zavattari, 12
I-20149 Milano MI, Italy

G. Mader
Dornier GmbH
Postfach 1420
D-7990 Friedrichafen 1 - West Germany

Carlo Meghini
Istituto di Elaborazione della Informazione -
Consiglio Nazionale delle Ricerche,
via S. Maria 46,
I-56126 Pisa PI, Italy

Daniela Musto
Istituto di Elaborazione della Informazione -
Consiglio Nazionale delle Ricerche,
via S. Maria 46,
I-56126 Pisa PI, Italy

Barbara Pernici
Politecnico di Milano,
piazza Leonardo da Vinci, 32
I-20133 Milano MI, Italy

Silvano Pozzi
Politecnico di Milano,
piazza Leonardo da Vinci, 32
I-20133 Milano MI, Italy
CEFRIEL, via Emanueli 15, Milano, Italy

Edda Pulst
BIFOA
Universitatstr. 45
D-5000 Koln 41 - West Germany

Jean Renè Rames
Thom'6 - Telesysteme
33, rue de Vouillé
F-75015 Paris, France
present address: ITN Consultants, 43 rue de Chateaudem, F-75009 Paris, France

Colette Rolland
Universite' de Paris I - Sorbonne
17, rue de la Sorbonne
F-75231 Paris CEDEX 05, France

Andrè Vignaud
Université de Paris 6 - P. et M. Curie
present address: 7, rue Edmond Goudinet, F-75013 Paris, France

Gerd Wolfram
BIFOA
Universitastr. 45
D-5000 Koln 41 - West Germany
present address: Kaufhof-Holding, Leonard-Tietz-Str. 1, D-5 Koeln 1