Antonia Albani
Jan L.G. Dietz (Eds.)

LNBIP 49

Advances in Enterprise Engineering IV

6th International Workshop, CIAO! 2010
held at DESRIST 2010
St. Gallen, Switzerland, June 2010, Proceedings

Springer

Lecture Notes
in Business Information Processing 49

Antonia Albani Jan L.G. Dietz (Eds.)

Advances in Enterprise Engineering IV

6th International Workshop, CIAO! 2010
held at DESRIST 2010
St. Gallen, Switzerland, June 4-5, 2010
Proceedings

Springer

Volume Editors

Antonia Albani
University of St. Gallen
Müller-Friedberg-Strasse 8
9000 St. Gallen, Switzerland
E-mail: antonia.albani@unisg.ch
and
Delft University of Technology
Mekelweg 4
2628 CD Delft, The Netherlands
E-mail: a.albani@tudelft.nl

Jan L.G. Dietz
Delft University of Technology
Mekelweg 4, 2628 CD Delft
The Netherlands
E-mail: j.l.g.dietz@tudelft.nl

Library of Congress Control Number: 2010926924

ACM Computing Classification (1998): J.1, H.3.5, H.4.1, D.2

ISSN 1865-1348
ISBN-10 3-642-13047-X Springer Berlin Heidelberg New York
ISBN-13 978-3-642-13047-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180 5 4 3 2 1 0

Preface

Enterprise engineering is an emerging discipline that studies enterprises from an engineering perspective. This means that enterprises are considered to be purposefully designed and implemented systems, which consequently can be re-designed and re-implemented if there is a need for change. Behavioral and managerial knowledge is perfectly adequate to identify the need for a change, but it is insufficient to bring it about. Next, enterprise engineering is rooted in both the organizational sciences and the information system sciences. The rigorous integration of these traditionally disjoint scientific areas has become possible following the recognition that communication is a form of action. Since then it has been quite common to speak of communicative acts, like requesting and promising. Consequently, communication (and information) is given an organizational interpretation: requests and promises are commitments, and communication is entering into and complying with commitments. This important insight clarifies the fact that enterprises belong to the category of social systems, i.e., their active elements (actors) are social individuals (human beings). Founding itself on this new scientific paradigm, enterprise engineering addresses the challenges that enterprises are currently faced with, both the internal and the external ones. The unifying role of human beings makes it possible to address problems in a holistic way, to achieve unity and integration in bringing about organizational change. This has not been shown before.

The development of such an innovative approach, as enterprise engineering is, requires the active involvement of a variety of research institutes and a tight collaboration between them. This is achieved by a continuously expanding network of universities and companies, called the CIAO! Network (www.ciaonetwork.org). Since 2005 this network has organized the annual CIAO! workshop, and since 2008 its proceedings have been published as "Advances in Enterprise Engineering" within the Springer LNBIP series. The book you are going to read contains the proceedings of the CIAO! Workshop 2010, which was held in conjunction with the DESRIST 2010 conference in St. Gallen, Switzerland.

June 2010

Antonia Albani
Jan L.G. Dietz

An Introduction to Enterprise Engineering

The Paradigm Shift

Enterprise engineering is an emerging discipline that studies enterprises from an engineering perspective. The first paradigm of this discipline is that enterprises are purposefully designed and implemented systems. Consequently, they can be re-designed and re-implemented, if there is a need for change. All kinds of changes are accommodated: strategic, tactical, operational, and technological. The second paradigm of enterprise engineering is that enterprises are social systems. This means that the system elements are social individuals, and that the essence of an enterprise's operation lies in the entering into and complying with commitments between these social individuals[1].

The Theoretical Roots

Enterprise engineering is rooted in both the organizational sciences and the information system sciences. Three concepts are already paramount to the theoretical and practical pursuit of enterprise engineering: enterprise ontology, enterprise architecture, and enterprise governance. *Enterprise ontology* concerns the understanding of an enterprise in a way that is fully independent of any implementation. The (one and only) ontological model of an enterprise shows the essence of its operation. It is the starting point for designing and implementing all kinds of changes. It is also extremely stable over time; most changes appear to be changes in the implementation. *Enterprise architecture* concerns the identification, the specification, and the application of design principles, which come in addition to the specific requirements of every change project. Design principles are the operational shape of an enterprise's strategic basis (mission, vision). Only in this way can one achieve and guarantee that the operations of an enterprise are fully compliant with its mission and strategies. Lastly, *enterprise governance* constitutes the organizational conditions for incorporating enterprise ontology and enterprise architecture in an enterprise's practice. It constitutes the primary condition for making the enterprise engineering approach feasible and beneficial.

The Current Evidence

The vast majority of strategic initiatives fail, meaning that enterprises are unable to gain success from their strategy. The high failure rates are reported from

[1] Basically and principally, only humans can take the role of social individual. We do recognize, however, the increasing belief among researchers that in the future artifacts could also take this role.

various domains: total quality management, business process reengineering, six sigma, lean production, e-business, customer relationship management, as well as from mergers and acquisitions. It appears that these failures are mostly the avoidable result of an inadequate implementation of the strategy. Rarely are they the inevitable consequence of a poor strategy. Abundant research indicates that the key reason for strategic failures is the lack of coherence and consistency, collectively also called congruence, among the various components of an enterprise. At the same time, the need to operate as an integrated whole is becoming increasingly important. Globalization, the removal of trade barriers, deregulation, etc., have led to networks of cooperating enterprises on a large scale, enabled by the virtually unlimited possibilities of modern information and communication technology. Future enterprises will therefore have to operate in an ever more dynamic and global environment. They need to be more agile, more adaptive, and more transparent. In addition, they will be held more publicly accountable for every effect they produce. These challenges are traditionally addressed by black-box-thinking-based knowledge, i.e., knowledge concerning the function and the behavior of enterprises, as contained in the organizational sciences. Such knowledge is sufficient, and perfectly adequate, for managing an enterprise (within the range of control). However, it is definitely inadequate for changing an enterprise. In order to bring about changes, white-box-based knowledge is needed, i.e., knowledge concerning the construction and the operation of enterprises. Developing and applying such knowledge requires no less than a paradigm shift in our thinking about enterprises, since the organizational sciences are dominantly oriented towards organizational behavior, based on black-box thinking.

The Evolutionary Milestones

The current situation in the organizational sciences resembles very much the one that existed in the information systems sciences around 1970. At that time, a revolution took place in the way people conceived information technology and its applications. Since then, people have been aware of the distinction between the *form* and the *content* of information. This revolution marks the transition from the era of data systems engineering to the era of information systems engineering. The comparison we draw with the information systems sciences is not an arbitrary one. On the one hand, the key enabling technology for shaping future enterprises is modern information and communication technology (ICT). On the other hand, there is a growing insight in the information systems sciences that the central notion for understanding profoundly the relationship between organization and ICT is the entering into and complying with commitments between social individuals. These commitments are raised in communication, through the so-called *intention* of communicative acts. Examples of intentions are requesting, promising, stating, and accepting. Therefore, as the content of communication was put on top of its form in the 1970s, the intention of communication is now put on top of its content. It explains and clarifies the organizational notions of collaboration and cooperation, as well as authority and responsibility. It also

puts organizations definitely in the category of social systems, very distinct from information systems. Said revolution in the information systems sciences marks the transition from the era of information systems engineering to the era of enterprise engineering, while at the same time merging with relevant parts of the organizational sciences, as illustrated in Fig. 1.

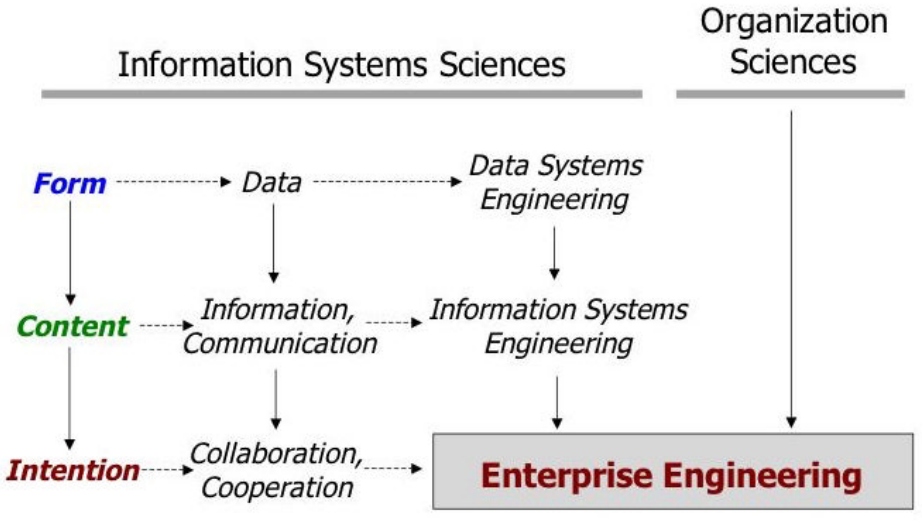

Fig. 1. Enterprise Engineering

The mission of the discipline of enterprise engineering is to combine (relevant parts from) the organizational sciences and the information systems sciences, and to develop theories and methodologies for the analysis, design, and implementation of future enterprises. Two crucial concepts have already emerged that are considered paramount for accomplishing this mission: enterprise ontology and enterprise architecture. A precondition for incorporating these methodologies effectively in an enterprise is the establishment of enterprise governance.

Theoretically, *enterprise ontology* is the understanding of an enterprise's construction and operation in a fully implementation independent way. Practically, it is the highest-level constructional model of an enterprise, the implementation model being the lowest one[2]. Compared to its implementation model, the ontological model offers a reduction of complexity of well over 90%. Only by applying this notion of enterprise ontology can substantial changes of enterprises be made intellectually manageable.

Theoretically, *enterprise architecture* is the normative restriction of design freedom. Practically, it is a coherent and consistent set of principles that guides

[2] Dietz, J.L.G., Enterprise Ontology – Theory and Methodology, Springer, 2006, ISBN 978-3-540-29169-5

the (re)design and (re)implementation of an enterprise, and that comes in addition to the specific requirements of a change project[3]. These principles are derived from the enterprise's strategic basis (mission, vision). Only by applying this notion of enterprise architecture can consistency be achieved between the strategic basis and the operational business rules of an enterprise.

Enterprise governance is the organizational competence for continuously exercising guiding authority over enterprise strategy and architecture development, and the subsequent design, implementation, and operation of the enterprise[4]. Adopting this notion of enterprise governance enables an enterprise to be compliant with external and internal rules, and to perform in an optimal and societally responsible way.

Modeling and Simulation

Every time that a change happens in the business environment or a change is required due to certain circumstances, it results in analysis and design of some aspects of the enterprise (organization, business processes, supporting technology, etc.). Current trends in business process management show that processes-oriented approaches are receiving increasing attention in analyzing and designing enterprises and implementing innovations addressing the external forces (customers, competitors, environment, etc.). As the very core of process innovation is change, and changes always need to be evaluated in comparison with different scenarios and situations, this demands an even more integral role of modeling and simulation in the design, redesign, and process improvement activities of enterprise engineering. Obviously any change is risky and may have serious consequences for enterprises. Early mitigation of risks associated with redesign and innovation is highly important, especially in situations with many uncertainties. Here is where modeling and simulation play an enormous role in the analysis, design, redesign, comparison of alternatives, and measurement of the effects of changes[5].

Ontology-Based Development of Information Systems

Based on the notion of enterprise engineering, new modeling methodologies are needed to cope with the specific aspects of an enterprise as a designed and engineered artifact. Such methodologies should not only comprise methods and models to design the enterprise in order to understand and change it, but also

[3] Hoogervorst, J.A.P., Dietz, J.L.G.: Enterprise Architecture in Enterprise Engineering. In: Enterprise Modeling and Information Systems Architecture, Vol. 3, No. 1, July 2008, pp 3-11, ISSN 1860-6059

[4] Hoogervorst, J.A.P., Enterprise Governance and Enterprise Architecture, Springer, 2009, ISBN 978-3-540-92670-2

[5] Barjis, J. (2007). Automatic Business Process Analysis and Simulation Based on DEMO. Journal of Enterprise Information Systems, Vol. 1, No. 4, pp. 365-381

to design and implement information systems supporting the operations and decision makings of such enterprises. Several enterprise modeling methodologies exist and are widely applied in practice today. But most of them are not based on a well-founded theory that integrates the notion of construction and operation of the enterprise in a fully implementation-independent way. Said approaches therefore result in unnecessarily complex, unstable, and unwieldy models including not only the essential features of an enterprise. The same holds for the models of the supporting information systems, which are based on those enterprise models. In order to provide valuable information to business people who make decisions about requirements, use the solutions and make decisions about future strategies, both the enterprise models and the supporting information system models need to be provided on a high level of abstraction. Therefore, there is a need for new and innovative methodologies applying the notion of enterprise ontology, and for new methods transforming such ontological models into information system models[6]. The resulting information system models have a reference character. That means that they are stable since they are based on ontological models, which are completely implementation independent. A business domain is not going to change often, but the implementation of that business domain may change easily.

June 2010

Jan L.G. Dietz
Antonia Albani
Joseph Barjis

[6] Albani, A., Dietz, J., 2008. Software and Data Technologies, Second International Conference, ICSOFT/ENASE 2007, Barcelona, Spain, July 22-25, 2007, Revised Selected Papers. Vol. 22. Springer Verlag, Ch. Benefits of Enterprise Ontology for the Development of ICT-Based Value Networks, pp. 322.

Organization

The CIAO! workshop is organized annually as an international forum for researchers and practitioners in the general field of enterprise engineering. Organization of the workshop and peer review of the contributions made to this workshop are accomplished by an outstanding international team of experts in the fields of enterprise engineering.

Workshop Chairs

Antonia Albani	University of St. Gallen (Switzerland) and Delft University of Technology (The Netherlands)
Jan L.G. Dietz	Delft University of Technology (The Netherlands)

Program Committee

Wil van der Aalst
Eduard Babkin
Joseph Barjis
Bernhard Bauer
Emmanuel delaHostria
Johann Eder
Joaquim Filipe
Rony G. Flatscher
Birgit Hofreiter
Jan Hoogervorst
Stijn Hoppenbrouwers
Christian Huemer
Peter Loos
Graham Mcleod
Aldo de Moor
Hans Mulder
Nikolaus Müssigmann
Moira Norrie
Martin Op 't Land
Erik Proper
Gil Regev
Pnina Soffer
Pedro Sousa
José Tribolet
Jan Verelst

Table of Contents

Enterprise Ontology

Organizational Modeling

System Development

Aligning the Constructs of Enterprise Ontology and Normalized Systems

Philip Huysmans, David Bellens, Dieter Van Nuffel, and Kris Ven

Department of Management Information Systems,
University of Antwerp, Antwerp, Belgium
{philip.huysmans,david.bellens,dieter.vannuffel,kris.ven}@ua.ac.be

Abstract. Literature suggests that, due to their complexity, organizations need to be designed in order to be effective and evolvable. Recently, two promising approaches have been introduced that are relevant in this regard. Enterprise Ontology creates essential models that are implementation-independent. Normalized Systems is concerned with the development of information systems with proven evolvability. In this paper, we combine both approaches. To this end, we express the transaction pattern—a central construct of Enterprise Ontology—using the constructs of Normalized Systems. By aligning these constructs, we attempt to introduce traceability between the Enterprise Ontology level and the Normalized Systems level. The resulting artefact exhibits the benefits of both Enterprise Ontology and Normalized Systems. We illustrate the application of the artefact in the context of enterprise architectures.

Keywords: Enterprise Ontology, Normalized Systems, Enterprise Architecture.

1 Introduction

Contemporary organizations have to be agile in order to be able to adapt to changing market environments. A change of the organization as a whole affects many different organizational elements. Given the complexity of organizations, it can be argued that organizations should be *designed* in order to exhibit true agility [7]. Enterprise architecture is proposed as a way to control this complexity. Despite the multitude of frameworks available, no common scientific or theoretical foundation seems to be agreed upon. Therefore, it is difficult to compare and evaluate the recommendations made by these frameworks [10]. In this paper, we explore an approach which focuses on the organizational ability to change. We base our approach on the systems theoretic concept of evolvability by applying the theorems of Normalized Systems. By adhering to the four theorems of Normalized Systems during software design and development, software architectures of proven evolvability are obtained [11]. Based on these theorems, Normalized Systems proposes five software elements to design the modular structure of software. This modular structure ensures that the software is free from so-called combinatorial effects. Enterprise Ontology provides abstract, implementation-independent organizational

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 1–15, 2010.

constructs which can describe a broad organizational scope with few construct instantiations (i.e., transactions) [2]. In order to work towards evolvable organizations, we explore an implementation of Enterprise Ontology transactions which are evolvable. Normalized Systems suggests that we therefore need an implementation which is free of combinatorial effects. Different alternatives seem to be available to reach this goal. A first alternative would be to define an implementation which closely mimics the structure of the transaction pattern, and ensure that this implementation is free of combinatorial effects. While a complete specification of the implementation is not possible on this abstraction level, we could base further specification upon this implementation. A second alternative would be to implement a completely specified Enterprise Ontology model, in which we could then eliminate the combinatorial effects. In this paper, we explore the first alternative. More specific, we explore the expression of the transaction pattern—a core Enterprise Ontology construct—in Normalized Systems elements. While similar, more practice-driven approaches may exist, we limit ourselves to the combination of Enterprise Ontology and Normalized Systems, because of their scientific foundation.

The rest of the paper is structured as follows. In Section 2, we introduce Normalized Systems and Enterprise Ontology. We then describe the alignment of the constructs in Section 3. Next, We position our approach within enterprise architectures in Section 4. Finally, we offer our conclusions in Section 5.

2 Scientific Foundations

In this section, we provide a brief introduction to both Enterprise Ontology and Normalized Systems. We primarily focus on the constructs of both approaches that were used in our research.

2.1 Enterprise Ontology

Theoretical Foundation. In order to grasp the complexity of organizations, models can be constructed. These models abstract away from the information that is available in the real world. Depending on the way of abstraction, very diverse models can be made. Enterprise Ontology views the organization as a social system [2]. Therefore, it is well suited to describe the interaction between an organization and its environment. Enterprise Ontology assumes that communication between human actors is a necessary and sufficient basis for a theory of organizations [2]. This is based on the language action perspective and Habermas theory of communicative action. The strong theoretical foundation ensures a consistent modelling methodology. Clear guidelines are provided to create abstract models. Since only the *ontological* acts are represented in the models, the same model will be created for organizations who perform the same function, but operate differently. For example, consider the BPR case at Ford [6]. The ontological model of the processes of the situation before and after reengineering are identical. Because of the focus on the essential business processes, Enterprise Ontology models can be very concise. Therefore, they provide a good overview

of a broad enterprise scope. Many case studies are reported where large organizations are described with few modelling artefacts (e.g. [13]).

In this paper, we will focus on the transaction pattern as the basic construct of Enterprise Ontology. We currently focus on the transaction as the main Enterprise Ontology primitive for two main reasons. First, the transaction pattern is a core element of the Enterprise Ontology theory. The transaction pattern is specified by the transaction axiom, the second axiom from the Ψ-theory on which Enterprise Ontology is based. Second, it is the basis on which other models elaborate. It therefore seems logical to base our initial effort on achieving a correct mapping of the transaction pattern.

The transaction pattern evolved thanks to contributions from many researchers [1,5,15,17]. The transaction pattern describes the coordination necessary to produce a certain result. This result is represented by a *production fact*. There are always two actors involved in a transaction: the *initiator* actor who wants to achieve the fact, and the *executor* actor who performs the necessary actions to create the fact. Delivering a product, performing a service or subscribing to an insurance are examples of production facts which could be created by completing a transaction.

Enterprise Ontology Artefact. The high-level structure of the transaction pattern consists of three phases. In the order phase, the actors negotiate the subject of the transaction. In the execute phase, the subject of the transaction is brought about. In the result phase, the result of the transaction is presented and accepted. In different versions of the transaction pattern, different ontological process steps are identified in the three phases. These steps are called *coordination acts*. The successful completion of an act results in a *coordination fact*. Enterprise Ontology distinguishes between the basic, standard and complete transaction pattern.

The graphical representation of the transaction pattern is shown in Figure 1. The combination of a coordination act and fact is represented by a circle in a square. The combination of a production act and fact is represented by a diamond in a square. Small circles represent process entry points, and small circles with a cross represent a choice between alternate flows. Light-grey boxes indicate which acts fall under the authority of a certain actor. In Figure 1, A01 is the initiator actor, and A02 is the executor actor.

Basic Transaction Pattern: The basic transaction pattern consists of the five standard acts which occur in a successful scenario (i.e., request, promise, execute, state and accept) [2, p. 90]. These five acts are shown in the centre of Figure 1. In the order-phase, the initiator actor first *requests* the creation of a fact. The executor actor then *promises* to fulfil this request. In the execute-phase, the executor actually performs the necessary actions to create the fact in the *execute* act. In the result phase, the executor first *states* the successful completion of the fact. Finally, the initiator *accepts* this statement. Consider this transaction in the case of a simple product delivery process. In a first process step, the customer requests the product. Once this request is adequately specified, the request coordination fact is created. Second, the supplier promises to deliver the product according to the agreed terms. This creates the promise coordination fact. The third process

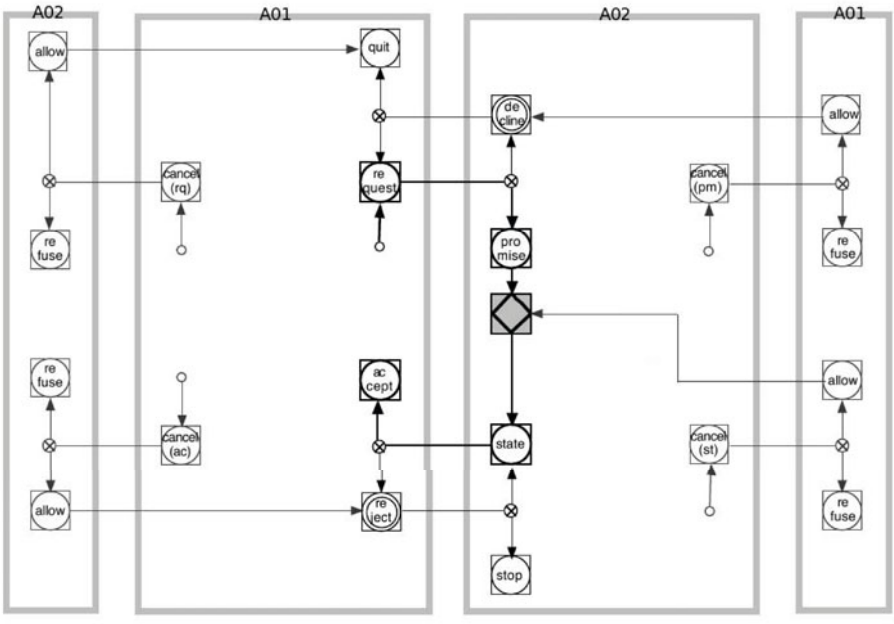

Fig. 1. Graphical representation of the Enterprise Ontology Transaction Pattern

step is the actual delivery. This results in the production fact "Product X has been delivered". In the fourth process step, the supplier states that the delivery has been completed. If the customer is satisfied with the delivery, he will accept the delivery in the fifth process step. Once the accept coordination fact is created, the transaction is considered to be completed.

Standard Transaction Pattern: The *standard* transaction pattern is the basic transaction pattern, augmented with the scenario in which the actors dissent [2, p. 93]. The coordination facts which indicate a dissent are represented by a double circle in Figure 1 (i.e., decline and reject facts). In the order-phase, the executor actor can decline the incoming request of the initiator actor. The initiator then has to decide whether he resubmits his request, or quits the transaction. In our example, the supplier could decline the delivery of a product which does not belong to his catalogue. The customer would need to select another product, or quit the transaction and search another supplier. The execute-phase is identical to the execute-phase in the basic transaction pattern. In the result-phase, the initiator actor can reject the stated production fact instead of accepting it. The executor then has to decide whether he wants to repeat the execution act and make the statement again, or stop the transaction.

Complete Transaction Pattern: In the complete transaction pattern, cancelation patterns are added to the standard transaction pattern. In Figure 1, the cancelation patterns are started in the four additional process entry points. According

to [2], every coordination fact can be cancelled at any time by the responsible actor. This cancelation can then be allowed or refused by the other actor. For example, when the customer changes his mind after requesting the delivery of a product, he can cancel his request. The executor then has to decide whether he allows this cancelation, in which case the transaction ends, or refuses the cancelation, and proceeds with the transaction.

2.2 Normalized Systems

Theoretical Foundation. The basic assumption of Normalized Systems is that information systems should be able to evolve over time, and should be designed to accommodate change. As this evolution due to changing business requirements is mostly situated during the mature life cycle stage of an information system, it takes the form of software maintenance. Software maintenance is considered to be the most expensive phase of the information system's life cycle, and often leads to an increase of architectural complexity and a decrease of software quality [4]. This phenomenon is also known as Lehman's law of increasing complexity, expressing the degradation of information system's structure over time [9]. Because changes applied to information systems are suffering from Lehman's law, the impact of a single change will increase over time as well [8]. Therefore to genuinely design information systems accommodating change, they should exhibit *stability* towards these requirements changes. In systems theory, stability refers to the fact that bounded input to a function results in bounded output values, even as $t \to \infty$. When applied to information systems, this implies that no change propagation effects should be present within the system. This means that a specific change to an information system should require the same effort, irrespective of the information system's size or point in time when being applied. *Combinatorial effects* occur when changes require increasing effort as the system grows; and should thus be avoided. Normalized systems are defined as information systems exhibiting stability with respect to a defined set of changes [11], and are as such defying Lehman's law of increasing complexity [8,9] and avoiding the occurrence of combinatorial effects. In this sense, evolvability is operationalized as a number of anticipated changes that occur to software systems during their life cycle [12].

The normalized systems approach deduces a set of four *design theorems* that act as design rules to identify and circumvent most combinatorial effects [11,12]. It needs to be emphasized that each of these theorems is not completely new, and even relates to the heuristic knowledge of developers. However, formulating this knowledge as theorems that identify these combinatorial effects aids to build systems containing minimal combinatorial effects. The first theorem, *separation of concerns*, implies that every change driver or concern should be separated from other concerns. This theorem allows for the isolation of the impact of each change driver. Parnas described this principle already in 1972 [14] as what was later called *design for change*. Applying the theorem prescribes that each module can contain only one submodular task (which is defined as a change driver), but also that workflow should be separated from functional submodular tasks.

The second theorem, *data version transparency*, implies that data should be communicated in version transparent ways between components. This requires that this data can be changed (e.g., additional data can be sent between components), without having an impact on the components and their interfaces. This theorem can, for example, be accomplished by appropriate and systematic use of web services instead of using binary transfer of parameters. This also implies that most external APIs cannot be used directly, since they use an enumeration of primitive data types in their interface.

The third theorem, *action version transparency*, implies that a component can be upgraded without impacting the calling components. This theorem can be accomplished by appropriate and systematic use of, for example, polymorphism or a facade pattern.

The fourth theorem, *separation of states*, implies that actions or steps in a workflow should be separated from each other in time by keeping state after every action or step. This suggests an asynchronous and stateful way of calling other components. Synchronous calls resulting in pipelines of objects calling other objects which are typical for object-oriented development result in combinatorial effects.

Normalized Systems Artefacts. The design theorems show that software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. Normalized Systems therefore proposes to encapsulate software constructs in a set of five higher-level software elements (i.e., data, action, flow, connector and trigger elements). These elements are modular structures that adhere to these design theorems, in order to provide the required stability with respect to the anticipated changes [11]. To map the Enterprise Ontology transaction pattern, three of these five higher-level software elements are needed. We will now elaborate on these three elements.

From the second and third theorem it can straightforwardly be deduced that the basic software constructs, i.e., data and actions, have to be encapsulated in their designated construct.

Data Element: A *data element* represents an encapsulated data construct with its get- and set-methods to provide access to their information in a data version transparent way. So-called cross-cutting concerns, for instance access control and persistency, should be added to the element in separate constructs.

Action Element: The second element, *action element*, contains a core action representing one and only one functional task. Arguments and parameters need to be encapsulated as separate data elements, and cross-cutting concerns like logging and remote access should be again added as separate constructs. [16] distinguish between four different implementations of an action element: *standard* actions, *manual* actions, *bridge* actions and *external* actions. In a standard action, the actual task is programmed in the action element and performed by the same information system. In a manual action, a human act is required to fulfil the task. The user then has to set the state of the life cycle data element

through a user interface, after the completion of the task. A process step can also require more complex behaviour. A single task in a workflow can be required to take care of other aspects, which are not the concern of that particular flow. Consider the ordering of parts for an assembly. The assembly workflow needs to know when the parts are ready to be assembled, but it is not concerned with how the parts are prepared. Therefore, a separate workflow will be created to handle the concerns of the individual parts. Bridge actions create these other data elements going through their designated flow. Fourth, when an existing, external application is already in use to perform the actions on, for instance, the different parts of an assembly, the action element would be implemented as an external action. These actions call other information systems and set their end state depending on the external systems' reported answer.

Workflow Element: Based upon the first and fourth theorem, workflow has to be separated from other action elements. These action elements must be isolated by intermediate states, and information systems have to react to states. A third element is thus a *workflow element* containing the sequence in which a number of action elements should be executed in order to fulfil a flow. A consequence of the stateful workflow elements is that state is required for every instance of use of an action element, and that the state therefore needs to be linked or be part of the instance of the data element serving as argument. We call this data element the life cycle data element of a flow. A graphical representation of a flow element is shown in Figure 2. This representation is consistent with the representation of Normalized Systems workflow elements, which are based on state machines [11, p. 143]. The black circles represent the different states of the flow, being the life cycles states of the corresponding data element. The state name is notated next to the state symbol. The squares represent the action elements.

3 Translating the Transaction Pattern

As discussed in Section 2.1, the transaction pattern of Enterprise Ontology is the starting point for our research. In this section, we present the mapping of the transaction pattern to the constructs of Normalized Systems, which are discussed in Section 2.2. We will start by translating the *basic* transaction pattern, and iteratively add more details in the *standard* transaction pattern and the *cancelation patterns*.

3.1 The Basic Transaction Pattern

We start by mapping the basic transaction pattern. The basic transaction pattern consists of the process steps request, promise, execute, state and accept. In Normalized Systems, this transaction pattern process is represented by a *flow* element. A flow element is driven by precisely one data element, the life cycle data element. Consider a transaction T01. In order to define a Normalized Systems flow, we thus need a `T01` data element. The completion of the different acts

in the transaction process is represented by the creation of ontological facts. In Normalized Systems, these facts are represented by the states which occur in the flow element, being the life cycle states of the corresponding data element. To reach these states, a state transition is required. A state transition is realized by an action element. The successful completion of that action element results in the defined life cycle state. In order to define the control flow of the process, we therefore need to specify the *trigger states*, *state transitions* and *transaction actions*. Regarding the request coordination fact, this implies that the T01 flow element, and thus also the corresponding T01 data element, should reach the state *Requested*. This means that upon initiation of a T01 transaction, a new T01 data element is instantiated trough its default constructor, resulting in the life cycle state *Initial*. The genuine act of requesting is encapsulated in the action element `Request`. The *concerns* of creating the data element and handling the request are separated as they can clearly evolve independently from each other. The request could, for example, contain additional information that needs to be processed. Since we are currently only regarding the successful flow of the transaction, we do not yet need any branching. The state transition can be expected to always result in the end state *Requested*. The resulting Normalized Systems flow is shown in Figure 2, and schematically represented in Table 1.

While all state transitions are defined as action elements, their different nature can mean that they need to be implemented differently. Consider the notification of the initiator actor in the promise process step. If this notification requires a human action, e.g., a manager who has to decide, the `Promise` action element would be implemented as a manual action. However, the promise process step

Table 1. Specification of the basic transaction pattern flow element

Workflow name		Basic Transaction Pattern	
Data element		T01-basic	
Start state	**Action name**	**End state**	**Failed state**
Initial	Request	Requested	
Requested	Promise	Promised	
Promised	Execute	Executed	
Executed	State	Stated	
Stated	Accept	Accepted	

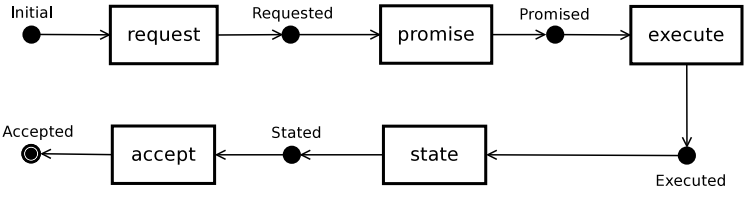

Fig. 2. Graphical representation of the basic transaction pattern flow

can also require more complex behaviour. When for example the product first needs to be reserved in the warehouse, the `Promise` action element would be implemented as a bridge action triggering a flow element on another data element, e.g., a `Part` element. When an existing application is already in use to perform these reservations, the `Promise` action element would be implemented as an external action.

3.2 The Standard Transaction Pattern

The standard transaction pattern adds the scenario in which the actors can dissent. When translating these additions to Normalized Systems primitives, some additional actions and states have to be included due to the Normalized Systems theorems. The resulting Normalized Systems flow element is graphically represented in Figure 3. Based on *separation of concerns*, the decision of the executor actor to promise or decline the request needs to be separated from the actual coordination act (i.e., the communication of the decision). The communication method can change independently, as shown by the various implementations of the `Promise` action element in the basic transaction pattern. Since the decision logic to promise or decline can also change independently of the communication method, these two actions should not be combined in one action element. Doing so would introduce a combinatorial effect. Therefore, we introduce an additional action element `ValidateRequest`. In the case where the executor decides to handle the request, the state *RequestValidated* is set. Otherwise, the state *RequestInvalidated* is set. The actual `Promise` action element remains identical to the action element described in the basic transaction pattern. If the request is however declined, the initiator actor needs to decide whether or not to resubmit the request. This decision logic is again separated from the other actions by encapsulating the decision logic in an action element `ValidateDecline`. If the initiator decides to resubmit, the state is set to *DeclineValidated*. The `Resubmit` action element then allows the initiator actor to possibly change the request and to resubmit it which will again result in the state *Requested*. If the initiator decides to abort the transaction, the state is set to *DeclineValidated*, which triggers the `Quit` action element to reach the end state *Quitted*.

Analogously, the initiator actor has to decide whether he accepts the stated production fact. We therefore introduce the `ValidateState` action element, which results in the *StateValidated* state in case of a successful acceptance, or in the *StateInvalidated* state in case of an unsuccessful one. The *StateValidated* state triggers the `Accept` action element, which contains the actual accept coordination act. In case the initiator does not accept the state coordination fact, the workflow is brought to the *Rejected* state through the `Reject` action element. The decision whether to handle the reject is taken in the `ValidateReject` action element. The reject handling itself is implemented as a dedicated `HandleReject` action element. If the executor does not handle the reject, the transaction reaches the end state *Stopped* through the `Stop` action element. All the described state transitions for the standard transaction pattern are summarized in Table 2.

Table 2. Specification of the standard transaction pattern flow element

Workflow name		Standard Transaction Pattern	
Data element		T01-standard	
Start state	**Action name**	**End state**	**Failed state**
Initial	Request	Requested	
Requested	ValidateRequest	RequestValidated	RequestInvalidated
RequestInvalidated	Decline	Declined	
RequestInvalidated	ValidateDecline	DeclineValidated	DeclineInvalidated
DeclineInvalidated	Quit	Quitted	
DeclineValidated	Resubmit	Requested	
RequestValidated	Promise	Promised	
Promised	Execute	Executed	
Executed	State	Stated	
Stated	ValidateState	StateValidated	StateInvalidated
StateInvalidated	Reject	Rejected	
Rejected	ValidateReject	RejectValidated	RejectInvalidated
RejectInvalidated	Stop	Stopped	
RejectValidated	HandleReject	Stated	
StateValidated	Accept	Accepted	

3.3 The Cancelation Patterns

The *complete* transaction pattern also includes the various cancelation patterns and is shown in Figure 1. A cancelation consists of two main issues: deciding whether or not to allow a cancel request and handling the cancelation itself. The first issue actually consists of initially receiving the cancel request, then deciding whether or not to allow the requested cancelation, and third potentially to notify the initiator of the rejected cancel request. As such, based on *separation of states* and *separation of concerns*, these three concerns will be separated. First, upon arrival of a cancel request, a dedicated `CancelRequest` data element will be created. This implies that for every life cycle data element that can be cancelled, a related `CancelRequest` data element instance will be created if such a request arrives. For example, for a life-cycle data element called `Order`, a corresponding `OrderCancelRequest` data element will be created. Second, an action element `AcceptCancelation` will implement the decision whether or not to accept. Third, in case of an rejected request, the initiator will probably have to be notified. This functionality is represented by a bridge action `Refuse` executing the notification in the way as discussed in [16]. In case of an allowed cancelation, the `CancelTransaction` standard action element will initiate the cancelation handling which will be explained next. The Normalized Systems specification for the workflow representing the cancel request issue is shown in Table 3 and Figure 4. In case of an allowed cancelation, `CancelTransaction` standard action element will initiate the handling itself explained hereafter.

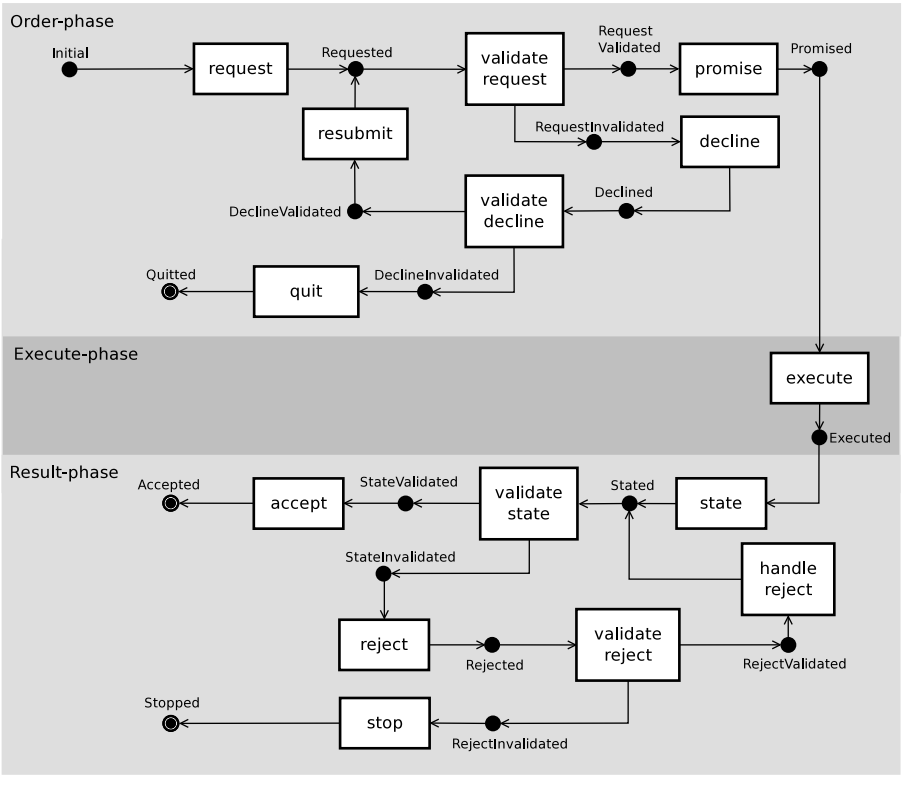

Fig. 3. Graphical representation of the standard transaction pattern flow

Table 3. Specification of the cancelation pattern flow element

Workflow name		Transaction Cancelation	
Data element		T01-CancelRequest	
Start state	**Action element**	**End state**	**Failed state**
Initial	CheckValidity	CancelRequestValid	
CancelRequestValid	AcceptCancelation	Allowed	not-Allowed
not-Allowed	Refuse	Refused	
Allowed	CancelTransaction	Canceled	

If the cancelation is allowed, it may be necessary to partly or completely roll back the transaction. Given the divergence of business contexts, a roll back can imply different actions given the state of the transactions. Therefore, the cancelation process will be designed using multiple scenarios implemented as separate action elements on the same life cycle data element. Consider the case where various parts are ordered to complete the assembly of a product. In case the parts have not yet been received, an order cancelation can be submitted

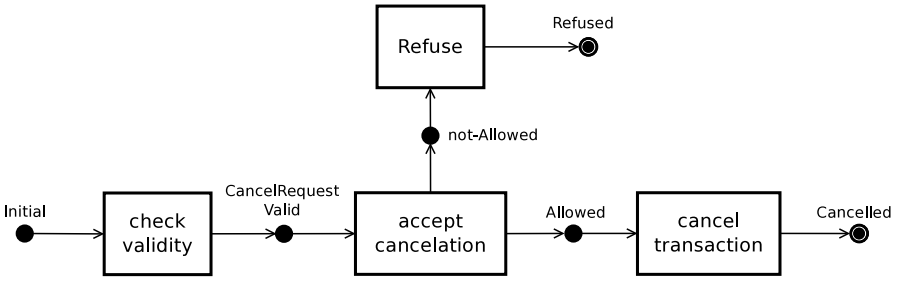

Fig. 4. Graphical representation of the cancelation pattern

to the parts supplier. In case the parts are already received and reserved, they should be released and made available for future assemblies. Thus, the scenario and constituent action elements are dependent on the life cycle data element's state when the cancelation request is initiated.

Since a cancelation can occur regardless of the current state of the transaction, it is modelled in the Enterprise Ontology transaction pattern as a separate entry point. However, the Normalized Systems theorems do not allow that the state of the main flow is simply altered by any other flow because a flow element actively interfering with another flow element is considered a so-called *GOTO statement*. In accordance with the seminal work of Dijkstra [3], Normalized Systems does not allow this kind of statements, and therefore prohibits such a direct state transition by another flow.

We outline the solution for adding cancelation patterns consistent with Normalized Systems theorems as described in [16]:

- A `cancelRequest` data attribute is added to the data element operating the flow.
- A cancel can be initiated in multiple ways. The particular situation should be assigned to the value of the `cancelRequest` data attribute by the `CancelTransaction` standard action element.
- The engine operating the respective flow element checks the `cancelRequest` data attribute. If this field is set, the current state of the flow will be saved in the so-called *parking state field*. The regular state field of the workflow will be set to "cancel requested".
- An action element will subsequently be triggered to decide which cancelation flow—i.e., sequence of action elements on the corresponding life cycle data element—has to be triggered as the cancelation scenario will differ according to the life cycle state as also illustrated by the cancelation patterns in Enterprise Ontology. Therefore, this action element will use the value of the so-called *parking state field*, uniquely describing the life cycle state of the corresponding data element when the cancel request was communicated.

This implies that a cancelation is handled as a sequence of action elements on the same life cycle data element. This is in line with the observation that requesting,

promising, executing, stating, declining, or cancelling a fact addresses the same concern. However, the sequence of actions about the cancel request itself are separated in their designated elements. It should be noted that we present a generic cancelation pattern. The possibility of triggering different cancelation flows, based on the value of the `cancelRequest` data attribute, allows us to implement the four different Enterprise Ontology cancelation patterns.

4 Application in Enterprise Architecture

In the previous section, we presented a translation of the Enterprise Ontology transaction pattern in Normalized Systems constructs. This artefact could be used in the context of enterprise architectures. We now outline the implication of our artefact in enterprise architectures as defined by Hoogervorst [7]. Hoogervorst proposes a method to design so-called construction models that enable the implementation of the implementation-independent Enterprise Ontology models. Based on the ontological models, four enterprise design domains (i.e., business, organization, information and technology) need to be designed. Enterprise architecture provides "the normative guidance for the design process" [7]. The architecture consists of principles, which have to be respected during the design of the construction models. These principles are the result of strategic choices. Therefore, organizations with identical ontological models can be implemented differently based on their different architectural principles, since different construction models will be designed. This is how organizations can differentiate from each other.

However, certain characteristics can be useful for any organization, such as evolvability. When the architecture needs to achieve such general strategic characteristics, architectural principles could be proposed which are more generally accepted. To achieve this, we need to know which principles affect the evolvability of construction models. According to Normalized Systems, the occurance of combinatorial effects affects evolvability. Principles which are analogous to the Normalized Systems theorems could thus affect the occurance of combinatorial effects—and therefore, evolvability—in contruction models. Such principles would need to guide the implementation of transactions to avoid combinatorial effects. Therefore, our implementation of the transaction pattern seems to fit the concept of enterprise architecture as intented by Hoogervorst: it guides the design of the transaction implementation by restricting design freedom, since only Normalized Systems elements can be used. Our artefact provides a basis for the further development of construction models which are free of combinatorial effects. The use of our construct is not limited to the design domain *information technology*. For example, designing the processes of the design domain *organization* based on our artefact enforces adherence to the Normalized Systems theorems, while respecting the integration between the processes which implement a certain transaction.

5 Discussion and Conclusions

This paper presents the first implementation of Enterprise Ontology transactions with explicit attention to combinatorial effects. It has two important contributions. First, our artefact shows that a mapping between Enterprise Ontology and Normalized Systems constructs is feasible. More specifically, it shows that such a mapping is feasible very early in the design process. Moreover, we presented a generic and systematic mapping. While it is possible that the mapping artefact needs to be refined or adapted, it can be used for the implementation of any transaction. This means that our artefact can be used as a starting point for designing evolvable organizations. We further illustrated this point by suggesting the use of the artefact within enterprise architectures. While our implementation remains at an abstract level, further specification of construction models can be guided by existing research, both scientific and practical. On the scientific level, Normalized Systems has proven to prevent combinatorial effects in software implementations. On the practical level, large-scale mission-critical systems are already developed using Normalized Systems elements.

Second, our mapping shows that the Normalized Systems theorems do impact the implementation of Enterprise Ontology models, when combinatorial effects need to be avoided. In order to implement transactions which are free of combinatorial effects, several guidelines can be prescribed:

- Additional state transitions need to be created in order to comply with the separation of concerns and separation of state theorems. We introduced these state transitions during the mapping of the standard transaction pattern.
- Based on previous research, we propose an implementation of the cancelation patterns which enables an implementation of different roll-back scenarios and adhering to the Normalized Systems theorems.

Following these guidelines will not affect the Enterprise Ontology models itself, since they are implementation-independent. The occurrence of combinatorial effects during changes will only affect the actual implementation of the Enterprise Ontology models. While we do not claim to have removed all combinatorial effects in our implementation, we achieved an effective and efficient mapping method by specifying these guidelines early in the design process. Effective, because the use of Normalized Systems elements implies the adherence to architectural principles. Efficient, because combinatorial effects are prevented instead of removed.

The presented approach suggests following future research subjects. First, we presented the mapping of a single transaction. Obviously, the construction of an organization implies the integration of several transactions. In subsequent research, we will focus on an approach to integrate different transactions, while respecting the Normalized Systems theorems. While the current mapping is mainly influenced by the separation of concerns and separation of states theorems, it can be expected that guidelines for integration will need to focus on the data and action version transparency theorems. Second, the focus of this paper was on the

conceptual mapping of constructs. In following publications, we will report on the applications of our artefact in various cases.

References

1. Auramäki, E., Hirschheim, R., Lyytinen, K.: Modelling offices through discourse analysis: the sampo approach. Computer Journal 35(4), 342–352 (1992)
2. Dietz, J.L.: Enterprise Ontology: Theory and Methodology. Springer, Berlin (2006)
3. Dijkstra, E.: Go to statement considered harmful. Communications of the ACM 11(3), 147–148 (1968)
4. Eick, S.G., Graves, T.L., Karr, A.F., Marron, J., Mockus, A.: Does code decay? assessing the evidence from change management data. IEEE Transactions on Software Engineering 27(1), 1–12 (2001)
5. Goldkuhl, G.: Generic business frameworks and action modeling. In: Proceedings of the Conference on Communication Modeling—Language/Action Perspective 1996, Springer, Heidelberg (1996)
6. Hammer, M.: Reengineering work: Don't automate, obliterate. Harvard Business Review 68(4), 104 (1990)
7. Hoogervorst, J.A.P.: Enterprise Governance and Enterprise Engineering (The Enterprise Engineering Series), 1st edn. Springer, Heidelberg (2009)
8. Lehman, M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE 68, 1060–1076 (1980)
9. Lehman, M.M., Ramil, J.F.: Rules and tools for software evolution planning and management. Annals of Software Engineering 11(1), 15–44 (2001)
10. Leist, S., Zellner, G.: Evaluation of current architecture frameworks. In: SAC 2006: Proceedings of the 2006 ACM symposium on Applied computing, pp. 1546–1553. ACM, New York (2006), http://doi.acm.org/10.1145/1141277.1141635
11. Mannaert, H., Verelst, J.: Normalized Systems—Re-creating Information Technology Based on Laws for Software Evolvability, Koppa, Kermt, Belgium (2009)
12. Mannaert, H., Verelst, J., Ven, K.: Exploring the concept of systems theoretic stability as a starting point for a unified theory on software engineering. In: Mannaert, H., Ohta, T., Dini, C., Pellerin, R. (eds.) Proceedings of Third International Conference on Software Engineering Advances (ICSEA 2008), pp. 360–366. IEEE Computer Society, Los Alamitos (2008)
13. Mulder, H.: Rapid enterprise design. Ph.D. thesis, TU Delft (2006)
14. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1053–1058 (1972)
15. van Reijswoud, V.: The structure of business communication: Theory, model and application. Ph.D. thesis, Technische Universiteit Delft (1996)
16. Van Nuffel, D., Mannaert, H., De Backer, C., Verelst, J.: Deriving normalized systems elements from business process models. In: International Conference on Software Engineering Advances, pp. 27–32 (2009), http://doi.ieeecomputersociety.org/10.1109/ICSEA.2009.13
17. Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Addison Wesley, Reading (1986)

Towards a G.O.D. Organization for Organizational Self-Awareness

David Aveiro[1,2], António Rito Silva[2,3], and José Tribolet[2,3]

[1] Exact Sciences and Engineering Centre, University of Madeira, Caminho da Penteada 9000-390 Funchal, Portugal
[2] Center for Organizational Design and Engineering, INESC-INOV Rua Alves Redol 9, 1000-029 Lisboa, Portugal
[3] Department of Information Systems and Computer Science, Instituto Superior Técnico, Technical University of Lisbon
{david.aveiro,jose.tribolet,rito.silva}@ist.utl.pt

Abstract. In this paper we draw on concepts from the Design and Engineering Methodology for Organizations, and also from its theoretical foundations, to discuss our notions of Organizational Self-Awareness and ontological meta model. These are deemed as central notions to understand and present, in a clear and precise manner, solutions for our main research purpose: finding concepts and methods to better handle organizational change caused by unexpected exceptions causing dysfunction in an organization's activity. Based on ontological notions like state base of a world and the ontological parallelogram, we arrive at precise definitions of what we call organizational self and its awareness. These then serve to put in perspective our proposal for the G.O.D Organization, considered to exist in every organization and being responsible for the Generation, Operationalization and Discontinuation of organization artifacts – e.g., actor role *pizza deliverer* – reflecting change of the organizational self. The main contribution of this paper is a discussion and clarification of how one can perceive an organization in a precise and thorough way, as to be able to keep a fact record of its relevant changes and, as a consequence, have a dynamic and "living model" of the organization.

Keywords: organizational engineering, organizational self-awareness, organizational change, model, meta-model.

1 Introduction

Our initial research efforts had the general purpose of understanding and clarifying what the *function perspective of an organization* should be. Normally, the function concept is associated with behavior, *activity* or operation of an organization or of a certain organizational unit like a marketing or IT department, normally responsible for the respective function [1]. In [2] we find that the function perspective means looking at a system from the point of view of the using system, in terms of provided functionality, i.e., kinds of behavior that can be caused. We regarded this to be an incomplete use of the term function. As a result of a review that we undertook on how

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 16–30, 2010.

this concept is used in such diverse areas as enterprise engineering, information systems, biology, sociology and philosophy, we found that, besides the aspect of *behavior*, also central to the function concept is the *normative* aspect (e.g., [3]), that is, the existence of certain normally expected values – *norms* – for certain vital properties of a system. In an organization, *deviations* from such norms imply a state of *dysfunction* that can possibly compromise its *viability*. Dysfunctions will have a *cause* which may be *expected* or *unexpected*. If the cause is expected, certain *resilience strategies* may already exist that can be activated to eliminate or circumvent dysfunctions [4], [3]. If the cause is unknown we will be in the presence of an *unexpected exception*. This unexpected exception will have to to be *handled* so that its concrete nature is detected and actions are undertaken that either eliminate or circumvent it, solving the dysfunction. The handling of unexpected exceptions constitutes another central aspect of the function perspective, namely *change* through the *(re)Generation*, *Operationalization* and *Discontinuation* of organizational artifacts which will eliminate or circumvent the determined cause of dysfunction. We consider an *organization artifact* (OA) as a construct of an organization like a business rule (e.g. "if invoice arrives, check list of expected items") or an actor role (e.g. library member). Change of OAs to handle dysfunctions is considered a special kind of dynamics that – inspired in philosophy literature on this subject – we call *microgenesis* [5]. We find that change is also driven by the detection of *opportunities of improvement* which will increase the viability of an organization and place it ahead of *competition* [6]. This is proactive change, as opposed to reactive change in the cases of resilience and microgenesis.

The focus of our research is in reactive change and on modeling the aspects of *resilience* strategies to solve known exceptions causing dysfunctions and (microgenesis) dynamics of handling unexpected exceptions also causing dysfunctions. This paper, focuses on theoretical issues underpinning our solutions for modeling resilience and microgenesis dynamics. In section 2 we develop on our main research problem and related work. Sections 3 and 4, the main contributions of this paper, present our discussion on the issues of organizational self-awareness and ontological meta model, necessary for a precise specification of our solutions. Section 5 provides a summary of or solution proposal – addressed in another report – which serves to consolidate the contributions of our discussion. Section 6 concludes with a short review of our contributions and issues raised for related work.

2 Problem, Motivation and Related Work

Above findings helped us to identify two relevant and closely interrelated more focused problems. On one hand, *a large amount of time is lost, in organizations, in the handling of unknown exceptions causing dysfunctions* as exception handling can sometimes take almost half of the total working time, and the handling of, and recovering from, exceptions is expensive [7]. On another hand, *current OE approaches seem to lack in concepts and method for a continuous update of organizational models, so that they are always up to date and available as a more useful input for the process of continuous change of organizational reality and decision on possible evolution choices*. We focus on these problems in the context of small timely changes, as opposed to large impact changes in the context of IT/IS projects, mergers,

acquisitions and splittings of organizations. It seems that the root problem for the above mentioned interrelated problems is an *absence of concepts and method for explicit capture, and management of information of exceptions and their handling, which includes the design and operationalization of OAs that solve caused dysfunctions*. Not immediately capturing this handling and the consequent resulting changes in reality and the model of reality itself, will result that, *as time passes, the organization will be less aware of itself than it should be*, when facing the need of future change due to other unexpected exceptions.

In terms of related research, the lack of awareness of organizational reality has been addressed in [8], with the coining of the term "Organizational Self-Awareness" (OSA). This construct has been further refined in [9] and [10]. OSA stresses the importance and need of continuously available, coherent, updated and updateable models of organizational reality. A recently proposed research discipline named Organizational Design and Engineering (ODE) [11], also defends this and further raises the importance of capturing and making organizational history and lessons learned available to organizational actors. OSA, and ODE claim that current OE approaches have the shortcoming of lacking in concepts and methods for a continuous update of models of organizational reality, aligned with the continuous change happening in the real terrain. However, both OSA and ODE have, for the most part, only addressed the issues of identification and formulation of this problem and, in terms of solution, mostly the aspect of representation, leaving the change aspect as future work.

This shortcoming of lack of continuous update of models aligned with the continuous change of reality has been addressed, by and large, in research and practice in the context of Workflow Management Systems (WfMS) – see, for example, [12] and [13]. However, current solutions assume that an organization will be using a WfMS, which will not be the case of many organizations. And, even in the case of organizations using WfMS, relevant activities may happen outside of IT context and we may also want to address exceptions related to them.

From our review and proposal of a broader notion of the function perspective and related insights brought from Complex Adaptive Systems (CAS) literature, we find that we may have two main types of change dynamics: *resilience* and *microgenesis.* From CAS [4] (p. 33) and philosophy [3], we find that systems maintain an internal model of the world (of themselves and the environment) so that they can activate specific resilience strategies to react, appropriately and in time, to certain known exceptions or fluctuations in critical norms that guarantee the system's viability. We also find that a system adapts with incremental changes [5], having as a main purpose to survive and evolve among competition, by having credit mechanisms which favor changes (adaptations) that increase the system's viability and constitute criteria of measuring success [4] (p. 34), [14] (p. 5). One of the premises from CAS theory is that, to solve new exceptions, "rule pieces" that constitute current resilience strategies that solve similar exceptions may be re-utilized to build new resilience strategies or new OAs to solve the new exceptions. From unexpected exception handling in WfMS [12] and insights from ODE discipline [11], we find that information on the history of organization change is an essential asset in the moments where change is again needed, i.e., in microgenesis dynamics. Modeling resilience and microgenesis dynamics and keeping a systematic history of their execution is deemed as a solution to our main research problem, so that exception handling and organization change is more

efficient and effective. Microgenesis is the main focus of our main research project. To precisely specify its dynamics we needed to also precisely specify resilience dynamics. Resilience and microgeneis have been the focus of other reports to be published elsewhere. In this paper we present a discussion on essential notions needed for a precise specification of microgenesis dynamics.

We ground our research in a particular Organizational Engineering approach, namely, the Design & Engineering Methodology for Organizations (DEMO) [2]. From several approaches to support OE being proposed, DEMO seems to be one of the most coherent, comprehensive, consistent and concise [2]. It has shown to be useful in a number of applications, from small to large scale organizations – see, for example, [15] and [16] (p. 39). Nevertheless, DEMO suffers from the shortcoming referred above. Namely, DEMO models have been mostly used to devise blueprints to serve as instruments for discussion of broader scale organizational change or development/change of IT systems [16] (p. 58) and does not, yet, provide modeling constructs and a method for a continuous update of its models as reality changes, driven by exceptions (microgenesis) nor for the continuous control (resilience) that we need to exert on organizations to guarantee viability. Contributions of our research – presented in the next sections – are heavily based in DEMO so, while proceeding, the reader which is unfamiliar with this methodology is advised to also consult [2] or [15] or other publications in: www.demo.nl.

3 Organizational Self-Awareness

3.1 Review of DEMO Concepts

We adopt the formal definition of ontological model of a world (from [17]) as: the *specification of its state space and its process space. Both are expressed in business rules*. By *state space* is understood the set of allowed or lawful states. It is specified by means of the state base and the existence laws. The *state base* is the set of fact types of which instances may exist in a state of the world. The existence laws determine the inclusion or exclusion of the coexistence of facts. We adopt also the ontological system definition from [18] (citing [19]) which concerns the construction and operation of a system. The corresponding type of model is the *white-box model*, which is a direct conceptualization of the ontological system definition presented next. Something is a system if and only if it has the next properties: (1) *composition*: a set of elements of some category (physical, biological, social, chemical etc.); (2) *environment*: a set of elements of the same category, where the composition and the environment are disjoint; (3) *structure*: a set of influencing bonds among the elements in the composition and between these and the elements in the environment; (4) *production*: the elements in the composition produce services that are delivered to the elements in the environment. From [18] we find that in the Ψ-theory based DEMO methodology, four aspect models of the complete ontological model of an organization are distinguished. The Construction Model (CM) specifies the construction of the organization: the actor roles in the composition and the environment, as well as the transaction kinds in which they are involved. The Process Model (PM) specifies the state space and the transition space of the C-world. The State Model (SM)

specifies the state space and the transition space of the P-world. The Action Model (AM) consists of the action rules that serve as guidelines for the actor roles in the composition of the organization.

We see that the definition of ontological model of [17], is extended into the more complete definition above. In [2] it is said that, from the AM one can derive all other aspect models, namely that "the AM is in a very literal sense the basis of the other aspect models since it contains all information that is (also) contained in the CM, PM, and SM; but in a different, and not so easily accessible, way". We agree that one can in fact obtain all information for the PM from the AM. But for the SM and CM this claim does not appear to be true. Many fact types and event types – part of the SM – do appear in action rules but one may not be able to consistently derive, for example, unicity and dependency laws from them. In action rules we do find reference to all transactions and P-acts and C-acts that create C-facts and P-facts. If we consider that a set of action rules constitutes and defines an actor role in the AM – just like presented in [2] – we will still be missing relevant information needed to completely obtain the CM namely: (1) which are the external actor roles (2) how internal actor roles are composed or decomposed in other (composite or simple) actor roles and (3) how all these actors interact through transactions. So it appears that, the AM is not sufficient to derive all other aspect models and completely describe an organization. Not only that happens but the AM, in the original DEMO method [2] and in the current [20], is the third one being elicited. On top of that, the only research work we found evaluating application of DEMO in projects – in a 10 year span – verified that the Action Model was not part of any project experience until then [21]. We consider that the AM is in fact an essential aspect to have a precise modeling of an organization, but probably should not be considered as the base from which all other models can be derived.

Before presenting some of our main claims, we present, in Figures 1 and 2, respectively, the meaning triangle and the ontological parallelogram, taken from [22] which explain how (individual) concepts are created in the human mind. We will also base our claims in the model triangle, taken from [2] and presented in Figure 3. We find that the model triangle coherently overlaps the meaning triangle. This happens because a set of symbols – like a set of DEMO representations (signs) that constitute a symbolic system – allows the interpretation of a set of concepts – like a set of DEMO aspect models, part of the ontological model, constituting a conceptual system. This conceptual system, in turn, consists in the conceptualization of the "real" inter-subjective organizational self, i.e., the set of OAs constituting the concrete organization system's composition structure and production. Figure 4 is an adaptation from the model triangle of Figure 3 and depicts our reasoning. We call the set of all DEMO diagrams, tables and lists used to formulate the ontological model as *ontological representation*. Now relating with the meaning triangle, we can verify that a particular sign (e.g., a transaction symbol with label membership fee payment), part of an ontological representation (e.g., actor transaction diagram) designates (i.e., allows the interpretation or is the formulation) of the respective concept of the particular transaction part of the respective ontological model (e.g., construction model). This subjective concept, in turn, refers to a concrete object of the shared inter-subjective reality of the organization's human agents (e.g., the particular OA transaction T02). Figure 5,

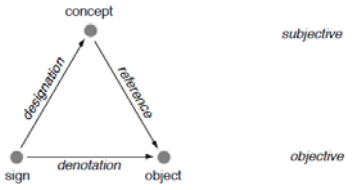

Fig. 1. The meaning triangle

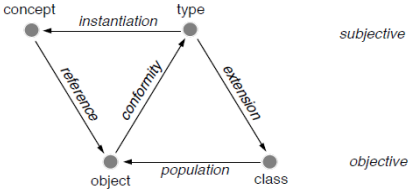

Fig. 2. The ontological parallelogram

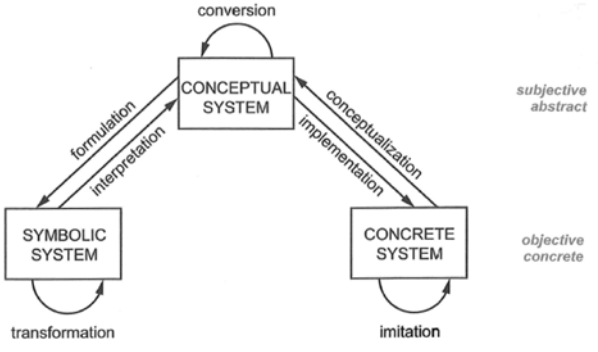

Fig. 3. The model triangle

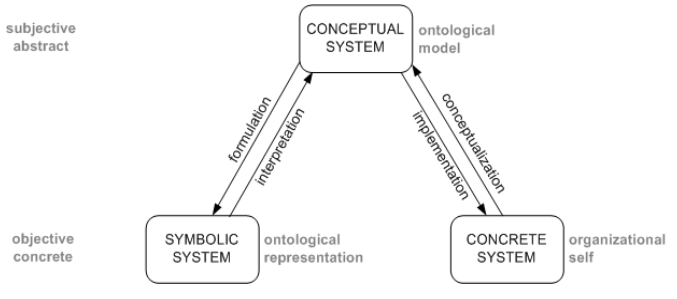

Fig. 4. Model triangle applied to organizations

an adaptation from the meaning triangle depicts this other reasoning. Another example of an OA related with T02 would be the transaction initiation OA, relating T02 with actor role registrar (also designated by A02) and formulated by a line connecting the transaction and actor role symbols of T02 and A02. Actor role registrar is, in turn, another OA of the construction space of the library. Once such role is communicated to all employees of a library, it becomes a "living" abstract object part of the shared inter-subjective reality of the library's human agents. Such object, along with other OAs of the organizational inter-subjective reality, give human agents a way to conceptualize their organizational responsibilities – in this case,

requesting membership fee payments to aspirant members. We name this set of all abstract objects living in the inter-subjective reality of an organization's members as the *organizational self*.

3.2 Devising the Notion of Organizational Self-Awareness

With the previous clarifications and preparation of scope, we now present one major claim of our research which is that *every organization should formally maintain an OAs base which is a clear and coherent specification of the organization system's composition, structure and production. These properties of an organization system can be named as organizational self. The ontological model of an organization consists in the conceptualization of the set of OAs constituting its current organizational self*. OAs are, themselves, inter-subjective concepts existing in the minds of an organization's human agents. But, following the logic of the ontological parallelogram, thoughts in one's mind and, thus, concepts, can be viewed as objects, as we can think about our own thoughts. So we consider OAs as concepts that are objects of the shared inter-subjective reality of an organization's human agents. Such OAs, in turn, specify: (1) the types of facts allowed to exist in the organization world, along with the laws restricting coexistence of facts – i.e., the *state space OAs*; (2) the types of events allowed to occur in the organization world, along with laws restricting the sequencing of occurrence of events – i.e., the *process space OAs*; (3) the action rules which are guidelines for allowed action, grouped in actor roles – i.e., what we suggest to call the *action space OAs*; and (4) the composition of the organization, in terms of allowed elementary (and/or composite) actor roles, how these are composed or decomposed in other (elementary or composite) actor roles, allowed external actor roles and allowed interactions between all such actor roles, through allowed transactions – i.e., what we suggest to call the *construction space OAs*.

An important notion that we point out in this paper is the fact that an organization – besides producing a set of products or services for its environment – also produces itself. That is, enclosed in its day-to-day operation, there will be parts of its operation which change the organization system itself, i.e., change the set of *OAs* that constitute its composition, structure and production. Another important notion is that, in parallel to this process of change, we have what we propose to call the process of *Organizational Self-Awareness* (OSA). We propose to redefine OSA as the continuous synchronization of the "real" organization system – i.e., the "real" and concrete organizational self – with its ontological model – i.e., the conceptual organizational self – and its ontological representation – i.e., a symbolic system which is a formulation of the conceptual organizational self. We can see that communication, the used symbolic system and rules for arranging the OAs play a major role in OSA. Because both the organizational self and its ontological model live in the minds of an organization's human agents and each individual subjective images may differ or be incoherent, a strong effort must be in place to synchronize all minds for the most coherent possible specification of the organizational self.

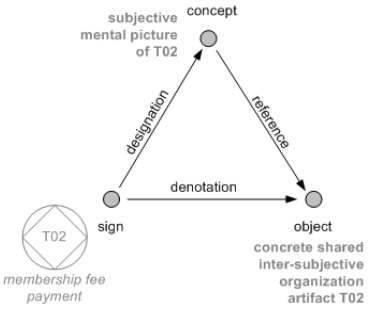

Fig. 5. Meaning triangle applied to a transaction OA

One cannot change what one is not aware of and, because organizational reality is constantly changing, following the tenets of OSA and ODE: if one wants to change the organization in a controlled and precise manner, it is helpful that one (1) has an ontological model of the organizational self and and respective ontological representation as most updated and coherent as possible, as well as (2) keeps a record of the change history of the organizational self. This means that, also enclosed in the day-to-day operation of an organization, there will be parts of its operation which change the organizational self. By formally and explicitly specifying these change acts one keeps a definite and updated record of produced OAs. Such a record – the OAs base – constitutes the means for one to always be able to conceptualize the most current and updated ontological model of the organizational self. This leads to another important claim of this paper which is: *the continuous production of the organizational self should include the synchronized production of the collective and subjective "picture" (awareness) of the organizational self – the conceptualization that constitutes its ontological model – by the synchronized production of the respective symbolic system – an ontological representation that allows the interpretation of the ontological model and the conceptualization (awareness) of the organizational self.*

Awareness means to be aware of the present and have memory of the past and how it shaped the present. Human consciousness is constantly aware of its actions and how they change reality, because reality and actions that are also part of reality, are constantly reflected in its consciousness and memorized. In a similar manner, organizational awareness of an organization O is realized by O constantly "reflecting" and "memorizing" all kinds of relevant acts that change relevant organizational reality, along with the "as-was" and "as-is" states of O's self. To separate concerns, we propose that such acts are performed by a (sub-)organization considered to exist in every organization that we call: G.O.D. Organization (GO). The GO's production world will contain the current state of O's self as well as its relevant state change history. The GO has the role of continuously realizing and capturing changes of organizational reality. Thus, by implementing the GO pattern in a real organization, in an appropriate manner, providing automatic generation of ontological representations derived from the OAs base, one can achieve Organizational Self-Awareness. This is possible because one can implement clear rules that, based on the arrangement of OAs of the organizational self, automatically produce the appropriate ontological

representation which, in turn, allows the appropriate interpretation of the ontological model that is the correct conceptualization of the organizational self. In other words, as another step of devising a solution to our research problem, we claim that *every organization has a G.O.D. Organization (GO) responsible for its microgenesis dynamics and the GO's world state base includes a record of the set of OAs that constitute the organizational self as well a record of its state transitions, up to its most current state, i.e., the set of all facts of Generation, Operationalization and/or Discontinuation of each OA.*

We adopt the formal definition of a world, presented in [17], where B is the set of facts that constitute the state base of a world, and S the set of facts that are current, specifying the current state of the world where $S \subset B$. Therefore, we will refer to the set of facts of the GO that constitute the state base of the organizational self (OS) as GB and to the subset of facts that constitute the current state of the OS as GS where $GS \subset GB$. In other words, every organization has a set GS which constitutes the specification of the OS's current state and a set GB which constitutes the full history of the OS, including its current state. In [2] it is considered that the notion of system state is ambiguous, because changes in the composition or structure of a system may also be considered as state changes. Current notions of coordination and production worlds of an organization O provided in DEMO do not address the issue of changes in the state of the composition and structure and production of the organization system. They only addresses changes in the state of its operation. These worlds focus on what O produces to its environment and coordination dynamics that occur for such production. Following our above claims, this ambiguity issue is solved by modeling state changes of the organization system as state transitions occurring in the production world of the GO, reflecting the Generation, Operationalization and Discontinuation of OAs of O. In other words, the world of O keeps a fact record of its "normal" production and the world of GO keeps a fact record of the production of O's self.

For a precise formulation of our proposal of the G.O.D. organization we needed to address the notion of ontological meta-model, presented next.

4 Ontological Meta Model

OAs constituting the organizational self are arranged in a certain manner as to specify all the spaces (state, process, action and structure) of an organization's world, i.e., they have to obey certain rules of arrangement between them. We will call the specification of these rules as the *ontological meta model*. The ontological meta-model is the conceptualization of the *OA space*. By OA space we understand the set of allowed OAs. It is specified by the *OA base* and *OA laws*. The *OA base* is the set of *OA kinds* of which instances, called *OAs*, may occur in the state base (set GB) of the GO's world. The *OA laws* determine the inclusion or exclusion of the coexistence of OAs.

The definition of the OA space is quite similar to the definition of state space of an organization's production world – specified in World Ontology Specification Language (WOSL) [22] – and, thus, it is appropriate to use WOSL to express the ontological meta-model in, what we propose to call: the *Organization Space Diagram* (OSD). DEMO's OSD is currently called as the *DEMO Meta Model* (DMM), the

chosen name for the specification provided in [23] and consisting, in practice, in the OSD corresponding to the four DEMO aspect models: SM, CM, PM and AM. They are called, respectively: *Meta State Model*, *Meta Construction Model*, *Meta Process Model* and *Meta Action Model*. We argue that, also respectively, *State Organization Space Diagram*, *Construction Organization Space Diagram*, *Process Organization Space Diagram* and *Action Organization Space Diagram* would be more coherent names. We argue this, because each of these are specifying, in WOSL, what we call the OA space of each of DEMO's aspect models. That is, these diagrams formulate, for each aspect model, the OA kinds out of which instances – OAs – can occur in the organizational self and coexistence rules governing how to arrange these instances. Another reason we propose to use the expression Organization Space Diagram is because we're in fact looking at a Space Diagram which, following the model triangle [2], is a symbolic system which is a formulation of the conceptual system of the ontological meta model. So, for coherency reasons, one should not use terms "Meta" and "Model" to name those figures but use, instead, the term Organization Space Diagram. The OSD allows the interpretation, in one's mind, of the ontological meta model. The complete set of organization artifact kinds and laws governing the arrangement of their instances constitutes the organization space. The conceptualization of the organization space consists in the ontological meta model which, in turn, is formulated in what we call the Organization Space Diagram. A depiction of this reasoning is present in Figure 6, another adaptation from the model triangle.

For a particular organization, each of its aspect models consists in the conceptualization of organization artifacts, instances of organization artifact kinds of their respective organization space. Following the adopted philosophical stances for an ontology and the logic of the ontological parallelogram, each aspect model is a set of concepts that refer to "real" organization artifacts. We remind that these artifacts, although being abstract objects, are objects of the inter-subjective reality of the organization's human agents. A certain organization artifact will be a member of an organization artifact class. Continuing our example of the previous section, organization artifact T02 is a member of class TRANSACTION KIND which is a particular organization artifact kind, also living in the inter-subjective reality (but objective in terms of the ontological parallelogram). Class TRANSACTION KIND is an *extension* of the subjective and generic type transaction kind, from which the concept of membership fee payment is an *instance*. Figure 7 depicts this last reasoning and example.

Summarizing, the ontological model of an organization is the set of all concepts that *refer* to the "real" inter-subjective organization artifacts that constitute the "real" inter-subjective organization, i.e., the organizational self. The ontological meta model applies to every organization and is the set of all types that refer to the "real" inter-subjective organization artifact kinds and constitute the "real" inter-subjective *organization space*. All DEMO representations (sets of symbols/signs) are a way to *formulate* the set of concepts that constitute the conceptual system that the ontological model of an organization is. They *denote* the set of organization artifacts that constitute the organizational self. The DEMO OSD is a way to formulate the set of generic types that constitute the organization space.

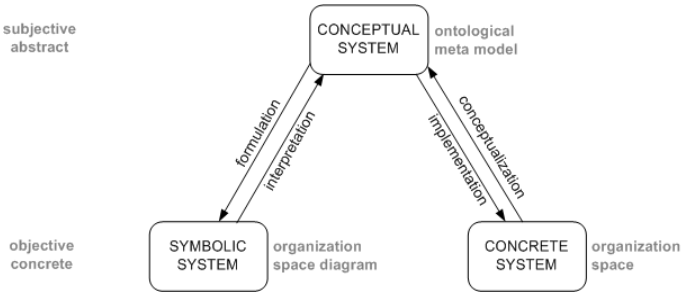

Fig. 6. Model triangle applied to the organization space

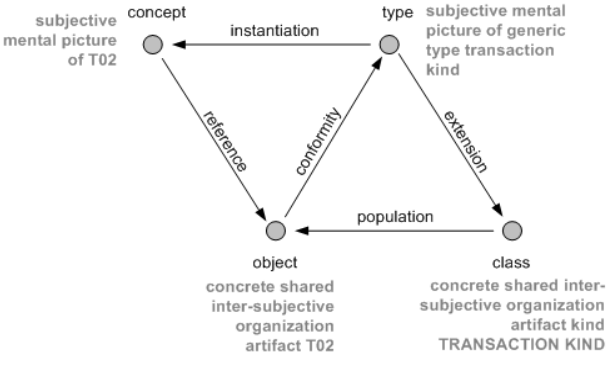

Fig. 7. Ontological parallelogram applied to a transaction organization artifact

An organization's action manifests by its agents following a certain order specified in the organizational self and these same agents constantly change the shape of this same self, as needed, continuously generating, operationalizing and discontinuing organization artifacts that conform with certain types part of the conceptual system that is the ontological meta model. The main business of the G.O.D. Organization of a particular organization O is to formally manage the life cycle of all relevant inter-subjective organization artifacts of O that constitute its self and are instances of organization artifact kinds. Organization artifacts specify the previously mentioned *state space*, *structure space*, *process space* and *action space* of O, shaping, in turn, the operation of O.

The G.O.D. organization is addressed in detail in another report but, in the next section, we present an overview of it, which serves to consolidate our above discussion of notions of organizational self-awareness and ontological meta model.

5 G.O.D. Organization Overview

Figure 8 presents, on the top, the formulation of part of the ontological meta model in the shape of part of the Construction OSD and of the State OSD. On the bottom left we find the formulation of part of the ontological model of the library, namely part of

its Organization Construction Diagram (OCD) and of its State Space Diagram (SSD). Organization artifacts represented in these two diagrams are instances of their respective kinds represented on the respective OSD. On the bottom right we find the formulation of part of the G.O.D. Organization's ontological model, i.e., part of the G.O.D. Organization's OCD and SSD. In the Library's SSD we find a fact type – that we can identify by name FT01 – expressed in the following predicative sentence, also depicted in Figure 8: *copies of [book] are delivered in [shipment]*. To express fact instances and fact types, we're adopting the notation used in the current version (3) of DEMO [20] which makes diagrams much more readable. FT01 itself is an OA (a fact type) of the state space of the library's SM, constituting part of the library's organizational self. FT01 allows the specification of fact instances describing which book instances of external object class BOOK are delivered in a certain instance of SHIPMENT.

Following our claim presented above, although, from the perspective of the ontological model, FT01 belongs to the type level, it happens that, from the point of view of the GO – responsible for change dynamics – FT01 is looked upon as a fact – a particular OA – instance of a certain fact type (from the meta model, thus, a meta type) – which is a particular OA kind. In our example, FT01 is a fact type of the library,

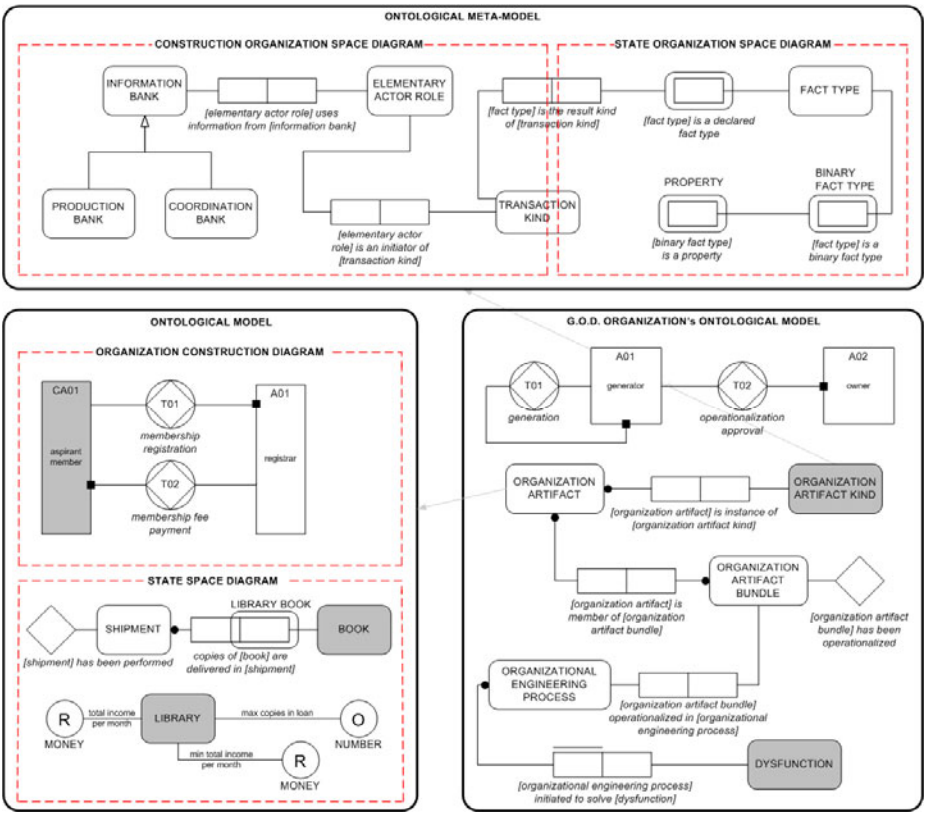

Fig. 8. G.O.D. Organization Overview

which, from the perspective of the GO is an OA, which is an instance of a particular OA kind, namely a fact type. Because FT01 is an OA that is part of the current organizational self of the library, it belongs to the respective set GS. We find object classes ORGANIZATION ARTIFACT and ORGANIZATION ARTIFACT KIND on the (part of the) G.O.D. Organization's ontological model depicted on the right part of Figure 8. The first is the population of all OAs that constitute a particular organizational self – in this case, of the library – while the second is the population of all OA kinds that constitute what we have previously called the organization space.

Changes in the organizational self will occur by state changes of bundles of OAs – whose population is represented by object class ORGANIZATION ARTIFACT BUNDLE – as never an isolated OA will be created. Let's suppose that, for some reason, one needs to specify a new result kind, to be added to the library's SM, affecting shipments, namely: *[shipment] has been returned*. This implies the generation of an associated transaction kind, as well as of an actor role to initiate it. All this, in turn, implies that the GO will produce OAs, instances of (1) OA kinds, that are object classes at meta level: fact type, transaction kind, actor role as well as of (2) OA kinds that are fact types at meta level: *[fact type] is the result kind of [transaction kind]*, *[elementary actor role] is an initiator of [transaction kind]*, etc. All such instances will be part of a bundle that will be operationalized after its generation and approval.

A major contribution of our research is making the aspect of change of the organizational self explicit with the GO. This (sub)organization allows us to make an "operational bridge" between the meta model and model levels so that we can keep a precise and thorough track of the state of the organizational self and its state transitions or, in other words, the life cycle of organization artifacts that constitute the organizational self. The OAs that are current and constitute an organization's current self, will be the ones that are part of bundles whose last result was that they have been operationalized. The conceptualization of this set of OAs constitutes the organization's ontological model.

6 Conclusions

Current notions of coordination and production worlds provided in DEMO do not address the issue of changes in the state of the composition and structure of the organization system which means that one can use DEMO only to take "static pictures" of an organization. Following our claims presented in this paper, this ambiguity issue is solved by modeling such state changes as state transitions occurring in the production world of the G.O.D. Organization, reflecting the Generation, Operationalization and Discontinuation of OAs. OAs are a central concept in this paper which we propose to be abstract objects, existing in the inter-subjective reality of human agents of an organization and constituting the organizational self which, in turn, defines the operation of the organization. The conceptualization of the organizational self constitutes the organization's ontological model. Keeping a precise record of the acts of production of OAs is the way that we propose to realize Organizational Self-Awareness, that we (re)define as the continuous synchronization of the real organizational self with its ontological representation, for a correct conceptualization of the ontological model. This implies that the OAs base should be constantly available to

all relevant human agents of an organization, so that they can always conceptualize the current and correct "version" of the part of the organizational self that they should be aware of.

We have shown how the G.O.D. Organization makes a bridge between the worlds of model and meta model, where the latter contains the set of generic OA (meta level) types out of which a set of (model level) OA instances can be generated that constitute an organization's self. With our proposal of the G.O.D. Organization, DEMO no longer is limited to a "static" picture of an organization and we can now have a full trace of the state of the organization system. The current picture of the organization, or, in other words, its ontological model, simply consists in the conceptualization of the set of OAs that are current, i.e., belonging to a bundle of OAs whose last event was "has been operationalized".

As related research being addressed in other reports, we delve on the specification of the full ontological model of the G.O.D. Organization which ends up being the specification of a method to formally capture microgenesis dynamics and realize Organizational Self-Awareness. We also address the ontological model of what we call the Control Organization, responsible for resilience dynamics, i.e., observing if certain critical properties characterizing an organization's operation – called measures – are respecting the values – called viability norms – that guarantee an organization's viability and activating (and deactivating) bundles of transactions to overcome dysfunctions on such norms and, if unsuccessful, to make a request for the G.O.D. organization to initiate what we call an Organizational Engineering Process to handle the problem and change the organizational self to solve it.

Acknowledgment

Research work that led to results presented in this paper was possible thanks to the financial support of a PhD scholarship (Ref.: SFRH / BD / 13384 / 2003) subsidized by "Fundação para a Ciência e a Tecnologia - Ministério da Ciência, Tecnologia e Ensino Superior" of the Portuguese government and by the European Social Fund.

References

1. Applegate, L.M., McFarlan, F.W., McKenney, J.L.: Corporate information systems management: text and cases. Irwin/McGrawHill (1999)
2. Dietz, J.L.G.: Enterprise ontology: theory and methodology. Springer, New York (2006)
3. Christensen, W.D., Bickhard, M.H.: The process dynamics of normative function. The Monist 85, 3–29 (2002)
4. Holland, J.H.: Hidden order: how adaptation builds complexity. Basic Books, New York (1996)
5. Bickhard, M.H.: Error dynamics: the dynamic emergence of error avoidance and error vicariants. Journal of Experimental & Theoretical Artificial Intelligence 13, 199–209 (2001)
6. Brown, S.L., Eisenhardt, K.M.: Competing on the edge: strategy as structured chaos. Harvard Business School Press, Boston (1998)
7. Saastamoinen, H., White, G.M.: On handling exceptions. In: Proceedings of conference on Organizational computing systems, pp. 302–310. ACM, New York (1995)

8. Tribolet, J.: Organizações, pessoas, processos e conhecimento: da reificação do ser humano como componente do conhecimento à "consciência de si" organizacional (organizations, people, processes and knowledge: from the reification of the human being as components of knowledge to the knowledge of organizational self). Sistemas de informação organizacionais. Sílabo Editora, Lisbon, Portugal (2005)
9. Magalhães, R., Zacarias, M., Tribolet, J.: Making sense of enterprise architectures as tools of organizational self-awareness (OSA). In: Proceedings of the Second Workshop on Trends in Enterprise Architecture Research (TEAR 2007), June 2007, vol. 6, pp. 61–70 (2007)
10. Zacarias, M., Magalhães, R., Caetano, A., Pinto, H.S., Tribolet, J.: Towards organizational self-awareness: an initial architecture and ontology. In: Handbook of ontologies for business interaction, pp. 101–121. Information Science Reference (2007)
11. Magalhães, R., Silva, A.R.: Organizational design and engineering (ode) - ode white paper - Version 1 (2009)
12. Mourão, H.: Supporting effective unexpected exceptions handling in workflow management systems within organizational contexts. Science Faculty of Lisbon University (2007)
13. Casati, F., Pozzi, G.: Modeling exceptional behaviors in commercial workflow management systems. In: Proceedings of the Fourth CoopIS International Conference on Cooperative Information Systems, pp. 127–138 (1999)
14. Axelrod, R., Cohen, M.D.: Harnessing complexity: organizational implications of a scientific frontier. Basic Books, New York (2001)
15. Dietz, J.L.G., Albani, A.: Basic notions regarding business processes and supporting information systems. Requirements Engineering 10, 175–183 (2005)
16. Op' t Land, M.: Applying architecture and ontology to the splitting and allying of enterprises. TU Delft (2008)
17. Dietz, J.L.G.: Architecture building strategy into design. Academic Service - Sdu Uitgevers bv. (2008)
18. Dietz, J.L.G.: On the nature of business rules. In: Advances in Enterprise Engineering I, pp. 1–15 (2008)
19. Bunge, M.A.: Treatise on basic philosophy, a world of systems, vol. 4. Reidel Publishing Company, Dordrechtz (1979)
20. Dietz, J.L.G.: Demo-3 models and representations (2009), `http://www.demo.nl`
21. Dumay, M., Dietz, J.L.G., Mulder, H.: Evaluation of demo and the language/action perspective after 10 years of experience. In: Proceedings of LAP 2005 (2005)
22. Dietz, J.L.G.: A world ontology specification language. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 688–699. Springer, Heidelberg (2005)
23. Dietz, J.L.G.: Demo meta model specification (forthcoming, 2009), `http://www.demo.nl`

Understanding the Realization of Organizations

Joop de Jong[1,2] and Jan L.G. Dietz[1]

[1] Delft University of Technology,
P.O. Box 5031, 2600 GA Delft, The Netherlands
[2] Mprise
P.O. Box 598, 3900 AN Veenendaal, The Netherlands
jdjong@mprise.nl, j.l.g.dietz@tudelft.nl

Abstract. An organization can be understood as a social system, i.e. a system whose elements are social individuals or actors. The actors operate in an environment of customers, suppliers, partners and others which share a part of the organization's world. The effects of the acts of all of these actors can be understood as state changes of the organization's world. All information that is needed by the actors consists in either the facts that constitute the world's states or in information that is derived from these facts. In DEMO (Design and Engineering Methodology for Organizations) an organization is conceived as the layered nesting of three aspect organizations: the B-organization (from Business), the I-organization (from Intelligence) and the D-organizations (from Document). Whereas B-actors perform business acts, I-actors remember and derive information concerning the business, and D-actors store, transport and retrieve documents that contain this information. The design of the integration between the B-, the I-, and the D-organization is called the realization of the organization. This paper presents and discusses how the realization of organizations can be understood thoroughly, as well as how the I-organization can be derived from the B-organization, and the D-organization from the I-organization.

Keywords: enterprise engineering, enterprise ontology, information management, DEMO.

1 Introduction

In order to cope with current and future problems and challenges in enterprises, a conceptual model of the enterprise is needed that is coherent, comprehensive, consistent and concise, and that only shows the essence of the construction and the operation of an enterprise' organization, abstracted from all realization and implementation issues. By realization is meant the activity of establishing the heterogeneous organization as a layered nesting of homogenous organizations and by implementation is meant the activity of making the organization operational by means of appropriate technology [1]. The way of thinking and the way of modeling such a conceptual model of the enterprise are provided by the Design and Engineering Methodology for Organizations, DEMO for short [1, 2]. Although DEMO abstracts from all realization issues, the underlying ψ-theory of DEMO conceives the organization as the layered

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 31–49, 2010.

nesting of three aspect organizations, viz. the B-organization (from Business), the I-organization (from Intelligence) and the D-organizations (from Document).

Dietz [1] writes that all three homogeneous systems are in the category of social systems. In previous papers Dietz [3] and Maij cs. [4] derive a set of use cases from the ontological model of the B-organization of the organization that constitute the starting point for the development of information systems, using one of the UML – based engineering methods. According to Shishkov and Dietz [5], the B-organization provides a starting point for the definition of functional requirements of information systems. Mallens, Dietz and Hommes [6] define an information system based on the B-organization as a system in the category of rational systems. They call the elements of the system rational individuals which perform rational actions like collecting, recalling, providing knowledge, calculating and making logical deductions. At last, Mulder [7] writes about actors from the I- and D-organization as rational components embedded in an information system and infrastructure system, respectively. Till now conceptual models of the B-, I- and D-organization are lacking.

This paper contributes to the design of a conceptual model of the B-organization, the I-organization and the D-organization that abstracts from all implementation issues in order to cope with current and future problems and challenges in remembering and reproducing facts and in archiving and collecting documents.

Section 2 provides a summary of the ψ-theory. Actor roles and transaction types are building blocks for the construction of the organization. A subject who fulfills an actor role uses several abilities while performing a transaction. As a prelude to section 4 some important issues for conceptual modeling the B-, I- and D-organization are discussed in section 3. Section 4 presents the way the conceptual models of the I-organization and of the D-organization are derived from the conceptual model of the B-organization and from the conceptual model of the I-organization, respectively. It discusses the way of modeling the I-organization and the D-organization and elaborates some important I- and D-transaction kinds. All steps from the creation of an original fact up to its storage in a fact bank, as well as the steps taken for information delivery based on the original facts are discussed. The paper ends with some conclusions and directions for further research in section 5.

2 Summary of the Ψ-Theory

For a good understanding of this paper a summary of the ψ-theory on which Dietz [1] based the DEMO methodology is presented. Dietz argues that in order to cope with the current and future challenges, a conceptual model of the organization is needed that is coherent, comprehensive, consistent and concise, and that only shows the essence of the operation of an organization model. Such a model, called an ontological model, abstracts from all implementation and realization issues. The underlying theory is called the ψ-theory. The ψ-theory consists of four axioms, viz. the operation axiom, the transaction axiom, the composition axiom and the distinction axiom, and the organization theorem. In this section, these axioms and the organization theorem are elaborated briefly. An exception is made for the composition axiom which is of no importance for the subject of this paper.

The *operation axiom* states that the operation of the organization is constituted by the activities of actors, which are elementary chunks of authority and responsibility fulfilled by human beings. Actors perform two kinds of acts: production acts, or P-acts for short, and coordination acts, or C-acts for short. These acts have definite results, namely production facts and coordination facts, respectively.

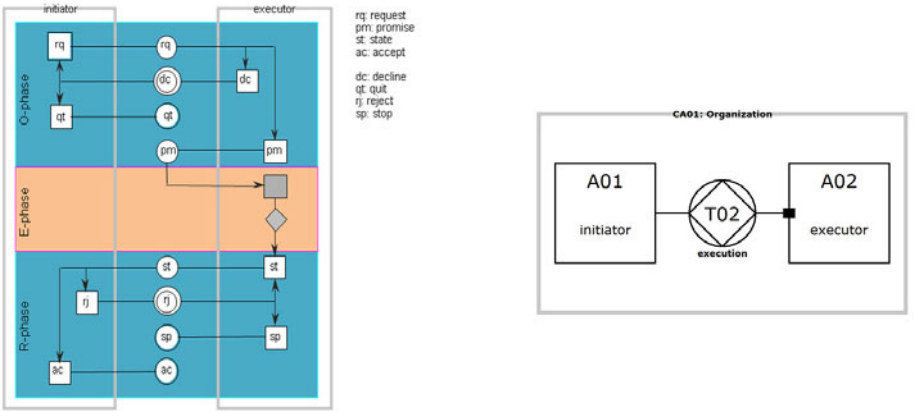

Fig. 1. The standard transaction pattern elaborated (left) and in DEMO notation (right)

By performing P-acts, actors contribute to bringing about goods or services or information or data that are delivered to other actors. A P-act is either material or immaterial. Examples of material acts are manufacturing and storage of goods and transportation acts or computing and deducing information acts. Examples of immaterial acts are the judgment by a court to condemn someone, granting an insurance claim and selling goods. By performing C-acts, actors enter into and comply with commitments towards each other regarding the performance of P-acts. A C-act is defined by its proposition and its intention. The proposition consists of a P-fact, e.g. “Purchase order #200 is delivered” and a delivery date. The intention represents the purpose of the performer; examples of intentions are “request”, “promise” and “decline”. The effect of performing a C-act is that both the performer and the addressee of the act get involved in a commitment regarding the referred P-act.

The *transaction axiom* states that coordination acts are performed as steps in universal patterns. These patterns, also called transactions, always involve two actor roles, i.e. two chunks of authority and responsibility. They are aimed at achieving a particular result, the P-fact. Figure 1 exhibits the standard transaction pattern. A transaction evolves in three phases: the order phase (O-phase for short), the execution phase (E-phase for short) and the result phase (R-phase for short). One of the two partaking actor roles is called the initiator, the other the executor of the transaction. In the order phase, the initiator and the executor pursue to reach agreement about the P-fact that the executor is going to bring about as well as the intended time of creation. In the execution phase, the executor brings about this P-fact. In the result phase, the initiator and the executor pursue to reach agreement about the P-fact that is actually produced as well as the actual time of creation (both of which may differ from the

requested one). Only if this agreement is reached will the P-fact become existent. The path request-promise-execute-state-accept in figure 1 is called the basic pattern; it is the course that is taken when the initiator and the executor keep consenting. However, they may also dissent. There are two states where this may happen, namely the states "requested" and "stated". Instead of promising one may respond to a request by declining it, and instead of accepting one may respond to a statement by rejecting it. It brings the process in the state "declined" or "rejected" respectively. These states are indicated by a double disk, meaning that they are discussion states. If a transaction ends up in a discussion state, the two actors must 'sit together', discuss the situation at hand and negotiate about how to get out of it. The possible outcomes are a renewed request or statement (probably with a modified proposition) or a failure (quit or stop).

The *distinction axiom* states that there are three distinct human abilities playing a role in the operation of actors, called performa, informa and forma (cf. fig. 2). Those abilities are recognized in both kinds of acts that actors perform.

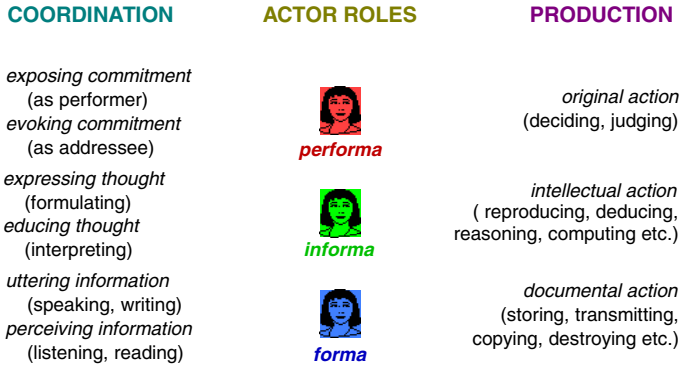

Fig. 2. The three human capabilities

Let us first look at the P-act of an actor. The forma ability is the human ability to conduct documental actions, such as storing, retrieving, transmitting, etc. These are all actions by which the content of the documents or data is of no importance. Actors which use the forma ability to perform P-acts are called documental actors or D-actors for short. The informa ability is the human ability to conduct intellectual actions, such as reasoning, computing, remembering and recalling of knowledge, etc. These are all actions by which the content of data or documents, abstracted from the form aspects, is of importance. Actors which use the informa ability to perform P-acts are called intellectual actors, or I-actors for short. The performa ability is the human ability to conduct new, original actions, such as decisions, judgments etc. The performa ability is considered as the essential human ability for doing business, of any kind. Actors which use the performa ability to perform P-acts are called business actors or B-actors for short. Earlier in this section we gave a short summary about the operation axiom and wrote about manufacturing, storage of goods and transportation as P-acts. Actually, that is not fully true. These actions have to be considered as actions which are based on original decisions. The P-act must be considered as taking this decision and the P-fact must be considered as the ultimate decision.

Let us next look at the C-act of an actor. By performing C-acts, actors enter into and comply with commitments towards each other with respect to the performance of P-acts. The effect of performing a C-act is that both the performer and the addressee of the act get involved in a commitment concerning the referred P-act. That commitment is a result of a performative exchange that only can taken place by actors in their performa ability. However the only way of the performer of the C-act to expose its commitment and to make it knowable to the addressee, is to express its informa ability, followed by the inducement in the mind of the addressee of an equivalent thought, by means of its informa ability. The intellectual understanding between both actors comes into existence by informative exchange. Expressing a thought can only be done by formulating it in a sentence in some language, and at the same time uttering it in some form, such as speaking or writing. Significational understanding between both actors only comes into existence by a formative exchange by actors in their forma ability. That means that the performer has to use a language that is known to the addressee. So, during a C-act both the performer and the addressee shape between abilities several times. We would put with great emphasis that B-actors, I-actors and D-actors only distinguish themselves by the kind of production act.

The last part of the ψ-theory that we would like to explain is the *organization theorem*. It states that the organization of an enterprise is a social system that is constituted as the layered integration of three homogeneous systems: the B-organization, the I-organization, and the D-organization. The D-organization supports the I-organization, and the I-organization supports the B-organization (cf. fig. 3). All three systems are called aspects systems of the total organization of the enterprise. A system can be considered from two different perspectives, namely from the function perspective or from the construction perspective. The function perspective on a system is based on the teleological system notion which is concerned with the (external) behavior or performance of a system.

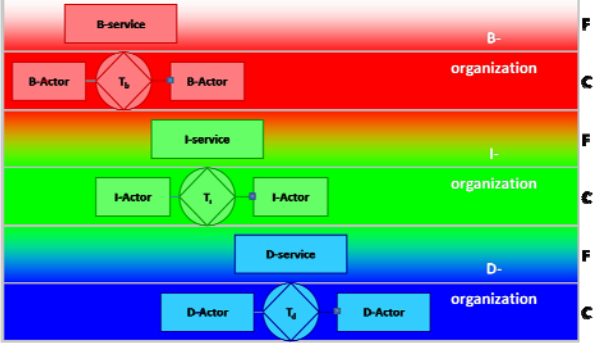

Fig. 3. The layered integration of an enterprise system

This notion is adequate for the purpose of controlling or using a system. It is about services or products which are delivered by the system. The construction perspective on a system is based on the ontological system notion. Systems have to be designed and constructed. During the design and engineering process questions of being effectively and efficiency of the chosen design have to be answered by the constructor of

the system. Our point of departure in this paper is the ontological system notion. The integration between the three organizations is established through the cohesive unification of human being. Let us elaborate this point more in detail. We take the I-organization as our starting point. From the functional perspective the I-organization provides an information service to the B-organization, i.e. to a B-actor which actually interprets the received data as required information in order to execute a particular act, i.e. a C-act or a P-act. However, how does a B-actor receive information from an I-actor actually? The answer is given by the distinction axiom. The subject who fulfills the B-actor role is able to shape, for a while, into an informa shape to receive the requested information [1, 8]. This information is produced by cooperating I-actors and it is based on original facts which have been stored in fact banks inside the organization boundary. To get fact data from a factbank I-actors shape into a forma shape in order to initiate a D-actor for retrieving the needed fact data. I-actors do not only reproduce existent facts but they also remember new facts which are created by B-actors. That means actually that the corresponding fact data have to be recorded in a fact bank within the organization boundary in order to be used later on by an I-actor for a specific infological action. Recording the fact data in a fact bank is done by a D-actor which is initiated by the I-actor which has been shaped into its forma form.

3 Prelude to Conceptual Modeling the B-, I- and D-Organization

3.1 Organizations Affect a Social World

An organization is understood to be a system which activities affect a social world. The notion of world is a very general one: there is a world of educating students; a world of manufacturing, transporting and delivering cars; a world of adjudicating suspects of criminal acts, etc. As an example, let us look at the world of manufacturing and delivering Ford cars; factories purchase raw materials, companies work on producing and assembling parts, others are busy in carrying raw materials and parts from one warehouse to the other and to distribution centers, a worldwide dealer channel sell new cars, etc. etc. Actually a large amount of companies cooperate and collaborate in a worldwide network and affect the state of the world of manufacturing and delivering Ford cars continuously. The state of this world changes by new original production facts which are created by actors within that network. Each purchase of a new part, each production fact in manufacturing a car, each transport fact leads to a state transition of this world.

Dietz [1] writes about a social world as an ‘universe of discourse’ that at any moment is in a particular state and which is defined as a set of facts. These facts are said to be current during the time the state prevails. The creation of a new fact leads to a state transition of the concerning world. A state change is called a transition. A transition is defined as an ordered pair of states, e.g. $T_1 = <S_1, S_2>$ is the transition from the state S_1 to the state S_2. The occurrence of transition is called an event. An event therefore can be defined as a pair $<T_1, t_1>$, where T_1 is a transition and t is a point in time. Transitions take place several times during the lifetime of a world; events however are unique: they take place only once. DEMO defines a social world by their ontological model. In other worlds, it defines a social world by a conceptual model that abstracts

from all implementation and realization issues. The ontological model is defined as a world consisting of the specification of its state space and its transition space. By state space is understood the set of allowable or lawful states and by transition space is understood the set of allowed or lawful sequences of transitions. The knowledge about the states and the state transitions of a world is called factual knowledge, which contrasts with procedural knowledge or know-how [1]. In the previous section we discerned two kinds of facts, viz. P-facts and C-facts. Every fact kind has its own world, so we talk about a C-world and a P-world.

Figure 4A exhibits an example of a specific actor network that operates on a world. The B-nodes (B1-B6) correspond with B-actors, the I-nodes (I1-I9) correspond with I-actors, and the D-nodes (D1-D9) correspond with D-actors. The links between nodes represent the transactions between actors. As you see in fig. 4A the actors are not brought together in organizations which operate on a shared world. However, it is impossible in practice to cover a world by one organization completely. Thousands of companies share the world of manufacturing and delivering Ford cars. Therefore, organization boundaries have to be drawn in the actor network to separate companies from each other (cf. fig. 4B). Considerations for splitting and merging organizations are still subject for research [9]. Figure 4B exhibits that actors are drawn inside as well as outside the organization boundary. Actors outside the organization boundary are called external actors; the others are called internal actors. The outside actors belong to a different organization. Although this organization shares our world they are

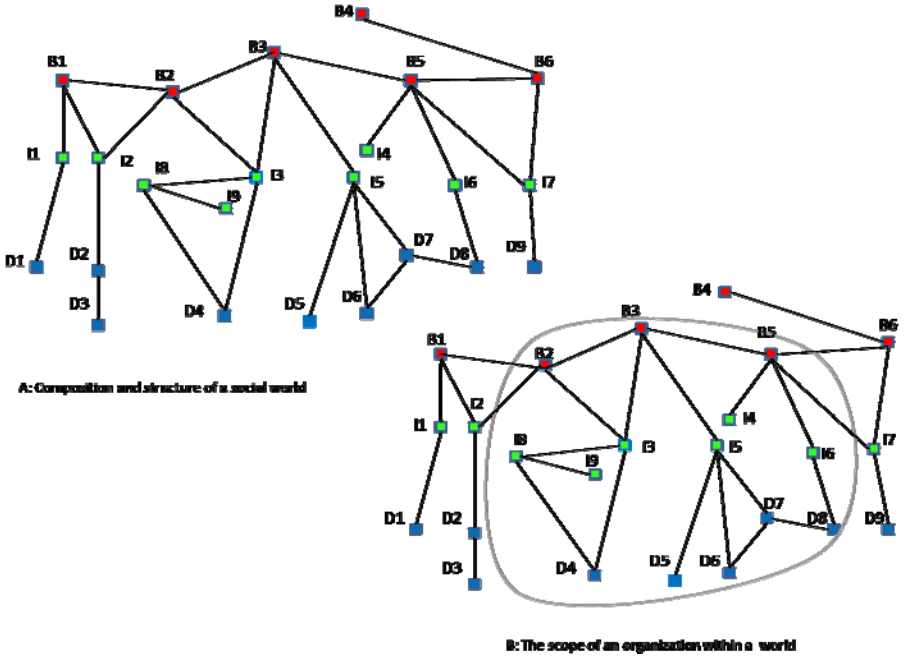

Fig. 4. Scoping the organization within a world

understood to be out of scope. The example by figure 4B exhibits that two links are defined between internal and external B-nodes, namely the link between node B1 and node B and the link between node B5 and node B6. By the corresponding transactions between the corresponding B-actors original facts are created. There are also defined links between internal B-nodes and external I-nodes, namely a link between nodes B2 and I2 and a link between nodes B5 and I7. Those links correspond with infological transactions between I-actors and B-actors whereby the underlying original facts are created by other organizations. New original facts which are created by B-actors inside the organization boundary (B2, B3, and B5) are always archived within this organization. Those facts are used as elementary building blocks for information production for B-actors inside or outside the organization boundary. The algorithms for the production of information are defined in the Object Property List, part of the DEMO state model of the organization. As fig. 4B exhibits the information for B2 and B5 is produced by another organization. Information production always takes place within the organization the concerning algorithm has been defined in the Object Property List.

3.2 Fact Handling through All Aspect Organizations

The essence of the organization is fixed by stakeholders and is fully determined by what they want to contribute to the environment of the organization. The contribution to the environment is provided by B-actors which are not able to operate effectively and efficiently without up-to-date information. According to the DEMO way of thinking information must be understood as supporting B-actors in their coordination acts [10]. For example, an executing B-actor does not perform the coordination act 'promise' before the ability to fulfill the request of the initiating B-actor has been checked. Another example is that the acceptance of a 'state' act cannot be done before the production result was delivered according to the conditions agreed upon. These rules, which link information to coordination acts, are modeled by the DEMO action model.

Let us discuss now the question how information comes into existence. All information which is requested by all B-actors affecting a world is derived from the factual knowledge of this world. Both the internal and the external world contain the factual knowledge which is needed for creating information for B-actors which are defined inside the organization boundary. B-actors, which affect the world we talk about, create new original P-facts which are archived in fact banks. A specific original fact is put into the fact bank that corresponds to the transaction kind by which it is created. An I-actor performs the intellectual action 'remembering' and D-actors perform the datalogical transactions 'transmit' and 'store' for actual storage. I-actors understand P-facts which have been stored in fact banks as elementary building blocks for deriving information in order to support B-actors. D-actors understand these P-facts as documents without any semantic meaning. For them unique identified packages of data without meaning in itself are kept in the fact bank. The meaning of a data package is remembered by an I-actor and must be reproduced by an I-actor if some other I-actors within the I-organization want to use the concerning fact for their infological actions. From the ontology point of view a fact exists and can never be disturbed or removed. It can only be copied for the benefit of actors within the I-organization. The D-organization provides copies of data packages that correspond with the requested

facts to I-actors in the I-organization for computing, reasoning, deducing, etc. At the end of the information delivery chain the derived fact is offered to the initial requester. The initial requester is always a B-actor in its informa ability. The B-actor extracts information from the offered derived fact in order to use the information in his conversations with other B-actors.

In summary, actors within the D-organization archive fact data into fact banks and collect fact data from fact banks. They are not conscious of the application of these data packages in intellectual actions which are performed within the I-organization. Actors within the I-organization remember facts, reproduce facts and perform intellectual actions on facts without any understanding of the meaning of the derived facts for B-actors. For example, the I-organization produces a monthly report with the turnover per product group. Actors within the I-organization have the competences and the authorities to construct such a report by making use of relevant facts which are collected by the D-organization. However, the I-actors do not have any idea about the added value of this report for the salesman who has asked for it.

For the right understanding of the exchange of production results between actors of the B-, I- and D-organization we look at the theory of semiotics [11, 12]. Actors which operate within a social system coordinate their actions with each other. So, any actor which needs some semantic meaning from another actor receives, according to the theory of semiotics, a sign. According to Stamper [13]: "business is getting things done by using information. All information is 'carried' by signs…" and "it must be a syntactically correct sign that is defined as anything that stands for something else for some community". Syntactically correct means that the sign is presented in a language that the receiver is able to understand. However, understanding a language is not the same as understanding the meaning of what is mentioned. The semiotic framework states that a sign has a semantic meaning. In other words, a sign has to be interpreted by the receiver to get a meaning or to get information. A sign becomes information when it can be used intentionally for certain purposes, such as communication.

Applied to the I-organization that supports the B-organization, I-actors do not offer information but only meaning or derived facts to B-actors. Finally, only the B-actors are able to determine of the offered semantic meaning is purposeful for them to act in its social world. In other words, I-actors do not deliver information but only original facts or derived facts. We discuss this notion in the next section more thoroughly.

4 Conceptual Modeling the B-, I- and D-Organization

4.1 The I-Organization Derived from the B-Organization

According to the Generic System Development Process model, GSDP for short [1, 8], the object system (OS) is designed and engineered from the using system (US) (cf. fig. 5). The development process of the I-organization can be considered as an instance of the GSDP; the B-organization is understood as the US and the I-organization is understood as the OS.

Let us discuss the use of GSDP more in detail. The most prominent system in the GSDP is the OS which is the system being designed, engineered and implemented. In

addition to the OS we discern a US. The US is the system that will use the functions or services offered by the OS, once it is operational. The development of a single homogeneous system of any type can be understood as an instance of the GSDP. GSDP uses the extended version of the system notion of Bunge [2]. The production of a system is defined as what is brought by the elements in the composition and transferred to the elements in the environment as the result of the interactions among the elements in the environment and the elements in the composition. Through its function, a system is able to support some other system, which uses the function. The function of the OS does not contain any information about the construction of the OS. The development of an OS consists of the phases design, engineering, and implementation.

In the first step, the functional requirements are determined. This step starts from the construction of the US and ends with the function of the OS. That has to be considered as a black box model that clarifies the behavior of the OS in terms of functional relationships between input and output of the OS. In this step the total service needed by actors of the US is analyzed. The services needed by the construction model of the B-organization are twofold. Firstly, services to remember new created C-facts and P-facts and secondly services for delivering facts and derived facts in order to support B-actors from the US with information.

In the second step, the specifications are devised. That starts with the specified function of the US-system and ends with the highest construction model of the OS-system. In our example the highest construction model of the I-organization. The total design of a system is understood as a process of alternate analysis and synthesis steps. An analysis step is one in which the problem is better understood; a synthesis step is one in which the solution becomes more clear. The construction model of the I-organization contains the construction of both defined services, viz. the construction for remembering C-facts and P-facts and the construction for reproducing original facts as well as the construction for determining derived facts, which are defined both inside and outside the organization, in order to support B-actors from the US with information. Derived facts are determined by performing intellectual actions, such as reasoning, calculating, etc. on original facts. Intellectual actions can be understood as

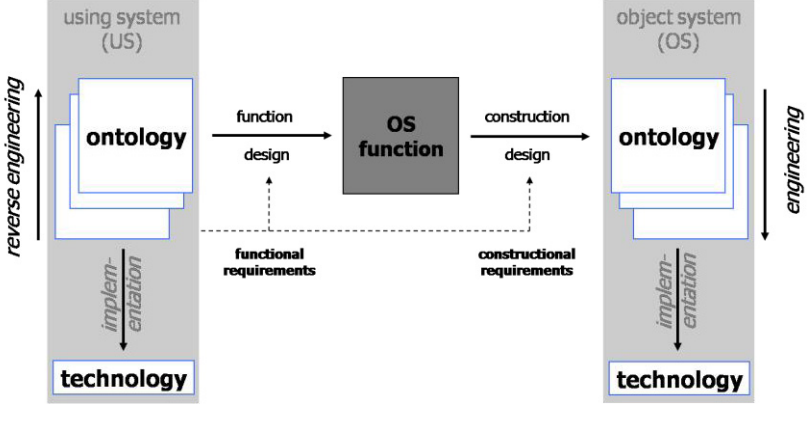

Fig. 5. Generic System Development Process

actions performed according to known procedures or algorithms. No original facts are generated by intellectual actions. A derived fact is not understood as a document or a sound or a picture but it must be understood as a part of semantic meaning.

In the third step, the engineering of the designed system takes place. The engineering of a system is the process in which a number of white box models, or construction models, are produced, such that every model is fully derivable from the previous one and the available specifications. Engineering starts from the ontological model and ends with the ultimate implementation model. In contrary to designing, engineering is not a matter of creativity but of craftsmanship. These white-box models clarify the internal construction and operation of the OS in terms of collaboration between its elements to deliver products to its environment. In case of the I-organization, source code of supporting software belongs to the ultimate implementation model of the I-organization.

During the fourth step, the implementation of the engineered model takes place. By implementation is understood the assignment of technological means to the elements in the implementation model, so that the system can be put into operation.

4.2 The Transaction Kinds between the B- and I-Organization

The highest construction model of the I-organization contains the construction of two service types. Firstly, the service type for remembering C-facts and P-facts. Secondly, the service type for delivering original facts as well as for delivering derived facts, which are defined both inside and outside the organization, in order to support B-actors with information.

Before we discuss the transaction types in detail it is important to understand two different matters. Firstly, a subject which fulfils actor roles always operates from one of three abilities, viz. the performa, informa or forma ability. A subject is also able to shape from any specific ability into any other ability. Secondly, according to the system theory of Bunge [14], transactions between actors only could happen between actors of the same category. Actually, a transaction between a B-actor and an I-actor is not correct. However, in which way does a B-actor receive the requested information? Let us discuss these issues step by step. Firstly, the subject who fulfills the B-actor role must shape into the informa ability. Secondly, the subject in its informa ability is only able to initiate a transaction with another I-actor as an external I-actor. The previous section exhibits that all subjects which fulfill B-actor roles and which need information fulfill external I-actor roles.

Figure 6 exhibits the transaction types which could happen in performing services from both service types. Figure 6.1 regards the first service type. The transactions in figure 6.1-6.4 regard the second service type. As said yet, a transaction between two actors of the same category is not correct. The first character of the actor code determines the actor kind, ‘B’ stands for B-actor, ‘I’ stands for I-actor and ‘D’ stands for D-actor. A short elaboration of each transaction kind is given below:

1. The subject that fulfils B-actor B-A01 shapes from its performa ability into its informa ability. It formulates a C-fact or a P-fact and initiates, as an external I-actor, a transaction with I-actor I-A01. The result of this transaction will be the remembering of an original C-fact or P-fact.
2. The subject that fulfils B-actor B-A01 shapes from its performa ability into its informa ability and initiates as an external I-actor a transaction with I-actor I-A02. The initiating subject asks for the reproduction of a original act.

3. The subject that fulfils B-actor B-A01 shapes from its performa ability into its informa ability and initiates as an external I-actor a transaction with I-actor I-A03. The initiating subject asks for a derived fact which is defined in the state model of the current organization.
4. The subject that fulfils B-actor B-A01 shapes from its performa ability into its informa ability and initiates as an external I-actor a transaction with I-actor I-A04. The initiating subject asks for a derived fact which is defined outside the current organization.

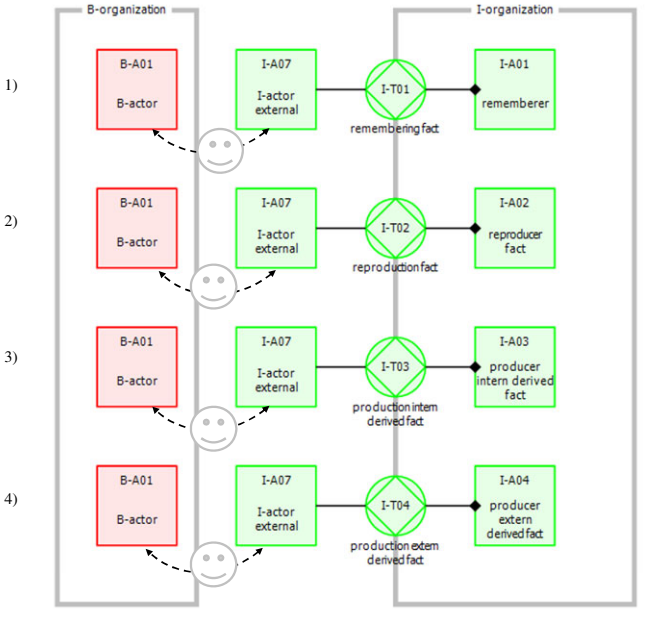

Fig. 6. The set of actor kind combinations between the B-organization and the I-organization

A B-actor in its forma ability is not able to initiate, as an external D-actor, a transaction with any D-actor directly. First, putting any document straight on into a fact bank is not possible. Some intelligent actions have to be executed in advance, e.g. the determination of the store. Such action is executed by an I-actor. Second, getting a document directly out of a fact bank is also impossible. Before getting this document you have to know where it has been stored. Finding its store can only be done by an intelligent action executed by an I-actor. Summarizing, remembering and reproduction of documents must always be done by the I-organization. There is no direct link between de B-organization and the D-organization.

4.3 The D-Organization Derived from the I-Organization

The development process of the D-organization can be considered as an instance of the GSDP (cf. fig. 5). The I-organization can be understood as the US and the

D-organization can be understood as the OS. Let us discuss the use of GSDP for designing the highest construction model of the D-organization more in detail.

In the first step, the functional requirements are determined. This step starts from the construction of the US and ends with the function of the OS. In this step the total service needed by actors of the US is analyzed. The services needed by the construction model of the I-organization are twofold. Firstly, services to archive new created C-facts and P-facts to remember and secondly services for collecting fact data which corresponds with the fact to reproduce.

In the second step, the specifications are devised. That starts with the specified function of the US-system and ends with the highest construction model of the OS-system. The construction model of the D-organization contains the construction of both defined services, viz. the construction for archiving C-facts and P-facts and the construction for collecting fact data in order to support reproducing I-actors with the corresponding facts. D-actors are working together in order to support I-actors for documental actions like storage, retrieval and transmission.

We discussed in the previous section that original facts must be understood as elementary building blocks for information. Based on the building blocks intellectual actions are performed in order to get new semantic meaning or derived facts. However, derived facts do not only have a meaning but they also have a form. This form could be, for example, a written text, a picture, a sound, a gesture, a touch. In brief, communication between people can be done by all our senses. It can be done by several types of documents. The D-organization in an organization contains all actors which produce actions on documents in order to support actors from the I-organization. Those actions are called datalogical actions.

4.4 The Transaction Kinds between the I- and D-Organization

The construction model of the D-organization contains the construction of two service types. Firstly, the service type for archiving new created C-facts and P-facts to remember and secondly the service type for collecting fact data which corresponds with the fact to reproduce.

Before we discuss the transaction types in detail we have to know that a transaction between an I–actor and a D-actor cannot happen. Therefore, the initiator of the transaction must first shape into the forma ability before it is able to initiate a transaction as an external D-actor with another D-actor within the D-organization.

Figure 7 exhibits the transaction types which could happen in performing services from both service types. The first character of the actor code determines the actor kind, ‘B’ stands for B-actor, ‘I’ stands for I-actor and ‘D’ stands for D-actor.

A short elaboration of the each transaction kind is given below:

1. The subject that fulfils I-actor I-A01 shapes from its informa ability into its forma ability and initiates as an external D-actor a transaction with D-actor D-A01. The initiating subject requests for archiving a package of data that corresponds with the fact to remember.
2. The subject that fulfils I-actor I-A02 shapes from its informa ability into its forma ability and initiates as an external D-actor a transaction with D-actor D-A02. The initiating subject asks for collecting a package of data that corresponds with the fact to reproduce.

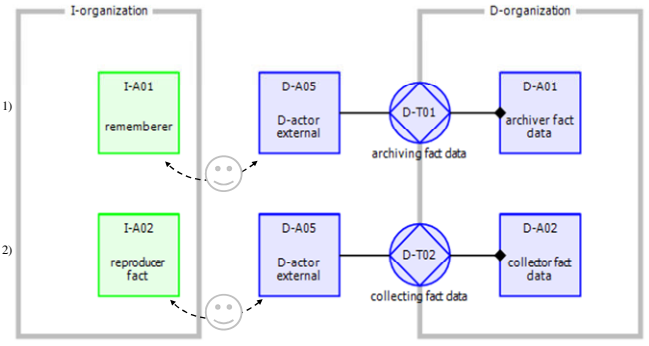

Fig. 7. The set of actor kind combinations in the D-organization

4.5 Archiving and Collecting Information

The information delivery chain can be divided in two kinds. The first kind contains the steps that a new fact takes from the start of its existence until its storage into a fact bank (cf. fig. 8). Such a fact can be understood as an elementary building block for the production of information. The second kind describes the way back from retrieving the corresponding fact data from the fact bank, producing a derived fact based on original fact data and finally the offering of the derived fact as information to a B-actor that has requested for it (cf. fig. 9).

The first kind of the information delivery chain starts with the B-actor B-A02. B-A02 creates a new original fact, for example the purchase of a car. After acceptance of the P-fact by B-A01 the actual P-fact starts to exist. That P-fact must be stored into the fact bank that corresponds with B-T02. B-A02 shapes into its informa ability, formulates the P-fact and initiates, as an external I-actor of the I-organization, the transaction I-T01 with I-actor I-A01 within the I-organization for remembering the P-fact. I-actors perform infological actions according to algorithms or procedures. I-A01, called rememberer, adds an identification to the P-fact and determines its storage place. Although the P-fact has to be understood as semantic meaning, it always must have a carrier. The carrier itself is fully irrelevant for I-actors. It could be a paper document, a digital document, sound waves, a picture or anything else. Actors within the I-organization and between the B-organization and the I-organization only exchange semantic meaning. Therefore those actors operate in their informa ability.

For actual remembering the P-fact the rememberer I-A01 shapes into its forma ability and requests, as an external D-actor of the D-organization, the archiver D-A01 to archive the corresponding fact data of the P-fact. During those steps the semantic meaning of the document is irrelevant, only the form of the document is important. Subsequently, D-A01 initiates the transaction to activate the transmitter for transporting the document to its destination and to activate the storer for storing the document.

The discussion about the first kind of the information delivery chain ends with two examples. The first example illustrates the situation that every actor role in figure 8 is performed by a different subject. The second example illustrates the situation that all actor roles are performed by the same subject.

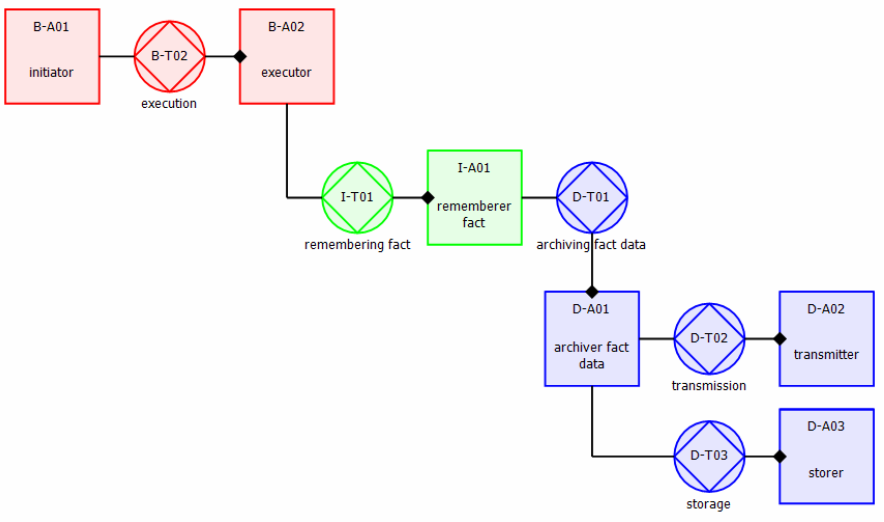

Fig. 8. Archiving P-facts

Firstly, an aspirant member John (B-A01) asks Bill (B-A02) for a membership of the library. Bill promises John to register his membership. Bill formulates the registration based on some information that he receives from John. Subsequently, he asks Tom (I-A01) for remembering the registration. Tom promises that he will remember the requested membership registration. Although he knows that the complete registration is a new original fact, he needs an identifier for finding the registration later back in a data store. Tom determines the identifier and links it to the registration. From now the registration is identified by an identifier and has to be preserved on a safe place. Next, based on the identifier Tom determines a safe store and asks Jim for archiving the unique defined data package in that store. Jim is willing to perform this request and asks for a transmitter and a storer to bring it to the given store. After archiving has been performed Tom receives a state from Jim. Tom accepts that archiving has been performed and sends a state to Bill that remembering has been arranged correctly. After accepting this state Bill sends a state to John that the registration has been performed. After acceptance by John the registration really exists. If John rejects the registration a roll-back of the previous steps has to be done.

Secondly, assume that all steps of the registration are performed by Bill. Bill formulates the registration and remembers that registration in his mind or by storing is on a safe place. That could be in a card index on his desk, transmitting is not necessary, or in a card index in another room, transmitting is necessary.

The second kind describes the way back from retrieving the corresponding fact data from the fact bank, producing a derived fact and finally offering it to a B-actor that has requested for it. Figure 9 exhibits the second kind of the chain. The initiative for retrieving is indirectly taken by a B-actor which needs information for being active in conversations with its connected B-actors.

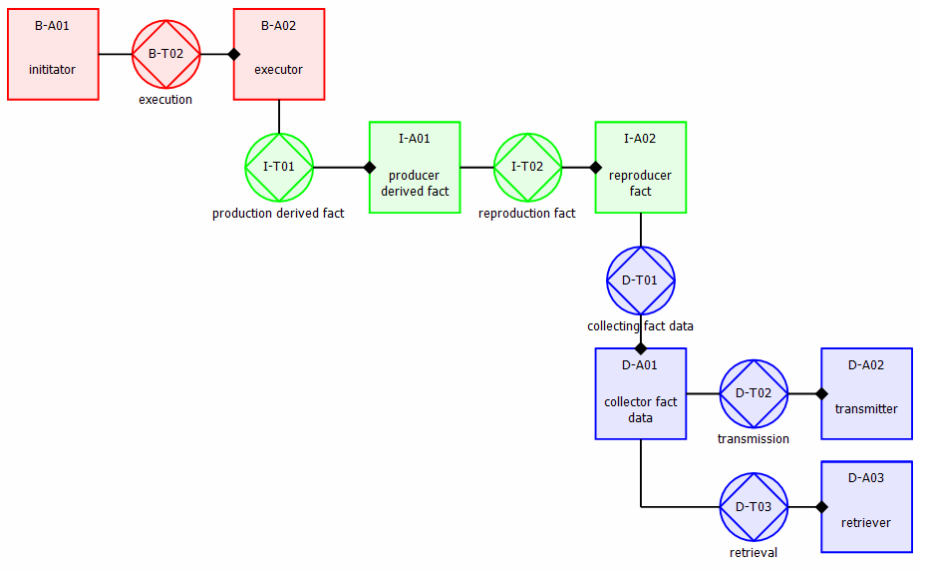

Fig. 9. Delivery of derived facts

It starts with a B-actor B-A02. B-A02 shapes into the informa ability and asks for information it needs. Two kinds of factual information processes are discerned. Firstly, the process of determining a derived fact is discerned. That derived fact is derived by infological actions from elementary facts. This process corresponds with an internal information link within the DEMO construction model of the organization. Secondly, the process of determining an external fact is discerned. That external fact is determined outside the organization boundary. It could be an elementary fact or a derived fact. This process corresponds with an external information link within the DEMO construction model of the organization.

Let us first discuss the first kind of information process that delivers derived facts. B-A02 initiates a transaction, as an external I-actor of the I-organization, with the producer of a derived fact I-A01. Based on original facts which are remembered by the I-organization, I-A01 derives a new fact that is defined in the Object Property List of the DEMO state model of the organization. I-A01 asks I-A02 to reproduce the needed original facts. However, I-A02 is only able to reproduce if it knows the unique identification and storage place of the corresponding document in the data store. If the I-actor asks the D-actor, for example, for a specific contract regarding the delivery of product X by supplier Y, the transaction between both actors will not succeed, because both actors do not understand each other. The transaction will only be successful if the I-actor asks for the unique identified document stored on the unique identified storage place. Now the question comes up: how does I-A02 know the identification of the corresponding data package and the identification of the storage place? We discussed already that the rememberer (cf. fig. 8) in the I-organization determines the unique identification and the storage place of new facts. Regarding this question, there is only one answer possible: the subject that fulfills the actor role

I-A02 in figure 9 must be equal to the subject that remembers the concerning fact in figure 7. For collecting the needed fact I-A02 shapes into its forma ability and asks for, as an external D-actor of the D-organization, the fact data collector D-A01 for collecting the identified data package from the given store. Actually, D-A01 collects the needed data package by initiating a transmission and a retrieving transaction with the transmitter D-A02 and the retriever D-A03, respectively.

The discussion about the second kind of the information delivery chain also ends with two examples. The first example illustrates the situation that every actor role in figure 9 is performed by a different subject. The second example illustrates the situation that all actor roles are performed by the same subject.

Firstly, Mary (B-A02) registers loans of books in the library. When someone asks her for a loan registration she checks first if the number of books, which has been lent already by the requester, does not exceed a maximum allowed number of books. She asks Kim (I-A01) for providing her this information. Kim promises Mary to provide her the requested information; she is able to fulfill her promise because of the availability of an algorithm for computing the corresponding derived fact. The algorithm works on original facts which have been created by previous production acts and which are remembered within the I-organization. Before computing the mentioned algorithm Kim asks Tom (I-A02) one or more times to reproduce an original fact. Tom promises that he will reproduce the requested fact. He knows the unique identification as well as the storage place of the fact data that corresponds with the fact because he has determined the identifier and the storage place for remembering this fact earlier. Next, Tom asks Jim to collect the unique defined data package. Jim is willing to perform this request and asks a transmitter and a retriever to deliver the fact data. After collecting has been performed Tom receives a state from Jim. Subsequently, Tom sends a state to Kim that the requested fact has been reproduced. When all requested facts are available Kim computes the algorithm and sends a state to Mary that the requested information has been produced.

Secondly, assume that all steps of the registration are performed by Mary. Mary computes the information she needs. She collects all original facts from the places where she has stored these facts earlier.

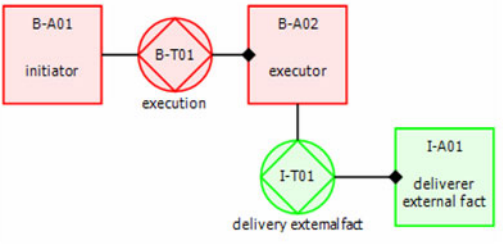

Fig. 10. Delivery of external facts

Figure 9 exhibits the presence of merely one I-actor for calculating the derived fact. However, in practice the derived fact can be calculated by several I-actors, each with their own responsibility and authorities. This is understood as a granularity issue which is not a subject of this paper.

The second kind of information process that delivers external facts is exhibited in figure 10. B-A02 initiates a transaction, as an external I-actor of the I-organization, with the deliverer of an external fact I-A01. That external fact is determined outside the organization boundary completely. The information process belongs to another organization. The operation between both organizations is elaborated in the implementation model of the organization.

5 Conclusions and Further Research

The distinguishing feature of this paper for the DEMO community is that the focus of current organization engineering research is extended to the B-organization, the I-organization and the D-organization. It explains the realization of the organization by discussing the way of cooperation between B-actors, I-actors and D-actors within the same world and more specific within the same organization. The structure of the complete information delivery chain, from the creation of an original production fact until the delivery of a derived fact that intentionally can be used for certain purposes, is elaborated from the ontological perspective. We understand the drawing of an organization scope within a world as the first step in implementing an operational organization. In our opinion, scope discussions are sourcing discussions and for that reason also implementation discussions.

Further research has to be done about the cooperation of several organizations working on different worlds and cooperating within the same enterprise. For example, cooperation with information suppliers, human resource recruiters, accountancy firms, asset management organizations. Other subjects for further research are the granularity of the I- and D- organization and the link with supporting IT-applications.

References

1. Dietz, J.L.G.: Enterprise Ontology – theory and methodology. Springer, Heidelberg (2006)
2. Dietz, J.L.G.: The deep structure of business processes. Communications of the ACM 49(5), 59–64 (2006)
3. Dietz, J.L.G.: Deriving Use Cases from Business Process Models. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 131–143. Springer, Heidelberg (2003)
4. Maij, E., et al.: Use cases and DEMO: aligning functional features of ICT-infrastructure to business processes. International Journal of Medical Informatics 65, 179–191 (2002)
5. Shishkov, B., Dietz, J.L.G.: Deriving use cases from business processes: the advantages of DEMO. In: The Fifth International Conference on Enterprise Information Systems, Angers, France (2003)
6. Mallens, P.J.M., Dietz, J.L.G., Hommes, B.J.: The Value of Business Process Modelling with DEMO prior to Information Systems Modeling with UML. In: EMMSAD 2001: Proceedings, 6th CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (2001); Interlaken
7. Mulder, J.B.F.: Rapid Enterprise Design, Technical University Delft, Delft (2006)
8. Dietz, J.L.G.: Architecture, building strategy into design. In: NAF working group Extensible Architecture Framework (xAF) (2008)

9. Op 't Land, M., Applying Architecture and Ontology to the Splitting and Allying of Enterprises. Delft University of Technology, Delft (2008)
10. Dietz, J.L.G.: DEMO: Towards a discipline of organisation engineering. European Journal of Operational Research 128, 351–363 (2001)
11. Stamper, R.K.: Information in Business and Administrative Systems. Wiley, New York (1973)
12. Liu, K.: Semiotics in Information Systems Engineering. Cambridge University Press, Cambridge (2000)
13. Stamper, R.K., et al.: Understanding the Roles of Signs and Norms in Organisations. Journal of Behaviour and Information Technology 19(1), 15–27 (2000)
14. Bunge, M.A.: Treatise on Basic Philosophy, A World of Systems, vol. 4. D. Reidel Publishing Company, Dordrecht (1979)

A Bottom-Up Competency Modeling Approach

João Marques[2], Marielba Zacarias[1,3], and José Tribolet[2,3]

[1] Universidad do Algarve, Portugal
[2] Instituto Superior Técnico of Lisbon, Portugal
[3] Organizational Engineering Center (CEO)
jmt@inesc.pt, mazacaria@ualg.pt, jose.tribolet@ist.utl.pt

Abstract. Competency-based management i.e. linking and assigning work according to the competencies required, has been acknowledged as a key enabler of organizational effectiveness and highlights the importance of modeling organizational competencies. Whereas Enterprise Architectures (EA) model organizations from different perspectives that describe and inter-relate their processes and resources, most EA frameworks provide limited means to model competencies. Available competency modeling approaches rely on static classification models that prove difficult to adjust to the constant changes in competency requirements. We propose a bottom-up modeling approach for representing organizational competencies using semantic units such as actions, goals and resources that (1) provides a more flexible competency classification framework, (2) allows answering questions about different competency-related concerns, and (3) allows assessing the alignment between the competencies required by activities and the competencies held by the actors performing them. This paper describes the competency model proposed and a set of tools used in supporting the model building process. Some benefits of the model are illustrated with a case study in an organizational setting.

1 Introduction

Enterprise Architectures (EA) are effective means to communicate the organization's structure, processes and goals. EA are widely used for systems development or process (re)design ends, and allow modeling organizations from different but interconnected perspectives, where most commonly depicted perspective encompass strategy, process, information, application and technology-related perspectives [1]. EA focus on business processes results in provisioning of limited means to model human competencies, of great relevance for competency-based approaches of Human Resource Management (HRM). HRM activities play a major role in ensuring the organization's survival and prosperity [2,3,4]. A firm's current and potential human resources are important considerations in the development and execution of its strategic plan, and nurturing "brainpower" plays a fundamental role as a source of competitive advantage. As a result, it is not difficult to understand why competencies are becoming a central theme in HRM practices. In *competency-based management*, job outputs are linked with human competencies as an approach for assigning work and producing work outputs.

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 50–64, 2010.

We define competencies as *capabilities* made accessible in the form of human actions and resources provided to the organization. Organizational activities require a number of *competencies* held by human actors. The main motivation of this research is developing means to capture and model the relationships created between human actors and activities by describing on one side, the competencies required by activities and on the other, the competencies provided by human actors . Organizations must be able to dynamically (re)schedule human resources according to the actual competency requirements of its activities. The more knowledge-oriented the activity, the more important it becomes describing the competencies required by the activity and the human actors that hold such competencies. However, current business environments are constantly changing, and so are competency requirements. Whereas there is a large amount of research on competency modeling in the area of knowledge representation, most approaches rely on static classification models that prove difficult to adjust constant changes [5]. Moreover, the process of capturing and describing is largely manual, and cannot keep up the pace of such organizational changes. Hence, two important questions remain unanswered:

1. *How to capture and model information related to organizational competency-related, so they can be defined and represented in a more flexible manner under a constantly changing business environment?*
2. *How to express the alignment between the competency required by activities and the competencies held by organizational actors?*

Our research addresses these questions through a bottom-up modeling approach for representing organizational competencies through a combined use of semantic units such as actions, resources, and goals that (1) provides a more flexible competency classification framework, (2) allows answering questions about different competency-related concerns, and (3) is linked to process-related concepts in order to allow assessing the alignment between the competencies required by activities and the competencies held by the actors performing them. This paper describes the competency model proposed and a set of tools used in supporting the model building process. The remainder of this paper is structured as follows; section 2 summarizes related work on EA, business process modeling and competency modeling approaches. Section 3 describes the proposed model. Section 4 illustrates some model benefits with a case study. Section 5 summarizes the tools used in building the model. Section 6 gives our conclusions and outlook.

2 Related Work

Most Business Process Modeling (BPM) approaches and EA frameworks do not provide means of relating human actors and activities through competencies. Integrated DEFinition Methods (IDEF) [6] defines concepts such as *function, input*, and *output* but does not provide means for representing how activities are carried out by actors. The Business Process Modeling Notation(BPMN) [7] focuses on describing the activity flows within business processes. The diagram elements

include *activities, flow and connecting objects, artifacts*, and *swimlanes*. The latter are used to associate functional capabilities or responsibilities to a particular actor. However, such elements are not well suited to model the competencies required to perform the set of activities included within each swimlane, since responsibilities are represented at very high levels of abstraction. Other modeling languages, such as IDEF3 [8] or Role Activity Diagram (RAD) [9] focuses on describing process flow but they neither specify the behavior of human or automated actors, nor the required competencies for each activity. The business process modeling methodology DEMO [10] also lacks competency modeling means.

Enterprise Architecture (EA) frameworks allow to represent organizations from several, but inter-related perspectives. The most commonly depicted enterprise perspectives are the *process*, *information*, *application*, and *technology* perspectives [1]. Whereas the former describes enterprise *activities* i.e. what organizations do, the remaining perspectives describe its *resources* i.e. the entities required for their operation. Though these frameworks, allow resource-related perspectives where competencies could be included, they are mostly process-driven approaches that tend to disregard the modeling of competencies. The Open Group Architecture Framework (TOGAF) [11] and the Integrated Architecture Framework IAF [12] are industry standard architecture frameworks. While both include *business, information systems*, and *technology infrastructure* architectures, they overlook the relationship between actor and activities, since they do not include competency-related concepts within the business architecture. The Zachman framework [13] is a generic enterprise classification framework that comprises *functional (activity)* and *organizational (actor)* dimensions but does not specify how to represent each dimension or how to relate them.

Two EA frameworks better suited for specifying actor-activity linkages are Archimate [14] and the CEO framework [15]. Archimate presents a unified way of modeling EA, in which the concept of *service* plays a fundamental role in connecting *business processes* and *business functions*. While the former is a collection of causally related units of internal behavior such as *activities*, the latter allows to group behavior according to, for instance, required *skills* or *knowledge*. The CEO framework presents three constructs for organizational modeling: *entity, role*, and *activity*. Activities describe how entities collaborate through roles to produce specific outcomes, where roles represent sets of *services*. Such relation can be modeled as a marketplace, in which activities demand roles and actors offer them. With this approach, it is possible to select the actors that provide the roles required by a given activity. Nonetheless, in spite of providing means of linking activities and actors through competencies, these frameworks lack proper means of modeling competencies, their components and inter-dependencies.

There is a large amount of research on competency modeling in the area of knowledge representation. Tarasov [16] proposes the use of ontologies to formally represent individual and enterprise competencies where individual and enterprise models share a common conceptual core that simplifies competency matching and gap analysis. Gronau [17] presents the Knowledge Management Description Language (KMDL) to manage skill catalogues, which unlike common business

process modeling languages, allows to model the creation, use and need of knowledge in common processes. KMDL provides an object library containing *information, task, position, requirements, person, knowledge object*, and *knowledge descriptor* concepts. CommOnCv [18], which aims at dealing with the problematics of e-recruitment, models competencies as a set of *resources* mobilized to reach an *objective*, or to carry out a *mission* within a particular *context*. The same author presents the *Competency, Resource, Aspect*, and *Individual* (CRAI) model in [19], which structures and formalizes the concept of competency, and provides guidelines for its deployment in competency management information systems. Overbeek [20] proposes a knowledge market paradigm as a means of improving the fit between the demand and supply of knowledge through an approach to match actors and tasks based on the cognitive characteristics provided by actors that are required by specific types of tasks.

Competencies are frequently represented as hierarchical structures of competency trees. Lang and Pigneur [21] propose using a standard vocabulary displayed in a tree to describe the competencies held by actors instead of textual descriptions, as a means to simplify competency processing. Varela and Soares [22] propose a competency tree with dynamically defined upper categories. Although structuring competencies in trees simplify their management, they present some other problems related to flexibility and reusability. Since trees are usually deep, the description of the competency is not easily decomposable, leading to a dispersion of concepts between multiple nodes. Moreover, reusing competencies in different contexts is hindered as broader functional categorization is embedded at the tree's upper levels. To overcome such shortcomings, [5] proposes a framework for competency modeling based on a multi-dimensional tree structure, where each dimension relates the element with its usage context, allowing the creation of multiple views on the same object while keeping its uniqueness. Nodes in the same dimension are connected using intra-dimension links showing allowing to group elementary concepts in coarser-grained concepts. Inter-dimension links associate nodes from different dimensions, allowing to represent competencies through a multi-dimensional composition of concepts.

3 Competency Model Proposed

The competency model presented here allows a formal representation of competencies held by organizational actors and required by organizational activities. It presupposes that a competency involves a *capability* associated with a given *subject matter* in order to reach an *objective*. Our model combines the aforementioned multi-dimensional competency tree structure with the meta-model for organizational modeling put forward by [23]. Aveiro's meta-model is a fact-oriented modeling approach based on description logics and Object-Role Modeling (ORM) [24], and it is fully described in terms of *entities* and *relationships*. An *entity* represents things that have physical interest or virtual existence that is of interest to be modeled. Entities are described with *nouns* (or noun phrases). A *relation* captures relations between entities, and is itself an entity. It is expressed as a <*Subject* - ***Predicate*** - *Object*> triple, where *Subjects* and

Objects are *entities* while **Predicates** define the particular type of relation between them. **Predicates** are expressed with verbs.

The model is built upon the following entities: (1) Competency, (2) Task, (3) Goal, (4) Action, and (5) Resource. Such entities define the competency concept in a general way. *Competency* expresses the set of Knowledge, Skills, and Attitudes (KSA) provided by *Actors* when playing a given *role* to carry out a formal organizational *activity*. Examples of competencies include: Program SQL, Administrate Unix System, or Implement ITIL.

KSA are modeled in terms of *actions* and *resources* associated to $<Task, Goal>$ pairs. A *Task* is the unit of work undertaken by an *Actor* with a particular *Competency*. The relationship *In Order To* represents the cause-effect relationship between the *Task* and its *Goal*. In other words, it represents the competency *purpose*. Our model decouples *Task* from its Goal, allowing them to be reused in different and unrelated competencies. Goal expresses a state or condition to be brought about or sustained through a *Task*. Examples include: Satisfy client expectations, Report data from source systems, Ensure design specifications.

Tasks are described through *Action-Resource* relationships. *Action* expresses the type of KSA brought into play by an actor in order to carry out a *task*. *Actions* may be mental, interpersonal or physical. *Mental Actions* correspond to such data-related operations as Analyze or Understand. *Interpersonal Actions* correspond to people-oriented operations like Ask or Interview. *Physical Actions* correspond to things-related operations like Make, Write or Develop. *Resource* comprises the physical or abstract entities on which actions operate, and are identified with nouns (or noun phrases). Some examples include: development tool, SQL, Unix Operating System, and databases. *Resources* may range between single items to very complex resources composed of several items. By decoupling Action and Resources, our model allows their re-use in different competencies. For instance, *Task* Write SQL queries is described by *Action* Write and *Resource* SQL queries, each of which can be present in other competencies.

Figure 1 depicts the model entities and relations and shows that competencies and resources can be organized in a tri-dimensional structure composed by (1) actions, (2) resources and (3) goals. These dimensions allows to interconnect competency to different EA views. Figure 1 illustrates how the proposed competency model is related to organization and business views. The organization view describes who are the organizational *actors* and the *roles* played by such actors. *Actors* provide *competencies* to *activities* through the *roles* they play in executing such activities. The business view describe the *activities* composing *business processes*, their *goals, inputs/outputs*, as well as their coordination mechanisms. *Activities* have *goals* formally acknowledged by the organization. The purpose of *competencies* is to help achieve activity *goals*. Section 3.1 describe the model relations and data types.

3.1 Relations and Data Types

The model defines two data types (1) *verbs* and (2) *nouns*. The *Verb* type is used to express the part of the competency statement referring to an action, or

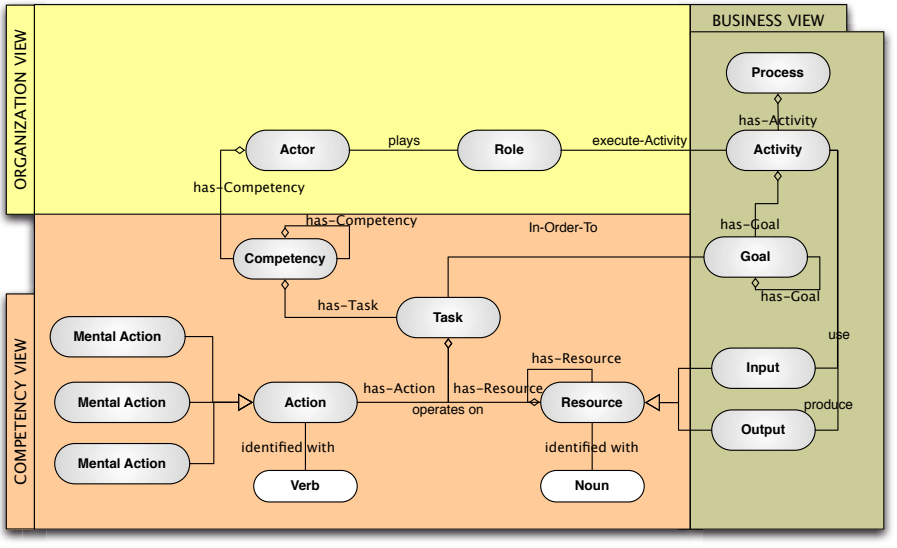

Fig. 1. Competency Model

capability. *Verbs* may describe mental actions, like "to know", and "to decide"; interpersonal actions, like "to ask", and "to interview"; physical actions, like "to write", and "to drive"; or any combination of these three types, as in [25]. The *Noun* type is used to express the part of the competency statement referring to the subject or object of a verb. *Nouns* may vary from broader concepts like "business applications" or "operating system", to specific ones like "SAP" or "Linux". It is expected that both Verb and Noun types may be used on the definition of unrelated competencies. Examples of such expressions are "to code a bubble-sort in C" and "to code a Web-service in Java". To cope with the representation of such features, the *Verb-Noun* type allows them to be uniformly reused while defining different competencies. It is also useful to show information of the capabilities related each subject, and vice-versa. Hence, we reify verb-noun as a predicate to define the following relations:

- *Verb-Noun* - ***Has*** - *Verb*
- *Verb-Noun* - ***Has*** - *Noun*

Actions are identified with verbs. To formally model actions, the following relation is defined: *Action* - ***Has*** - *Verb*.

Goal expresses the competency end-result. It corresponds to a mission to carry out, as in goal modeling literature. *In Order To* expresses the causal-relationship between the competency and its end-result. It is useful to show information of which Task is related to a Goal, and vice-versa. We reified *In Order To* as an entity, to define its relations with the *Task* and *Goal* entities:

- *In Order To* - ***Has*** - *Task*
- *In Order To* - ***Has*** - *Goal*

Goals are organized in hierarchies where Task goals are sub-goals of activities. However, such hierarchy is not reflected here since it is part of the business view.

Task refers to a particular unit of work reflecting described in terms of a particular *Action* and an associated resource, and reflects a particular KSA. The meaning of a *Task* is thus defined by an <*Action,Resource*> pair. Such meaning is formally expressed with the relations below. We first reify the <*Action,Resource*> relationship to define its relations. Then we associate *Task* with the reified relationship. Resources are organized in hierarchies. Top-level resources refer to broad *subject matters*.

- *Action-Resource* - ***Has*** - *Action*
- *Action-Resource* - ***Has*** - *Resource*
- *Task* - ***Has*** - *Action-Resource*
- *Resource* - ***Has*** - *Resource*

Finally, the meaning of a *Competency*, is given by a *Task*, its Goal, and both higher and lower-level competencies. The following relations are defined in the perspective of a Competency:

- *Competency* - ***Has*** - *Task*
- *Competency* - ***Has*** - *In Order To*
- *Competency* - ***Has*** - *Competency*
- *Competency* - ***Is-Part-Of*** - *Actor*

The last relation is extrinsic to the entity Competency since it is the inverse of the relation *Has*, intrinsic to the entity *Actor* (defined in the organization view).

4 Case Study

This section describes a case study undertaken to assist in the design and evaluation of the competency model proposed in section 3, through an *Action Research* methodology. *Action research* is applied research. It happens where the researcher is allied to the group under study and attempts to provide practical contributions while developing theoretical knowledge. It produces practical and theoretical outcomes, frequently emancipatory outcomes and makes explicit the researcher biases [26]. The research took place in a Portuguese professional services company, which will be from this point forward refer to as *Customer*. The Customer provides audit, consulting, financial advisory, and risk management services. The research has been developed in the Customer's Technology Integration (TI) area.

The Customer follows a framework for assessing performance and career development named Consulting Global Excellence Model, which is organized in shared and specialization components. The shared component contains common development and performance themes which apply to all practitioners. These themes are grouped into four common areas: *Service Excellence*, *Marketing, Sales & Communications*, *Management Effectiveness*, and *Leadership Effectiveness*.

The *Specialization* component contains specific development and performance themes, which apply to practitioners of each service area. There are currently six specializations: *Enterprise Applications*, *Human Capital*, *Strategy & Operations*, *Outsourcing*, *Project Controllers*, and *Technology Integration (TI)*. *Competencies* refer to the set of KSA that practitioners need to develop.

The research setting involved two organizations: the Center for Organizational Engineering (CEO) of Lisbon, and the Customer's TI area. The research team included a student pursuing his master degree at the CEO while working at the Customer's TI area, and practitioners with experience in competency-based management. It also included two CEO researchers acting as counselors. In the team's first meeting, they agreed to select the TI area due to (1) the heavy workload of most areas at the moment of the study, (2) IT culture where people are more likely to try new and innovative approaches, and (3) the student' knowledge of the setting.

The Customer's TI area adopts a top-down approach for classifying expected career level competencies in static competency clusters and themes. Individual competencies were related to specific roles. For example, the *The Business and IT Strategy* cluster encompasses four themes; (1) Technology Awareness in Clients' Business/Industry, (2) Technology Drivers/Enablers in Clients' Business/Industry, (3) Ascertains Clients' Technological Needs, and (4) IT Strategy Development. Each theme defines a set of individual competencies for the roles *Analyst, Consultant, Senior Consultant, Manager, Senior Manager*, and *Partner/Director*. The list below shows the competencies defined for each of these roles for the IT Strategy Development theme.

- *Analyst*: Assists in the development of specific components of the IT strategy.
- *Consultant:* Assists in the development of specific components of the IT strategy.
- *Senior Consultant:* Develops implementation plans for specific components of the IT strategy (e.g., work plan, transitional architecture plans)
- *Manager:* Designs, develops, and validates new IT strategies for clients and develops IT stragegy implementation plans.
- *Senior Manager:* Gains consensus for IT strategy implementation plans
- *Partner Director:* Supports IT strategy implementation plans with the client's top leaders.

Our plan was to evaluate our approach in capturing and representing competency's building blocks, namely actions and resources, to define distinct isolated competencies. Following a bottom-up approach, these competencies would then be composed into coherent sets, so they could be bound to employees and activities, as well as the Customer's predefined competency themes and clusters. This section shows results on capturing individual competencies and the latter type of binding. The results were gathered using the following method: (1) Initial action and resource set definition, (2) task capture, (3) manual maintenance and review of new actions and resources; and (4) Representing these elements and their inter-relationship.

Initial action and resource definition. As seen before, similar actions are typically described using different verbs, and the same verb may be used in describing different actions. For instance, both appraise and evaluate verbs are related with assessing the value of something. Likewise, similar resource items may be described

using different nouns, and the same noun may be used in describing different resource items. For instance, SAS may refer to **SAS Enterprise Guide**® or **SAS Enterprise Miner**®. Hence, a basic set of action types and resources and their meanings, has been discussed and validated by the practitioners. Following the recommendations on job analysis in [25], the use of specific verbs instead of summary verbs has proved easier to achieve consensus around meanings. For example, achieving consensus around the meaning of the action build was easier than discussing the meaning of the action consult. The initial action and resource set was collected from support documentation where competencies were described as illustrated in the list above. Here, actions and resources have been highlighted to show how they were extracted. Resource nouns were collected without qualifiers (e.g. *complex* technical documentation, *quality* deliverables, *complex* client processes). This action ended with 52 distinct action types, and 223 resources identify regarding the TI universe.

Maintenance and review of new elements entails identifying and eliminating synonyms. It was also verified that each element conveyed the same meaning to the participants.

From the 52 distinct action types collected from the support documentation, the most recurrent were physical and mental actions: develop, understand, identify, manage, and ensure (see Table 1). This makes sense due to the area of the

Table 1. Predominant Action Type Set

Action	Meaning	Type	Freq.
Develop	acquire, arise, break, build up, educate, evolve, explicate, formulate, germinate, get, grow, make grow, modernise, modernize, originate, prepare, produce, recrudesce, rise, spring up, train, and uprise.	Physical	8,98%
Understand	empathise, empathize, infer, interpret, read, realise, realize, see, sympathise, sympathize, and translate.	Mental	7,42%
Identify	describe, discover, distinguish, key, key out, name, and place.	Mental	7,03%
Manage	bring off, care, carry off, contend, cope, deal, do, finagle, get by, grapple, handle, make do, make out, ne	Mental	6,25%
Ensure	ascertain, assure, check, control, guarantee, insure, secure, see, and see to it.	Mental	4,69%

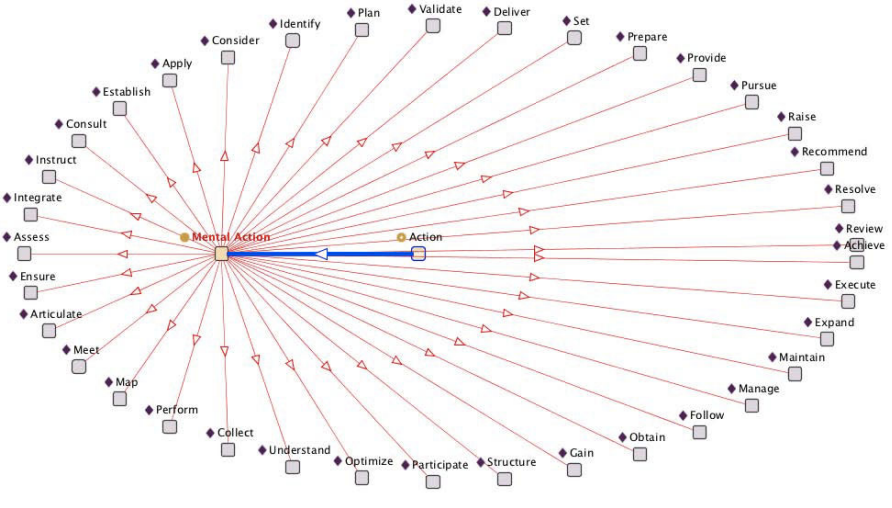

Fig. 2. Mental actions performed by TI human resources

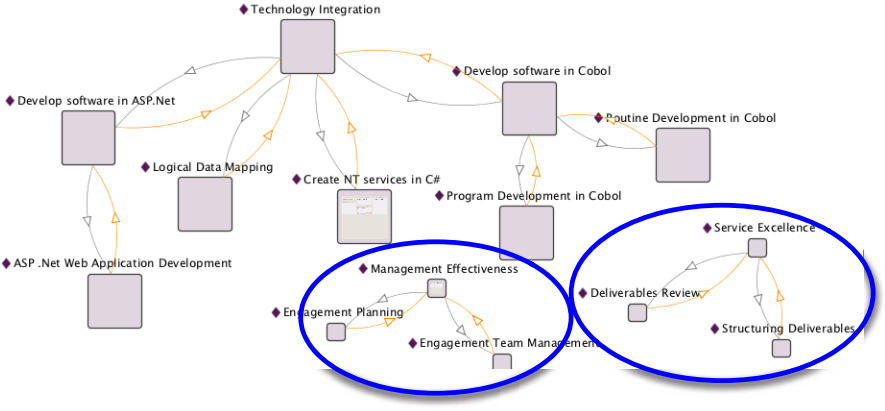

Fig. 3. Competencies of the Business and IT Strategy cluster

observed subjects, in which problem-solving capabilities are regarded as the most important, and where they aim at understanding client's technological needs, identifying potential issues and/or IT opportunities, and ensuring quality deliverables, while managing and developing IT solutions. Though action collection started with this basic set, it was extended in the task capture phase.

Task capture. Tasks were collected with task statements in the form (Action Resource→Goal). The use of goals proved to ease task statement writing, as they tend to make it more objective. A total of 32 task statements were captured from 12 participants.

Model representation entails representing the competency model and its elements in a language suitable for its visualization, discussion and later actualization. The model was formalized and built with Proteg (section 5), which was used to create, visualize, and manipulate the model in various representation formats for discussing the collected information with the selected subjects. Once the model was created, a number of queries were made to evaluate the model:

Q1 *What type of actions is the Customer capable of?*
Q2 *What are the Customer competencies?*
Q3 *What competencies does "Business & IT Strategy" competency cluster include?*
Q4 *Which activities/actors/roles/goals are associated to a given competency?*
Q5 *Which resources are associated to the "Business & IT Strategy" cluster?*

The model answered successfully all questions. Due to space limitations, we only show exemplary answers to Q1, Q3, and Q5 (figures 2-4). Figure 2 shows the mental actions that the Customer's TI human resources are capable of performing. Figure 3 shows the competencies themes comprised within the Busines and IT Strategy cluster; Technology Integration, Service Excellence and Management Effectiveness, as well as the individual competencies comprised within each theme. Figure 4 offers a resource-oriented view by showing the six resources comprised in this competency cluster; IT Cost, IT Governance, IT Operation, Project & Portfolio Management, Scorecard Models, and Sourcing Agreements.

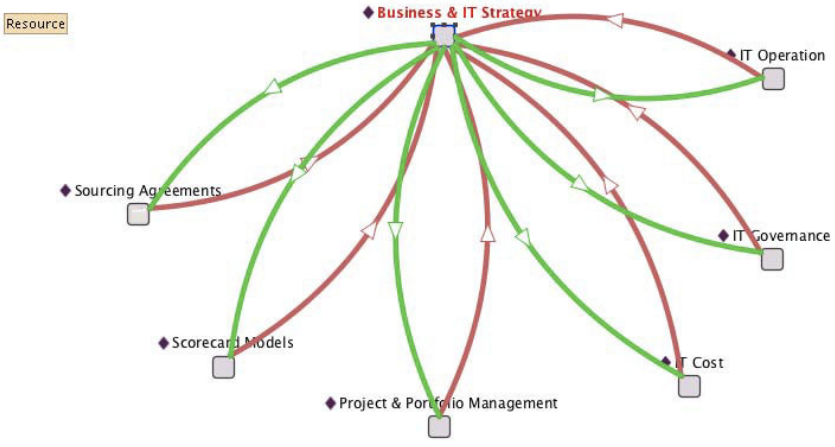

Fig. 4. Resources used within the Business and IT Strategy cluster

5 Supporting Tools

The model was built using various tools including: an ontology editor, an English lexical database useful for natural language processing, and a micro-blogging application.Tasks were collected with task statements, using two complementary tools: **Yammer**® and **Microsoft Excel**® forms. **Yammer**® (yammer.com) is a micro-blogging tool for enterprises. In **Yammer**®, co-workers exchange short, frequent answers to a simple question, such as "What are you working on?" As employees answer that question, a company feed is created in one central location, enabling co-workers to discuss ideas, post news, ask questions and share links and other information. The company feed can be accessed in real time via the desktop, the Web, IM, SMS text messaging, mobile devices or email. **Yammer**® also serves as a company directory in which every employee has a profile; it also features a knowledge base in which past conversations are archived and searchable. Anyone in a company can start their **Yammer**® network and begin inviting colleagues; privacy of each network is ensured by limiting access to those with a valid company email address. A **Yammer**® network has been created to capture competencies-related information in a more informal manner. Practitioners were invited to freely send brief text updates describing what they were doing at the time. Unlike **Yammer**®, the **Microsoft Excel**® form was sent by email to all practitioners. Here, practitioners described up to five tasks using a closed-vocabulary (the action and resource set), and related them to existing competency themes and clusters.

In order to assist the manual review and maintenance of new elements, **WordNet**® (wordnet.princeton.edu) has been used to identify synonyms among distributionally similar words. **WordNet**® [27] is a lexical database for the English language. It groups English words into sets of synonyms called *synsets*, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of

dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications. An application has been developed using a C# interface for **WordNet**®. For each element in initial action and resource set, the application returns its synonyms from **WordNet**® database, which are then manually compared to the remaining elements in a recursive manner, until no two elements with the same meaning exist.

The ontology editor **Prot ègè**® was the tool selected to create, visualize, and manipulate the model in various representation formats for discussing the collected information with the subjects involved in the case study. **Prot ègè**® (protege.stanford.edu) is an ontology editor and a knowledge base editor put forward by the Standford University. At its core, **Prot ègè**® implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats such as frames, RDF [28] or OWL [29]. The representation format used was **Prot ègè**®-frames. All graphical views were generated using **Jambaya**® plug-in. This plug-in is developed by the CHISEL at the University of Victoria [30]. Below we show the formal definition of the task entity with **Prot ègè**®-Frames representing the intrinsic relations of the *Competency* entity defined in section 3.1.

```
(defclass Competency
    (is-a Entity)
    (role concrete)
    (multislot hasInOrderTo
        (type INSTANCE)
;+      (allowed-classes In+Order+To)
        (create-accessor read-write))
    (multislot hasCompetency
        (type INSTANCE)
;+      (allowed-classes Competency)
        (create-accessor read-write))
    (multislot inverse_of_hasCompetency
        (type INSTANCE)
;+      (allowed-classes Competency)
        (create-accessor read-write))
    (multislot hasTask
        (type INSTANCE)
;+      (allowed-classes Task)
        (create-accessor read-write)))
```

6 Conclusions and Outlook

This paper proposes a multi-dimensional approach to represent competencies in terms of the following semantic units; actions, resources and goals where each unit represents a different dimension as depicted in figure 5. These units are building blocks that allow defining elementary competencies. Following a bottom-up approach, these competencies can be composed into macro competencies such as themes, clusters, so they can be handled as a unit, and thus bound to actors and activities, specifying supply and demand of competencies.

Arguably, the proposed model is too complex to be practical. However, competencies are indeed complex concepts that to properly be connected to other enterprise views and concepts entails regarding them from different perspectives. Moreover, the case study provided empirical evidence of the model feasibility and

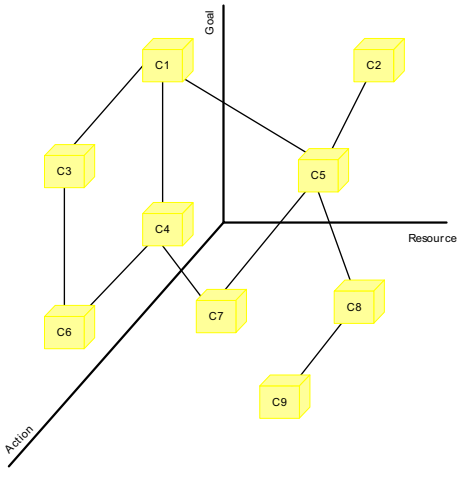

Fig. 5. Competency dimensions

usefulness. The advantage shown by this case study is three-fold. First, defining competencies by selecting from a set of action types and resources was easier than defining whole competency definitions. Second, separate action and resource sets are proving to be more stable than competency definitions that do not make this separation. Third, this decoupling allows using the action classification proposed in [20] and matching actor cognitive characteristics to organizational activities. Nonetheless, since the number of elementary competencies can be rather high, it is necessary to use automated tools to support competency capture and representation. We have explored a number of tools that eased model building and discussion but further research is required for larger number of competencies. Future work also entails addressing issues regarding both tool development and modeling such as:

- *Activity-Actor alignment*: devise alignment metrics between the competency suppliers (actors) and requesters (activity)
- *Semantic Web compliance*: prepare the model for usage with semantic Web (including searching and reasoning), and semantic-enabled wikis
- *Automated maintenance and review of new semantic blocks*: explore WordNet to test for synonyms in questionnaire answers and action logs;
- *Automated semantic blocks discovery*: explore automated discovery of semantic units from job descriptions, using rule-based grammars and text mining techniques.

References

1. Schekkerman, J.: How to Survive in the Jungle of Enterprise Architecture Frameworks. Trafford (2004)
2. Porter, M.E.: Competitive Advantage: Creating And Sustaining Superior Performance: With A New Introduction. Free Press, New York (1985)

3. Schuler, R.S.: Strategic Human Resources Management: Linking the People with the Strategic Needs of the Business. Organizational Dynamics Summer 21, 18–32 (1992)
4. Schuler, R.S., Jackson, S.E.: Understanding Human Resource Management in the Context of Organizations and Their Environments. Annual Review of Psychology 46, 237–264 (1995)
5. Caetano, A., Pombinho, J., Tribolet, J.: Representing Organizational Competencies. In: SAC 2007: Proceedings of the 2007 ACM symposium on Applied computing, pp. 1257–1262. ACM, New York (2007)
6. Federal Information Processing Standards Publication 183: Integration Definition For Function Modeling (IDEF0) (December 1993)
7. White, S.A.: Business Process Modeling Notation (BPMN). Business Process Management Initiative (BPMI), Version 1.0 (May 2004)
8. Mayer, R.J., Menzel, C.P., Painter, M.K.: deWitte, P.S., Blinn, T., Perakath, B.: IDEF3 Process Description Capture Method Report. Knowledge Based Systems, Inc. (September 1995)
9. Ould, M., Huckvale, T.: Process Modelling: Why, What And How. In: Software Assistance For Business Re-Engineering, pp. 81–97. John Wiley and Sons Ltd., Chichester (1994)
10. Dietz, J.L.G.: Enterprise Ontology. Springer, Heidelberg (2006)
11. The Open Group: The Open Group Architectural Framework (TOGAF), Version 8.1 Enterprise Edition (2003), `http://www.opengroup.org/architecture/togaf8-doc/arch/`
12. CAP Gemini: Integrated architecture framework IAF (2007), `http://www.capgemini.com/resources/`
13. Zachman, J.A.: A Framework For Information Systems Architecture. IBM Systems Journal 26(3), 454–470 (1987)
14. Lankhorst, M.: Enterprise Architecture at Work, Modelling, Communication and Analysis. Springer, Heidelberg (2005)
15. Sousa, P., Caetano, A., Vasconcelos, A., Pereira, C., Tribolet, J.: Enterprise Architecture Modeling with the Unified Modeling Language. In: Enterprise Modeling and Computing with UML. IGI Global, pp. 69–97 (2006)
16. Tarassov, V., Sandkuhl, K., Henoch, B.: Using Ontologies for Representation of Individual and Enterprise Competence Models. In: 2006 International Conference on Research, Innovation and Vision for the Future, pp. 206–213 (2006)
17. Gronau, N., Uslar, M.: Creating Skill Catalogues for Competency Management Systems with KMDL. In: Khosrow-Pour, M. (ed.) Innovations Through Information Technology. IDEA Group Press (2004)
18. Harzallah, M., Leclére, M., Triche, F.: CommOnCV: Modelling The Competencies Underlying A Curriculum Vitae. In: SEKE 2002: Proceedings of the 14th international conference on Software engineering and knowledge engineering, pp. 65–71. ACM, New York (2002)
19. Harzallah, M., Berio, G., Vernadat, F.: IT-Based Competency Modeling And Management: From Theory To Practice In Enterprise Engineering And Operations. Computers In Industry 48(2), 157–179 (2002)
20. Overbeek, S., van Bommel, P., Proper, H.: Embedding knowledge exchange and cognitive matchmaking in a dichotomy of markets. Expert Systems with Applications (36), 12236–12255 (2009)

21. Lang, A., Pigneur, Y.: Digital Trade of Human Competencies. In: HICSS 1999: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences, Washington, DC, USA, vol. 5, p. 5008. IEEE Computer Society, Los Alamitos (1999)
22. Varela, A.R., Soares, F.M.: Modelação e análise da interacção entre recursos humanos e processos de negócio. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa (2003)
23. Aveiro, D., Mendes, J., Tribolet, J.: Organizational Modeling With A Semantic Wiki. In: SAC 2008: Proceedings of the 2008 ACM symposium on Applied computing, pp. 592–593. ACM, New York (2008)
24. Nijssen, G.M., Halpin, T.: Conceptual Schema and Relational Database Design, 2nd edn. Prentice Hall, Englewood Cliffs (1996)
25. Fine, S.A., Cronshaw, S.F.: Functional Job Analysis: A Foundation for Human Resources Management. Lawrence Erlbaum Associates, Mahwah (1999)
26. Baskerville, R.L.: Investigating Information Systems With Action Research (October 1999)
27. Fellbaum: WordNet: An Electronic Lexical Database (Language, Speech, and Communication). The MIT Press, Cambridge (May 1998)
28. W3C: Resource Description Framework (RDF) (2002)
29. W3C: Web Ontology Language (OWL) Guide (2004)
30. Storey, M.A., Noy, N.F., Musen, M., Best, C., Fergerson, R., Ernst, N.: Jambalaya: an Interactive Environment for Exploring Ontologies. In: IUI 2002: Proceedings of the 7th international conference on Intelligent user interfaces, p. 239. ACM, New York (2002)

Context-Aware Collaborative Platform in Rural Living Labs

Olfa Mabrouki[1,2], Abdelghani Chibani[2], Yacine Amirat[2], Monica Valenzuela Fernandez[3], and Mariano Navarro de la Cruz[3]

[1] CityPassenger SA, Avenue de l'Atlantique
Les Conquérents BP 903, 91976 Courtaboeuf Cedex, France
omabrouki@citypassenger.com
[2] Université Paris XII Val-de-Marne, LiSSi, E.A. 3956
120-122 rue Paul Armangot, 94400 Vitry sur seine, France
{olfa.mabrouki,chibani,amirat}@univ-paris12.fr
[3] Grupo Tragsa, Gerencia TIC ||ICT Division
Subdireccin de I+D+i ||Subdirectorate of Innovation and R&D
Julin Camarillo, 6 B; 4 planta Madrid Spain
{mvaf,mnc}@tragsa.es

Abstract. In this paper, we present a collaborative context-aware framework for rural living labs or rural innovation ecosystems as Social Spaces for Research and Innovation (SSRI). The proposed framework exploits seamless integration of standard ubiquitous computing technologies to support smart collaboration and knowledge sharing between rural communities. We underline an open collaborative platform based on context-aware components useful in any rural living lab area. This platform is focused on the bus component which acts as a connector for the different living labs. The bus architecture gives the advantage to facilitate the dynamic integration of living lab services using discovery and binding methods. It guarantees also the large scale interconnection of all the living labs. Building such framework requires resolving a main issue of the design approach. Moreover, we experiment the use of such platform in rural community of fishery sector. The work presented in this paper is one of the main results of the Collaboration@Rural (C@R) European research project: a collaboration platform for working and living in rural areas (FP6-2005-IST-5-03492) that aims to build a platform of networked living labs for context-aware collaborative working in rural areas.

Keywords: rural living labs, collaborative platform, context awareness, ubiquitous computing, orchestration.

1 Introduction

The term Living Lab (LL) was given at the first time in 2003 by ProfWilliam Mitchell from MIT, Media Lab and School of Architecture and city planning. He defines this new concept as a research methodology for sensing, prototyping, validating and refining complex solutions in multiple and evolving real life contexts.

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 65–76, 2010.

This new concept is also represented as innovation environments where stakeholders form a partnership of enterprises, users, public agencies and research organizations. Since then, many definitions of LL have been proposed in the literature. In a LL, cooperation is established for creating, prototyping and using new products and services in real-life environments. Users are not seen as an object of innovation and customers but as early stage contributors and innovators [1]. Thus, we might view LLs as concrete implementations of user driven open innovation environments [2]. Consequently, we have agreed on the following definition: LL is a research methodology for innovation that challenges the whole research and innovation process in real-life conditions by human, social, cultural, organizational and institutional aspects, and has an impact on sustainable service,business and technology development.

Several researches in the field of LL were undertaken in the literature. The most of them concerns the proposition of ubiquitous technologies applications to impact socially and economically LLs environments. Other works deal with LLs in education [3] and home environment [4]. In the last five years, the European commission funded researches to build LLs as open innovation platforms. These LLs are intended to be used in a real-world environment for collaboration among stakeholders in the value of ICT (Information and Communication Technologies) production [5].

Hence, the paradigm of LL is used more and more in the area of ubiquitous computing as it involves user in the early phases of service innovation. Moreover, Rural Living Labs (RLLs) are a special kind of this paradigm that enhances ICT rural development and that are user-centered design. Ubiquitous collaboration in LLs targets to provide users with integrated context-aware services capable of exploiting all the facilities of both wired and wireless environments. These services allow creating flexible and effective virtual teams. Such integrated services placed in the rural context may be adapted to different users situations and environment characteristics.

The C@R LLs have been setting up to experiment advanced collaborative work and business innovations to enhance attractiveness of rural areas and strengthen rural development. Within the C@R project, a LLs vision and methodology have been developed and implemented to create rural and regional innovation environments. This project aims to propose and develop an innovative collaborative platform for rural communities and demonstrate the use of this platform by integrating various tools for various user communities. It promotes the development of an open collaborative architecture which enables the reuse and the contextualization of services and collaborative tools. Seven LLs have been launched; six in Europe and one in South-Africa, to establish and experiment collaborative platforms and applications enhancing Small and medium enterprises (SME) related work and business collaboration in specific sectors.

In this paper, we highlight our proposal for designing the collaborative context-aware platform for LLs and managing context-aware services. The methodology used is called Open Service Oriented Architecture (OSOA). The context-aware collaborative platform enhances social and economic conditions in rural environments

through the introduction and experimenting of new ways of working and managing business. This platform is focused on the bus component which acts as a connector for the different LLs. The bus architecture gives the advantage to facilitate the dynamic integration of LL services using discovery and binding methods. The paper is organized in the following way: section II introduces the framework for context-aware collaborative platform in LLs. In the section III, we illustrate the use of this framework for rural LLs in the fishery sector.

2 Framework for Context-Aware Collaborative Platform in Living Labs

An efficient methodology to set up a framework for context-aware collaborative platform is indispensable. We present and argue the use of the OSOA approach for the design of such architecture. Then, we describe the framework with its several layers and highlight the different features of context-aware components in this framework.

2.1 Framework Modeling Methodology: OSOA Approach

Service Oriented Architecture (SOA) nowadays is a well known term providing principles of how to develop and integrate a system of loosely coupled services. SOA defines not only the low-level software architecture design principles but is a complete enterprise software concept including among others security, governance, deployment and integration. The term Open Service Oriented Architecture (OSOA) which is used to define the C@R SOA [6][7] approach also defines a set of principles to develop and integrate a system of loosely coupled services but also components. The differences to a traditional SOA approach are the concepts built upon and how they are combined to provide the ground of a service oriented collaborative working environment (CWE).

On the software architectural level there currently exist many different flavors and interpretations of service-oriented architecture (SOA) concepts, which are being promoted by different organizations. One of the most popular and active SOA developers group is the Open SOA (OSOA) Collaboration [8], which represents an informal group of industry leaders with a common goal. They work together on the definition of a language-neutral programming model able to meet the needs of enterprise developers who are developing software following the SOA principles.

The OSOA follows similar concepts and represents a system that exploits the SOA benefits using language-neutral concepts to build the ground for a service oriented CWE. Since the C@R architecture will be used in a broad community, among different LLs, and in a diverse set of use case scenarios it is essential to build the architecture upon well defined standards and to avoid proprietary concepts. The most important standards used in the C@R OSOA improve the quality of developed software and thus ensure the interoperability, maintainability and reusability of the individual components. This enables the utilization of advanced Collaborative Working Environments even in rural setups.

The C@R CWE System is certainly a different and wider concept than the OSOA Collaboration Group proposal and other SOA approaches, as it is not just considering software developers agreeing on a software architecture providing user services, but as a whole Open System to enable a CWE considering all available actors: users, equipment, service providers,software providers, CWE system designers and stakeholders.

Reference Architecture Design. The main purpose of the C@R project is to introduce and involve collaborative works environments which will raise rural development. Actually, the issue is related to the networked LLs where users need to collaborate, communicate and exchange information. Setting up an open collaborative architecture that enables the reuse, the contextualization and the personalized of services and tools is mandatory to cover the rural character of such LLs.

Architecture layers. A layered architecture design realizes decoupled components [9]depicted in the figure 1 to deal with different aggregation levels of business functionality, namely:

- **CCS. C**ollaborative **C**ore **S**ervices implemented as reusable software components that encapsulate distinctive core functionality. Such functionality provides basic services (e.g. 3 G connectivity, SMS delivery, order creation etc.). CCSs plug into the C@R Control BUS where they are registered. Every CCS provides a public API, implemented as a Web-service;
- **SCT. S**oftware **C**ollaboration **T**ools [9] comprise aggregated functionality, which can be integrated into a final RLL application, but is of such a degree of independence to be usable for various applications even across different Living Labs. Simple SCTs provide only one CCS and more sophisticated SCTs orchestrate several CCSs into a business process to the RLL application via a web service interface. SCTs can be defined using different languages. One of the basic objectives of C@R is to use as much as possible standard languages. Current implementations of the SCTs use BPEL [10] and/or BPMN [11] that allows the creation of scripts, which are executed by the orchestration engine;
- **OC. O**rchestration **C**apabilities [12] provide collaborative functions and libraries that will be used by executable scripts that define the composition of SCTs. Three orchestration capabilities are identified, namely Context Awareness, Distributed Workspace, and Advanced Services. A Collaborative situation may involve atomic functions from different OCs such as Messages Broadcasting, Shared Display, Videoconference systems, etc. categorized as the three identified orchestration capabilities. OCs can be implemented as Web Services (following the CCS design) or as static libraries deployed together with the Control BUS;
- **RLL. R**ural **L**iving **L**ab applications cover end user interactions (via a User Interface) with a system supporting collaborative workflows that overcome problems related to rural activities. These applications make use of underlying

layered business functionality encapsulated in SCTs but also linking directly to CCS and OC functionality.

Additional to these layers a Control BUS has been conceptualized and implemented in order to centrally deal with component registration and brokerage.

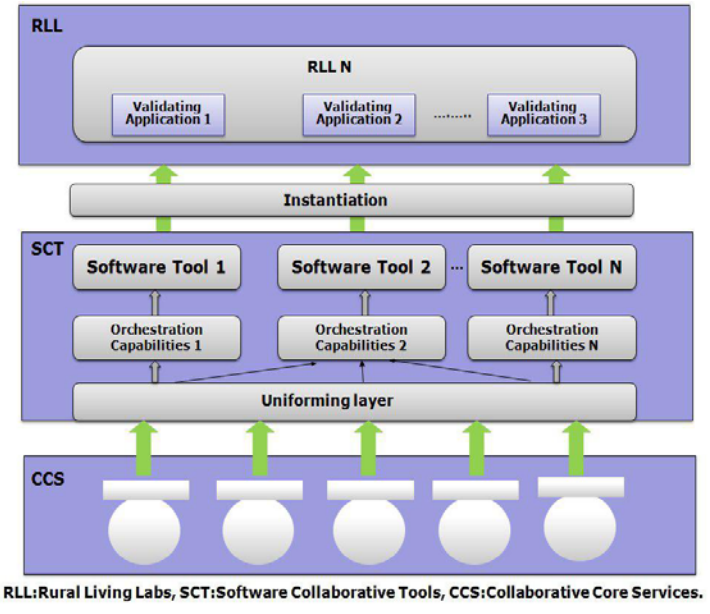

Fig. 1. Framework for context-aware networked RLLs

Control Bus. Control functions of the elements of the C@R Architecture are encapsulated in the Control BUS. It acts as a resource broker, where signaling information about resources is exchanged, enabling the system to search for resources, managing their interconnection and supporting collaboration among different CWEs. The BUS acts as an informing middleware that is a conceptual inter-layer space designed for CCS component harmonization, homogenization and adaptation to standards. It makes the C@R architecture more powerful and flexible allowing an easy integration of proprietary or new standard CCS components. This key piece of C@R architecture consists of five modules (see figure 2):

- **Bus maintenance.** This module is responsible for keeping the logs of all the BUS activities, and for all tasks related to the management of the BUS itself. Furthermore, the module offers configuration files and interfaces for the administrators to control the behavior of the BUS;
- **Registrar.** This module is responsible for keeping a database of all components (CCS and Orchestration Capability) connected to the system. Furthermore it implements search functionality, allowing any element to look for other elements (CCS, Orchestration Capability) connected to the platform;

- **Connector.** This module allows interconnection of components and the management of release and monitoring of connections among them;
- **Instantiation management.** This module serves to support the instantiation process;
- **Bus Inter-working.** This module is responsible for negotiating the communication with other C@R platform BUSes, or even control layers of other platforms. It provides collaboration among different BUSes and enables resources sharing.

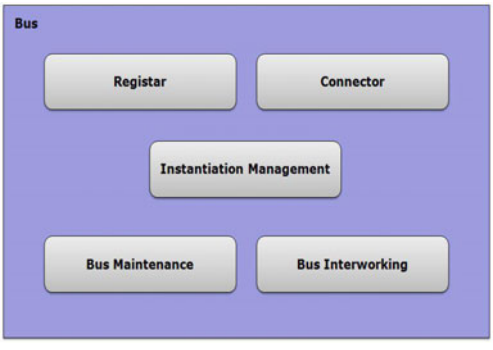

Fig. 2. Control bus

BUS implementations require the establishment of information channels with the resources that it pretends to manage and interconnect. Those Control Communications are centralized by the BUS and use Web Services as transport technology, while Data Communications are P2P and may use any kind of transport technology.

Orchestration capabilities. The Orchestration Capabilities (OC), as defined by C@R [12], provides collaborative functions and libraries that will be used by executable scripts that define the composition of SCTs. Three orchestration capabilities have been identified:

- Distributed Workspaces includes the minimum necessary methods and data structures to build high-level collaborative functions;
- Context Awareness adds functions providing environmental status and methods for context change reaction;
- Access to Advanced Services includes SIP/IMS & Security capabilities.

C@R has analyzed current collaboration activities performed at Rural Areas (using the seven Rural Living Labs included in C@R. This analysis concluded on the identification of 45 services that where common to two or more RLL and 60 services that are specific for only one RLL. The identified common services are the candidates for being including in any of the OCs of C@R in order to provide the basic support for creating complex collaboration services.

Context-aware components. Context awareness was introduced for the first time by Schilit and Theimer [13]. They defined context of an entity as a set of information concerning the identity of the entity, its location, identities of nearby objects and changes to those objects. Ryan et al. [14] present context of an entity as its environmental information, such as location, time, temperature and its identity. Dey [15] considers context of an entity as its physical, social, emotional, and mental (focus-of-attention) environments, location and orientation, date and time of day, other objects in the environment.

The majority of these researchers share a common vision of context as it represents a set on information about location, time and activity of a person. In the C@R architecture, CCS components are contextaware components as they provide contextual information about the user. Actually, these components provide information about user location, user profile, spoken languages and Web sensors, namely the following components have been implemented and used in the individual Living Labs (see figure 3):

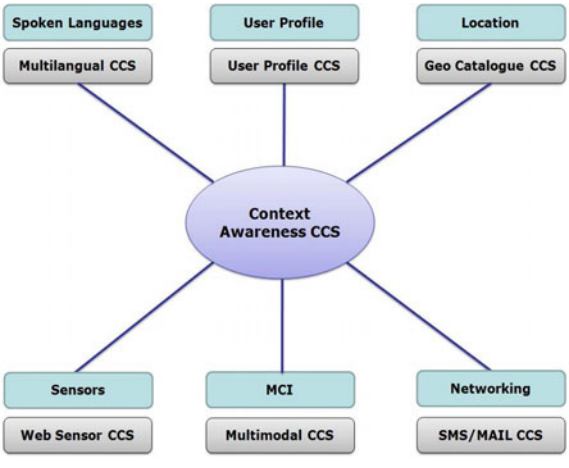

Fig. 3. Context-aware components

– **User profile:** the User Profile Component (UPC) describes the user personal information, preferences and role. Once the user is authenticated, it is possible to load his particular profile and particular preferences. In this way, available components and their settings are customized on startup. The context awareness API related to UPC allows full access with appropriate authorization to context information component to allow technicians the reuse of the UPC by simply invoking some methods. This component is for its nature distributed and cross-LLs as potentially. It is expected to run on the top of different databases which are implemented into several and heterogeneous products both open source and commercial licenses;

- **Geo-Catalogue:** catalogue services are the key technology for locating, managing and maintaining distributed resources. With catalogue services, client applications are capable of searching for resources in a standardized way (i.e. through standardized interfaces and operations). Catalogue component developed for C@R architecture automatically collects CCS component metadata and provides their registration into metadata catalogue. This internal component gives to users/developers possibility to register metadata records for new developed CCS as well as to know which CCS are currently available in the C@R architecture. Once registered metadata can be searched, extended or updated using some of metadata applications (Metadata editor or Metadata Catalogue System);
- **Web sensor:** this component presents many opportunities for adding a real-time sensor dimension to the Internet and the Web. This has extraordinary significance for science, environmental monitoring,transportation management, public safety, facility security, disaster management, utilities Supervisory Control and Data acquisition (SCADA) operations, industrial controls, facilities management and many other domains of activity;
- **MDLC:** the **M**ultilanguage **D**ata **L**oading **C**omponent is a context-aware component which allows user applications to retrieve a set of configuration files that contain the localized texts needed to interact with the end users. This software component enables user applications to adapt their interfaces to a particular user. Hence,the communication happens in terms of their preferred language. The component aims to empower user applications with the capability to perform the loading of different languages as a set of localized texts at the interface level stored in configuration files which are particular to a specific application.

3 Experimenting C@R Platform in the Fishery Sector

Among the different rural regions of LLs, we have chosen as an experiment example the Cudillero living lab (in Asturias, Spain). This LL has been developed in collaboration with public administrations, the local authorities and the fishery guilds. Applications developed for the fishermen try to enhance current business processes in order to make fishing production more profitable, e.g. helping on the day-by-day activity of the users in the vessels and in the auction process via the transmission of reports on the catches (arrival hour, sizes of the catches, total weight of the catches, etc.) and thereby contributing with a significant time and workload saving. The applications also contribute to improve the safety of fishermen in case of accidents or health emergencies, providing an immediate response from the health authorities. Furthermore, the collaboration between vessel and port will serve to optimize the organization of the port activities.

The following use cases have been implemented in the Cudillero RLL: GPS based catches data sending; Weather reports; Alerts management service and safety on board; Messages delivery service by instant messaging and presence. Based on mockups and prototypes validated by the user community, the software

platform is implemented according to the principles of the proposed reference architecture. Once prototypes are developed, the basic software components and their interactions are determined. As a result the three layered architecture is mapped onto the individual components (see Figure 4, for use case Catches data sending). From bottom to top, CCS (Collaborative Core Services) components are the atomic resources which are orchestrated thanks to a control middleware, by service scripts and collaborative functions in the SCT layer of the architecture. These collaborative core services (CCSs) in layer 1 are registered in the resources broker (Bus) enabling the system to search for resources and managing their interconnection. A homogeneous layer (BusCCSOperations library) registers and connects CCSs to the Bus, in order to make each identified CCS available to the C@R platform.

Use case	Components	Description
GPS based catches data sending	MQS	Messages Queuing Service. This component is used to guarantee that messages are sent when lack of communication coverage.
	GPS_LS	GPS Service to get the boat GPS position
	SRS	Speech Recognition Service to allow fishermen to fill in the reports through their voice
	MDLS	Multilanguage Data Loading Service. MDLS is a software component that discriminates the application language and texts.
	AAAS	Authentication Authorization and Audit Service to get software components connected and registered to C@R architecture and let users to register in the platform
	DMS	Data Management Service to manage data from CDSApp
	DSS	Data Storage Service responsible for the database
	SMS_S	SMS Service
	EM_S	Email Service
	CDS App	Catches Data Sending Application, including SCTs and user interface
	FIS App	Fishery Information System Application- including SCTs and user interface
	UPC	User Profile Component

Fig. 4. Prototypes in basic components, in this case catches data sending

Services for Cudillero operate in a main domain where two sub domains are distinguished: the fishing boats sub domain and the fishery sub domain (see figure5). Each sub domain relies on one C@R bus. Different sub domains are registered in the Cudillero domain through the bus internetworking capability. This module also enables the reutilization of basic resources or the information exchange with other domains as other ports (i.e. Aviles port). Layer 2 in Cudillero utilizes a specific component as an Orchestration Capability: the authentication Authorization and Audit Service (AAAS). AAAS acts as a transversal service that needs to be preregistered in the bus to let the rest of CCSs to be authenticated to the C@R platform.

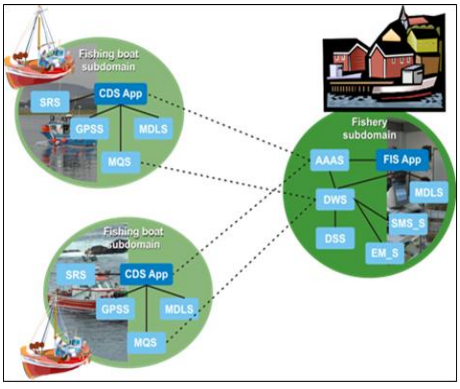

Fig. 5. Software components in Cudillero sub domains

Fig. 6. CDS APP main data screen

Layer 3 defines the Collaborative services instantiation process. SCTs or software collaboration tools are the key elements to instantiate the collaborative platform relying in each bus. The SCTs deal with the modeling of the business processes of each sub domain. This piece of software is first compiled to get a BPEL script. These scripts contain information about all the necessary elements and basic services to be connected and started to run the platform. This BPEL script is uploaded to the SCT scripts repository. When the instantiation process begins, the SCT is downloaded to a server (composition engine) configured with some instantiation parameters (additional code). As a result, CCSs defined in

the SCT are deployed to the server, registered to the bus and started. Most of the software components are identified as Collaborative Core Services (CCSs) except the AAAS that acts as a transversal service that needs to be preregistered in the bus to let the rest of CCSs to be authenticated to the C@R platform.

Two special CCSs were distinguished in each sub domain: 1) CDS App - Catches Data Sending Application, see Figure 5, and 6) FIS App - Fishery Information System Application. These specific CCSs (CCS applications) consist of the graphic user interfaces and the service framework. These are main threads that use and orchestrate the basic services (rest of CCSs) to compose end user applications.

4 Conclusion

One of the key objectives of C@R is the development of a reference architecture reflecting advantageous concepts that overcome a variety of challenges and pain points typical for rural CWEs. Deriving common characteristics of such architecture turned out to be difficult due to the limited capabilities of end users to reflect on technical needs and to the differences in target sectors of the 7 Living Labs involved.

C@R found out overlaps between architectural needs if not between all Living Labs at least between some of them. These overlaps have been translated into several principles (decoupling, open standard compliancy, flexible infrastructure support, service orchestration, interoperability, etc.) that drove the architectural design and the implementations in the individual Living Labs.

The common principles of the reference architecture have been realized exemplary and subsequently validated in terms of added value. Such common principles include the usage of most important standards (e.g. web services, BEPL), component representation (e.g. BPMN), tools (e.g. Intalio Designer), reusable, encapsulated functionality (OC services, CCSs), security models (e.g. AAS) or service brokerage (e.g. BUS). Besides commonalities the flavored implementations in the different Living Labs also showed distinctive differences that reflect the local specifics, e.g. the usage of the sub domain concept in Cudillero (fishing boats) or the limited usage of the BUS in Sekhukhune due to network impediments. The full potential of architectural benefits couldnt be leveraged during the lifetime of the project. Nevertheless the validation of architecture implementations in distinctive experiments provided promising results: In particular the C@R reference architecture is capable to facilitate the reuse of collaboration services, concepts and components across design and runtime environments of different CWEs.

The required degree of flexibility to develop and operate software collaboration tools has been assured through the usage of the most relevant standards in the fields associated to services. Openness and interoperation of CCS components for SCT orchestration coming from different platforms have been showcased. The performance of SCTs using the base components of the architecture is satisfactory and cost and effort required to develop software collaboration tools are competitive.

References

1. Hippel, E.V.: Democratizing Innovation. MIT Press Books, vol. 1. The MIT Press, Cambridge (2006)
2. Guzman, J.G., Schaffers, H., Bilicki, V., Merz, C., Valenzuela, M.: Living labs fostering open innovation and rural development: Methodology and results, Asia-Pacific Tech. Monitor (September 2007)
3. Abowd, G.D.: Classroom 2000: An experiment with the instrumentation of a living educational environment. IBM Systems Journal 38, 508–530 (2000)
4. Kidd, C.D., Orr, R., Abowd, G.D., Atkeson, C.G., Essa, I.A., Macintyre, B., Mynatt, E., Starner, T.E., Newstetter, W.: The aware home: A living laboratory for ubiquitous computing research, pp. 191–198 (1999)
5. Oliveira, A., Fradinho, E., Caires, R.: From a successful regional information society strategy to an advanced living lab in mobile technologies and services. In: Hawaii International Conference on System Sciences, vol. 4, p. 83a (2006)
6. D2.1.3, C.P.D.: C@r and osoa design (July 2008)
7. D2.5.1, C.P.D.: C@r and osoa design (July 2008)
8. Edwards, M.: Open service oriented architecture (June 2006), `http://www.osoa.org`
9. D2.1.5, C.P.D.: Workflows for rural activities (October 2009)
10. OASIS: Bpel web services business process execution language version 2.0., `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpelv2.0-OS.html`
11. Group, O.M.: Bpmn, `http://www.bpmn.org`
12. D2.5.2, C.P.D.: Integration of distributed workspaces, localization and context awareness and access to advanced service components into sct for rural environments (October 2009)
13. Schilit, B., Theimer, M.: Disseminating active map information to mobile hosts. IEEE Network 8, 22–32 (1994)
14. Ryan, N.S., Pascoe, J., Morse, D.R.: Enhanced reality fieldwork: the context-aware archaeological assistant. In: Gaffney, V., van Leusen, M., Exxon, S. (eds.) Computer Applications in Archaeology 1997, British Archaeological Reports, Oxford, Tempus Reparatum (October 1998)
15. Dey, A.K.: Context-aware computing: The cyberdesk project. In: AAAI 1998 Spring Symposium on Intelligent Environments, Palo Alto, pp. 51–54. AAAI Press, Menlo Park (1998)

A Formal Approach to Architectural Descriptions – Refining the ISO Standard 42010

Sabine Buckl[1], Sascha Krell[2], and Christian M. Schweda[1]

[1] Lehrstuhl für Informatik 19
Technische Universität München
Boltzmannstraße 3, 85748 Garching, Germany
{sabine.buckl,schweda}@in.tum.de
[2] Professur für Wirtschaftsinformatik
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
sascha.krell@unibw.de

Abstract. Architectural descriptions representing and modeling the architecture of a system or parts thereof are typically used in the engineering disciplines to plan, develop, maintain, and manage complex systems. Primarily originating from construction engineering, the means of architecting and architectural descriptions have been successfully transferred to related disciplines like software engineering. While a rich and formal theory on conceptual modeling exists as well as frameworks on how to approach architectural descriptions, e.g. the ISO standard 42010, only few attempts have yet been made to integrate the prescriptions and guidelines from these sources into a formal architectural description framework. In this paper, we establish such a framework against the background provided by the ISO standard 42010 by formally defining the terms *concern*, *view*, *viewpoint*, and *architectural description*. Further, an outlook discusses potential application areas of the framework.

1 Motivation

Development, design, and maintenance of architectures have a long history in the engineering disciplines. Primary originating from construction engineering the objectives of architecture – to be strong or durable (firmitas), useful (utilitas), and beautiful (venustas) [1] – and their means have been applied to other disciplines to address challenges in related domains (cf. [2,3]). One of these disciplines is software engineering, in whose context the term architecture can be defined in accordance with the ISO Std. 42010 as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" [4]. In the same sense, Rechtin coined the following proverb "every system has an architecture" in [5]. He thereby, emphasized on the fact that an architecture is an *intrinsic* property of a system that cannot be neglected. The architecture, however, is not necessarily documented, i.e. made explicit. This idea yields a

A. Albani and J.L.G. Dietz (Eds.): CIAO! 2010, LNBIP 49, pp. 77–91, 2010.

delicate but central distinction of the two terms *architecture* and *architecture description* that will reverberate through the remainder of the article. During the software development process, for instance, a variety of different documentations, so called "views" or "models", of the system under consideration are created and used to facilitate the communication between the involved stakeholders, e.g. customers, software architects, and software engineers. Examples for such "views" are different diagrams corresponding to the different diagram types as proposed by the UML, the de-facto standard for software engineering [6]. In accordance with the ISO Std. 42010 the entirety of these views is referred to as *architectural description*, a "collection of products documenting the architecture" [4] of the system.

In the context of architectural descriptions, the way human beings, i.e. stakeholders, comprehend knowledge about what is perceived to exist, i.e. the system under consideration, plays an important role. The science dealing with such aspects is referred to as *epistemology* [7]. In [8] Becker and Niehaves present an epistemological reference framework for information systems research, which details on questions in which way a person can arrive at *true* cognition. In the context of architectural descriptions, especially the *ontological aspect* discussed in the framework is of interest. The ontological aspect is concerned with the object of cognition, i.e. the architecture of the system under consideration. Thereby three positions can be differentiated

ontological realism, according to which the system's architecture exists independently of human cognition,

ontological idealism, according to which the system's architecture is a construct depending on the consciousness of the observing person, and

kantianism, according to which the system's architecture consists of parts, which are dependent and independent of the observing person.

Abstaining from in-depth discussions on the ontological perspective, we resort ourselves to the position of ontological realism that well aligns to the understanding of Rechting [5].

Although the importance of architectural descriptions, e.g. in the context of software engineering is unquestioned, no general formal framework for such descriptions has yet been developed. For special purpose languages as e.g. the UML, formalization approaches have nevertheless been undertaken, see e.g. [9]. The absence of a general framework is especially interesting as rich and formal theories for conceptual modeling exists as well as frameworks on how to approach architectural descriptions (cf. [4]). Building on the framework of the ISO Std. 42010, Dijkman et al. present in [10] a basic formalization approach targeting consistency of viewpoints in the context of multi-viewpoint modeling. Doubtlessly their approach provides a valuable contribution in the field, but does not deliver a comprehensive formalization for the field of architecture description. This in contrast is the experienced research gap, that our paper addresses by introducing and discussing frameworks for conceptual modeling and for architectural descriptions in Section 2. These frameworks are used as foundation

for the approach to architectural descriptions presented in Section 3, which formally defines the terms *concern*, *view*, *viewpoint*, and *architectural description*. Finally, Section 4 provides a critical reflection of the contribution of the paper and discusses future topics of research.

2 Describing Architectures

Postponing more elaborate considerations on the structure of an architectural description to Section 2.2, we restrict ourselves to an intuitive understanding of description as a purposeful abstraction of an architecture, i.e. as some sort of *conceptual model* thereof. This aligns with the definition of the latter term given by Mylopoulos, who claims in [11] that the activity of conceptual modeling, which is the activity of creating conceptual models, is "the activity of describing the physical and social world around us for purposes of communication and understanding". He further states, that conceptual models are developed for certain users or *stakeholders*, as we call them in accordance with the ISO Std. 42010 (cf. [4]). In the remainder of the paper, we will understand architectural descriptions as conceptual models of the architecture, more precisely as specifications of conceptual models of the architecture. We precedingly discuss further properties of conceptual models and on ways to describe these properties in a more abstract fashion in the following section.

2.1 Introduction to Conceptual Models

Guizzardi introduces in [12] a framework that can be applied to promote the understanding of the relationships between architectures, conceptual models thereof, and architectural descriptions. Figure 1 summarizes the framework, thereby also introducing additionally relevant concepts as *conceptualization* and *modeling language*.

Central concepts of the framework are on the one hand the *(mental) model* of the architecture and the corresponding *model specification*. The mental model thereby alludes to a stakeholder's understanding of what the architecture looks alike. The model specification is therein understood as the *representation* of a model using an appropriate modeling language. This introduces the concept of the representation that in accordance to Guizzardi [12] can be understood as function mapping from the space of model entities to a corresponding space of syntactically correct instantiations of modeling language constructs. Complementing the representation function, the framework provides the *interpretation function* that conversely maps from syntactically correct instantiations of the modeling language constructs to corresponding model entities. With these definitions at hand, it intuitively becomes clear that the purposefulness and appropriateness of an architectural description, i.e. a model specification, heavily depends on the backing modeling language's ability to express and represent the concepts' underlying the mental model. Staying in the field of conceptual modeling, Guizzardi derives in [12] a framework for evaluating the appropriateness of a

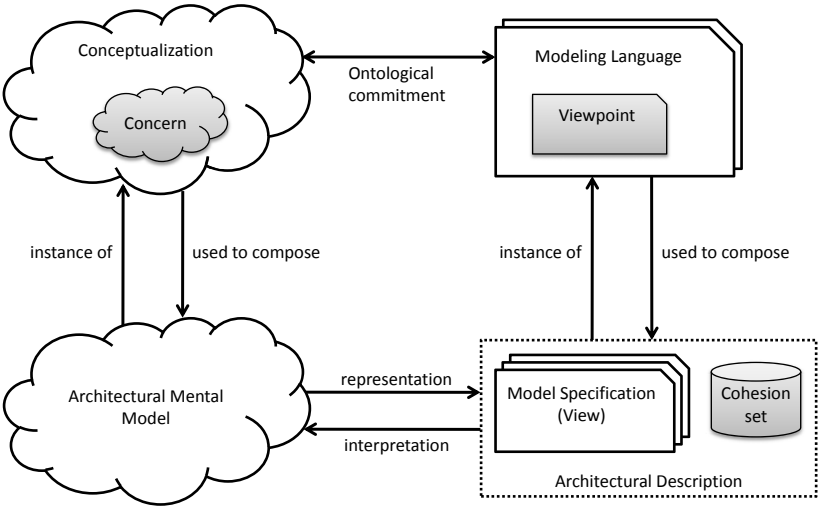

Fig. 1. Conceptual modeling on architectures

modeling language. This framework builds on a basic idea coined by Gurr in [13] according to which the representation function needs to be an *isomorphism* with the interpretation function as its inverse. Elaborating on the difficulties associated with proofing the isomorphism nature of a function mapping mental models to representations thereof, Guizzardi reformulates the demand to two requirements for the representation function and two complementing requirements targeting the interpretation function. These requirements read as follows:

Lucidity. A specification must be *lucid* with respect to its corresponding model, which means that every construct in the specification must represent at most one entity from the model. Thereby, overloaded representation constructs are forbidden. *(injective representation function)*

Soundness. A specification must be *sound* with respect to its corresponding model, which means that every construct in the specification must represent at least one entity from the model. Thereby, construct excess in the representation is avoided. *(surjective representation function)*

Laconicity. A specification must be *laconic* with respect to the corresponding model, which means that every entity in the model can be derived by interpretation from at most one construct in the specification. Thereby, construct redundancy is forbidden. *(injective interpretation function)*

Completeness. A specification must be *complete* with respect to the corresponding model, which means that every entity in the model must be derivable by interpretation from at least one construct in the specification. Thereby, it is ensured that the entire model can be derived from its specification. *(surjective interpretation function)*

In the light of above requirements, different definitions for the term *modeling language* are discussed in literature, where e.g. Guizzardi in [12] stays to a

structural perspective on languages inspired by the idea of the two mappings. More advantageous for our subsequent considerations is a notion of language that provides more insights into the internals of modeling. In this vein, we adopt the definition advocated by Kühn in [14], where he names the following language constituents:

Syntax.[1] The syntax defines the structure of the language, i.e., the constructs that form the language body, and the grammar, i.e., the rules for combining the constructs to valid language expressions.

Semantics. The semantics provides the meaning for the language constructs, i.e., prescribes the interpretation of the constructs to corresponding model entities.

Notation.[2] The notation introduces user-perceivable concepts, as e.g. symbols, used to present language constructs to a language user.

In terms of Kühn [14], the representation function of Guizzardi [12] consists of semantic and notational aspects. Explicitly accounting for the latter concepts is beneficial for our subsequent consideration due to multiple reasons, of which the most prominent one is "notational plurality". This aspect is discussed by Kühn on a general level, but also emphasized in the context of different architecture-related disciplines, as e.g. EA management, for which Buckl et al. present in [15] a technique for decoupling notational aspects from semantic ones and show how such a technique can be implemented in a tool. With this in mind, one can justify to abstain from in-depth considerations on the notational aspect of languages.

2.2 ISO Std. 42010

The ISO Std. 42010 on architectural descriptions for software and systems engineering [4] establishes a terminological framework for discussions on architectural descriptions. As part of this framework, the relevant terms of architecture and architectural description are defined. A more in-depth discussion on the structure of an architectural description is provided along a *metamodel*[3] for such descriptions, of which Figure 2 presents the relevant part for this article. The concepts introduced thereby are defined in [4] as follows:

Stakeholder. A stakeholder is an individual, team, organization, or role having concerns with respect to the system.

Concern. A concern is an area of interest in a system pertaining to developmental, technological, business, operational, organizational, political, regulatory, social, or other influences important to one or more of its stakeholders.

View. A view is a representation of a system from the perspective of an identified set of architecture-related concerns. Thereby, the system is abstracted to *architectural models*.

[1] In other language specifications, the concept is alluded to as *abstract syntax*.

[2] In other language specifications, the concept is alluded to as *concrete syntax*.

[3] The standard actually uses the term "conceptual model" and denotes "metamodel" only as alternative term. To avoid confusions, the remainder of the article employs the latter term.

Viewpoint. A viewpoint contains conventions for the construction and interpretation of a view.

The notion of architectural model is further detailed in the standard, leading to the conclusion that such model should be identified with a model specification in terms of Guizzardi [12]. In addition, explanation for the distinction between view and model is provided, emphasizing on the possibility to reuse architectural models in different views. While such modularity might in fact be beneficial during applications of the framework, it does to us seem only of minor importance in the context of the subsequent formal approach to architectural descriptions. In this vein, we apply without loss of generality a simplification to the metamodel by uniquely identifying views and architectural models and subsuming these concepts under the term "view".

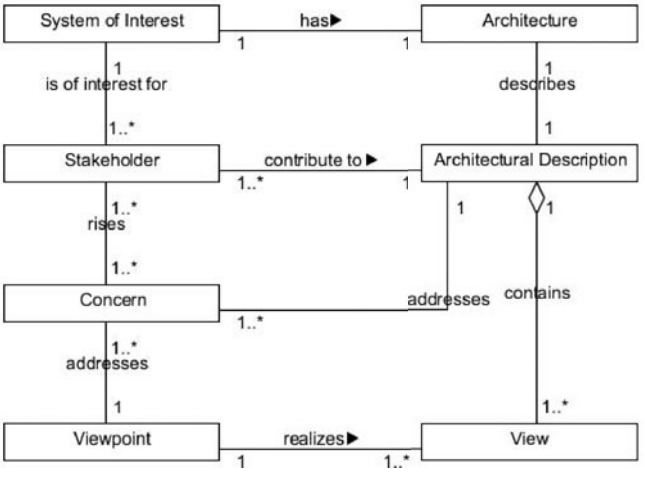

Fig. 2. Metamodel of the ISO Std. 42010 [4]

With this shortened version of the standard at hand, we briefly revisit the concepts introduced in Section 2.1. The stakeholder concept allows to make explicit the owner of a mental model that is represented as part of an architectural description. The conceptualization underlying the mental model as reflected in the corresponding modeling language is mirrored in the stakeholder's concerns. Put in other words, a concern employs an underlying conceptualization that again is tightly linked to the syntax and semantics of the modeling language, while notational aspects may not apply here. These notational aspects finally come into play during the construction of the actual views, which are developed and interpreted along the guidelines stated in the viewpoint. From this, one can sensibly derive that the modeling language constitutes a part of the viewpoint. The above considerations lay the basis for our subsequent elaborations on a formal approach to architectural descriptions.

3 A Formal Approach to Architectural Descriptions

The ISO Std. 42010 provides a valuable framework for understanding architectural descriptions embedded into their creation and utilization context, as well as their related stakeholders. This is achieved by a metamodel of concepts used for making the description's context explicit. Following the interpretation of architectural descriptions or more precisely their constituting views as *models* of the architecture in terms of conceptual modeling, a disadvantage of the informal conceptualization of the standard becomes apparent. In particular, the standard does not allow for reasoning on an architectural description's underlying representation and interpretation function, which recruits from corresponding functions in the constituting viewpoints.

Subsequently, a formalization of the concepts from the metamodel of the ISO Std. 42010 is provided, which can be used for profound formal considerations on the model nature of an architectural description. Preparing this formalization, we introduce two basic concepts as follows:

- the *set of all possible architectures* $\mathcal{A}$, which corresponds to the "set of possible worlds" as introduced by Guizzardi in [12], and
- the *set of all entities and relationships* $\mathcal{D}$. The entities and relationships make up the constituents of an architecture, i.e. exist in the corresponding "world".

With these basic notions in mind, our roadmap for this section at first reconciles the terms of *conceptualization* and *modeling language*, respectively. Building on a formalized understanding of these terms, we derive relationships on the corresponding structures, leading to the notion of an *ontological commitment* of a language in respect to a conceptualization (cf. [12]). Formal definitions for *concerns* and *viewpoints* are presented as extensions of *conceptualizations* and *modeling languages*, respectively. Alongside, formalized relationships are introduced to define the terms of *embedding*, *equivalence*, and *orthogonality* of concerns on the one hand, as well as the *appropriateness* of a viewpoint in respect to a concern on the other hand. Concluding our formalization approach, we elaborate on *consistency* requirements on architectural descriptions that employ multiple viewpoints. For the subsequent discussions, it also has to be noted that albeit the formality of the presented model, we do not assume that each stakeholder describes his or her conceptualizations, concerns, or viewpoints on such an abstract level.

3.1 Conceptualization and Modeling Language

Guizzardi introduces his definition of conceptualization in [12] as a set of structuring principles used (implicitly) during the creation of a mental model. In this vein, a conceptualization provides *concepts* that are used to classify real-world entities and relationships[4] to *classes* or *association types*. In line with this understanding, a conceptualization $z \in \mathcal{Z}$ is described as a tuple consisting of:

[4] Entities and relationships are subsequently subsumed as "instances".

- the set of admissible architectures $\mathcal{A}_z \subseteq \mathcal{A}$,
- the set of admissible instances $\mathcal{D}_z \subseteq \mathcal{D}$, and
- the set of concepts[5] $\mathcal{R}_z \subseteq \bigcup_{n\in\mathbb{N}}(\mathcal{A}_z \rightarrow \mathbb{P}(\mathcal{D}_z^n))$.

The above definition yields a *contextual* and *extensional* understanding of concepts, i.e., defines the concept not solely over its instances, but in relation to the architecture that they are contained in, and understands a concept in a set-theoretic fashion. This also explains, why the powerset notation is used: a concept $\langle a, \mathcal{D}_1\rangle$ is a tuple of an architecture and the corresponding instances that together form the *extension* of the concept.

With this understanding of conceptualizations, we can further establish an *equivalence* relationship $\equiv \subseteq \mathcal{Z} \times \mathcal{Z}$. This relationship builds on the existence of a bijective function $\underline{d}_z : \mathcal{D}_{z_1} \rightarrow \mathcal{D}_{z_2}$. Based on this, the equivalence of two conceptualizations $z_1 = \langle \mathcal{A}_{z_1}, \mathcal{D}_{z_1}, \mathcal{R}_{z_1}\rangle$ and $z_2 = \langle \mathcal{A}_{z_2}, \mathcal{D}_{z_2}, \mathcal{R}_{z_2}\rangle$ can be defined as:

$$z_1 \equiv z_2 \Leftrightarrow \mathcal{A}_{z_1} = \mathcal{A}_{z_2} \wedge \\ \forall \langle a, \{d_1, ..., d_n\}\rangle \in \mathcal{R}_{z_1} : \langle a, \{\underline{d}_z(d_1), ..., \underline{d}_z(d_n)\}\rangle \in \mathcal{R}_{z_2}.$$

Pursuing the path from a conceptualization towards a modeling language, a language $l \in \mathcal{L}$ is defined as tuple $l = \langle \mathcal{S}_l, i_l, n_l\rangle$ with three constituents: *(abstract) syntax* $\mathcal{S}_l$, a mapping function $i_l : \mathcal{S}_l \rightarrow \mathcal{D}_l \cup \mathcal{R}_l$ reflecting the *semantics*, and a *notation* mapping n_l. The latter mapping defines for each syntactical concept the corresponding symbolic representation. Resorting to the considerations from Section 2.1, we abstain from discussions on the notation function here.

Linking conceptualizations and modeling languages, we resort to the notion of *ontological commitment* as discussed by Guizzardi in [12]. In these terms, a modeling language commits itself to a conceptualization, iff it is suitable to represent the conceptualization's concepts. In the operationalization of the commitment relationship $\sim \subseteq \mathcal{L} \times \mathcal{Z}$, the suitability is formalized as the existence of a bijective function $m : \mathcal{D}_l \cup \mathcal{R}_l \rightarrow \mathcal{D}_z \cup \mathcal{R}_z$ such that

$$l \sim z \Leftrightarrow \exists\, L \subseteq image(i_l) : m(L) = \mathcal{D}_z \cup \mathcal{R}_z.$$

Refraining to the equivalence relationship between conceptualizations as defined above, an alternative definition of the ontological commitment relationship can be established between a modeling language and an equivalence class of conceptualizations $[z]_\equiv$. Based on this notion, a modeling language l commits itself to a set of (equivalent) conceptualizations $[z]_\equiv$, iff a bijective function $m : \mathcal{D}_l \cup \mathcal{R}_l \rightarrow \mathcal{D}_z \cup \mathcal{R}_z$ exists such that

$$l \sim [z]_\equiv \Leftrightarrow \exists\, L \subseteq image(i_l) : m(L) = \mathcal{D}_z \cup \mathcal{R}_z,$$

where z on the right hand side denotes an arbitrary element from the equivalence class $[z]_\equiv := \{z' \in \mathcal{Z} | z' \equiv z\}$.

[5] The symbol $\mathbb{P}$ represents the powerset of a given set.

3.2 Concern and Viewpoint

An architectural concern represents a stakeholder's "area of interest" in an architecture. In this vein, the concern is reflected by a distinct mental model of the architecture or more precisely a part thereof, complemented with an underlying conceptualization. The concern further denotes which parts of the overall architecture are mirrored in the corresponding "model", i.e. selects the relevant *instances*. Put in other words, a concern goes beyond a sole conceptualization but also brings along a *filter function* f that determines which parts of the architecture are of interest. Formally, a concern $c \in \mathcal{C}$ can be defined as tuple $c = \langle \mathcal{A}_c, \mathcal{D}_c, \mathcal{R}_c, f_c \rangle$ with:

- the set of admissible architectures $\mathcal{A}_c \subseteq \mathcal{A}$,
- the set of admissible instances $\mathcal{D}_c \subseteq \mathcal{D}$,
- the set of concepts $\mathcal{R}_c \subseteq \bigcup_{n \in \mathbb{N}} (\mathcal{A}_c \rightarrow \mathbb{P}(\mathcal{D}_c^n))$ that is further consistent with
- the filter function $f_c : \mathcal{A}_c \rightarrow \mathbb{P}(\mathcal{D}_c)$.

The term "consistent" in this case means

$$\forall \langle a, \{d_1, ..., d_n\} \rangle \in \mathcal{R}_c : \{d_1, ...d_n\} \subseteq f_c(a).$$

At this point it is necessary to give some general remarks. We do not try to formalize the term mental model or tantamount architecture[6] itself. In our opinion this is a pointless task, because of the impossibility to discover the complete nature of a complex system. Any kind of identification, with a colored graph for example, would be a simplification of that system, which might be allowed for modeling purposes but not for a formalization approach, aiming at a complete and consistent specification. Neither we can give construction rules for the above introduced filter functions with the same argument. A certain filter function strongly relies on a specific theory about the modeled field, which cannot be anticipated in advance. The intension behind a concern has thus to be left to philosophical observations.

Figure 2 makes obvious that a viewpoint can relate to more than one concern. From our perspective such an aggregation of concerns can only be made, if the aggregated concerns can themselves be interpreted as *one* consistent concern again. Descriptive this means, all participating concerns must employ a compatible conceptualization, reflected in an assignment of equal concepts for the same instances, or must in contrast be completely disjoint, i.e. filter for other parts of the architecture under consideration. Otherwise, we would gain complex types, which could cause problems in interpreting them, i.e. there might be no correspondent entity in the real world domain for a given complex type. This would break the *soundness* requirement of Guizzardi (cf. Section 2.1) hence depriving an architectural description using this "inconsistent" viewpoint of its validity as representation of a mental model of the architecture. In these cases, it would be more beneficial to define a further viewpoint to avoid this situation.

[6] Epistemological, an architecture could be as well understood as the intension of a system of interest.

Approaching the notion of a "consistent aggregation" of concerns, we can establish a relationship among concerns expressing that one concern can be *embedded* into another one. Put in other words, embedding ($c_1 \sqsubseteq c_2$) means that the area of interest represented by one concern c_1 is completely covered by the area of interest of another concern c_2. The *embedding*-relationship ($\sqsubseteq \subseteq \mathcal{C} \times \mathcal{C}$) can formally be defined via an auxilliary set $\mathcal{D}'_{c_1} \subseteq \mathcal{D}_{c_2}$ of instances covered by c_2 and a bijective function $\underline{d}_c : \mathcal{D}'_{c_1} \rightarrow \mathcal{D}_{c_2}$ as follows[7]

$$\begin{aligned} c_1 \sqsubseteq c_2 \Leftrightarrow\ & \mathcal{A}_{c_1} = \mathcal{A}_{c_2} \wedge \\ & \forall\, \langle a, \{d_1, ..., d_n\}\rangle \in \mathcal{R}_{c_1} : \langle a, \{\underline{d}_c(d_1), ..., \underline{d}_c(d_n)\}\rangle \in \mathcal{R}_{c_2} \wedge \\ & \forall\, a \in \mathcal{A}_{c_2} : \{\underline{d}_c(d) \mid d \in f_{c_1}(a)\} \subseteq f_{c_2}(a). \end{aligned}$$

From the above *embedding*-relationship between concerns, we can consistently derive an *equivalence*-relationship as a mutual embedding. In formal terms, the equivalence of concerns $\equiv \subseteq \mathcal{C} \times \mathcal{C}$ can be defined as

$$c_1, c_2 \in \mathcal{C} : c_1 \equiv c_2 \Leftrightarrow c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_1.$$

The equivalence of concerns thereby builds on the equivalence of the underlying conceptualizations $z(c_1)$ and $z(c_2)$, respectively. In more detail two concerns c_1 and c_2 are equivalent (regarding a bijective function $\underline{d}_c$) as follows:

$$c_1 \equiv c_2 \Leftrightarrow z(c_1) \equiv z(c_2) \wedge \forall\, a \in \mathcal{A}_{c_2} : \{\underline{d}_c(d) \mid d \in f_{c_1}(a)\} = f_{c_2}(a).$$

Following the logic of this observation it becomes apparent, that the "biggest" possible concern comprehends the interests of all stakeholders and consequently reflects the whole architecture. It is out of doubt that such a concern would be overly sized to be handled effectively. Nevertheless, the embedded-relation is capable of building the foundation for *abstract* concerns, which comprise the interests of different stakeholders.

We suggest two additional terms in the matter of further improvement for the understanding of compatibility of concerns. Both provide sufficient but not necessary conditions for compatibility.

At first, we are aiming at concerns, which do not "affect" each other, and can therefore be put together without depriving the description's soundness. To substantiate this depiction, we establish the notion of *orthogonality*: Two concerns $c_1, c_2 \in \mathcal{C}$ are orthogonal $c_1 \perp c_2$, if they cover different areas of interest, more precisely, if there is no bijective function $\underline{d}$ that maps the concerns' instances ($\mathcal{D}_{c_1}$ and $\mathcal{D}_{c_2}$) consistently for all commonly admissible architectures. Preparing a formalization of orthogonality, we introduce the auxiliary function $A : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{P}(\mathcal{A})$ of architectures that are admissible in respect to two concerns, as follows

$$A(c_1, c_2) := dom(f_{c_1}) \cap dom(f_{c_2}).$$

[7] In the definition we apply the bijection $\underline{d}_c$ to a set of instances. Thereby, we want to denote the set that results from element-wise application of the bijection.

With this shorthanded notation, the orthogonality of concerns can be defined as[8]

$$c_1 \perp c_2 \Leftrightarrow \nexists\, \underline{d} : \mathcal{D}_{c_1} \rightarrow \mathcal{D}_{c_2} : \forall\, a \in A(c_1, c_2) : f_{c_1}(a) \cap \underline{d}(f_{c_2}(a)) = \emptyset,$$

where $\underline{d}$ is a bijective function.

The second sufficient prerequisite for compatibility of concerns is *conceptualization compatibility*. To allow for a concise definition of the latter relation, we introduce another auxiliary function $R_{\underline{d}} : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{A} \times \bigcup_n \mathbb{P}(\mathcal{D}^n)$ as follows[9]

$$R_{\underline{d}}(c_1, c_2) := \{\langle a, \delta \rangle | a \in A(c_1, c_2) \wedge \delta \subseteq \bigcup_n (f_{c_1}(a) \cap \underline{d}(f_{c_2}(a)))^n\}.$$

For two given concerns c_1, c_2, the value of $R_{\underline{d}}(c_1, c_2)$ denotes the possible concepts as tuples $\langle a, \delta \rangle$ over common instances and corresponding architectures that are admissible for both concerns. The value thereby depends on the selection of the bijective function $\underline{d} : \mathcal{D}_{c_1} \rightarrow \mathcal{D}_{c_2}$ in the sense of the former definition. For the context of a specific architecture $a \in A(c_1, c_2)$ Figure 3 illustrates the meaning of $R_{\underline{d}}(c_1, c_2)$.

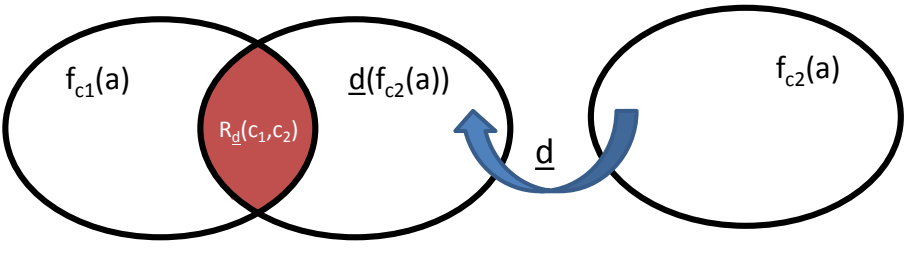

Fig. 3. Illustrating $R_{\underline{d}}(c_1, c_2)$ in the context of an architecture a

This allows for a concise definition of a *conceptualization compatibility*-relationship via a bijective function $\underline{d} : \mathcal{D}_{c_1} \rightarrow \mathcal{D}_{c_2}$ as

$$\begin{aligned} c_1 \sim c_2 \Leftrightarrow\ & \forall\, a \in A(c_1, c_2) \\ & \forall\, \delta_1 \in \{\{d_1, ...d_n\} \mid \langle a, \{d_1, ...d_n\}\rangle \in \mathcal{R}_{c_1} \cap R_{\underline{d}}(c_1, c_2)\} \\ & \forall\, \delta_2 \in \{\{\underline{d}(d_1), ...\underline{d}(d_n)\} \mid \langle a, \{d_1, ...d_n\}\rangle \in \mathcal{R}_{c_2} \cap R_{\underline{d}}(c_1, c_2)\} : \\ & \delta_1 \subseteq \delta_2 \vee \delta_2 \subseteq \delta_1 \vee \delta_1 \cap \delta_2 = \emptyset. \end{aligned}$$

For subsequent considerations about the term *viewpoint*, we assume to have a single concern, that might aggregate the interests of a set of stakeholders in the sense of the prior relations.

Refraining our considerations from Section 2.1, a viewpoint can be understood as an extension of a modeling language in the same manner, as a concern extends an underlying conceptualization. In this vein, we define a viewpoint $v \in \mathcal{V}$ as a tuple $v = \langle \mathcal{S}_v, i_v, n_v, f_v \rangle$ with:

[8] Footnote 7 also applies in this definition.
[9] Footnote 7 also applies in this definition.

- an abstract syntax denoted by a set $\mathcal{S}_v$ of the instances of $\mathcal{D}_v$ that the viewpoint employs,
- a semantics denoted by a mapping function $i_v : \mathcal{S}_v \rightarrow \mathcal{D}_v \cup \mathcal{R}_v$,
- a notation denoted by a function[10] n_v, and
- a filter, determining the relevant part of an architecture, mirrored in a function $f_v : \mathcal{A}_v \rightarrow \mathbb{P}(\mathcal{D}_v)$.

Rounding up the definition from above, we operationalize the relationship between a viewpoint and the (aggregated) concern that it is meant to address. Constructing this *addresses*-relationship, we can rely on the definition of the ontological commitment that relates conceptualizations and modeling languages, of which concerns and viewpoints are extensions. Therefore, it is justifiable to start with the generalized notion of commitment, building on the notion of equivalence classes of conceptualizations. Based on this, we establish an *addresses*-relationship as $\sim \subseteq \mathcal{V} \times [\mathcal{C}]_{\equiv}$ via a bijective function $m : \mathcal{D}_v \cup \mathcal{R}_v \rightarrow \mathcal{D}_c \cup \mathcal{R}_c$, as:

$$v \sim [c]_{\equiv} \Leftrightarrow \mathcal{A}_c = dom(f_v) \wedge \forall\, a \in \mathcal{A}_c : \{m(d) \mid d \in f_v(a)\} = f_c(a).$$

3.3 View and Architectural Description

Inline with the prefabricates of the preceding section, we define a view m_v corresponding to a viewpoint v as the application of v to an architecture $a \in \mathcal{A}$. We understand a view as a function $m_v : \mathcal{A}_v \rightarrow \mathbb{P}(\mathcal{S}_v)$, that satisfies the following expression

$$\forall\, a \in \mathcal{A}_v : \{i_v(s) \mid s \in m_v(a)\} \cap \mathcal{D}_v = f_v(a).$$

The function m_v evaluates to the elements of the viewpoint's syntax that are needed to represent the relevant architectural instances in a *sound* and *lucid* way (cf. Section 2.1). This formalization of view hence complies with the term *specification model* introduced by Guizzardi in [12].

A considerable advantage of a multi-viewpoint description of an architecture can only be achieved, if all views can be tied together in a "sensible" way. Our proposal establishes this correlation by connecting the sets $\mathcal{D}_v$ and $\mathcal{S}_v$ of all implemented viewpoints v into one *cohesion set* at architectural description level. This set allocates all instances to their underlying concepts defined in $\mathcal{R}_v$ in the different viewpoints as well as the abstract syntax as context in which these elements are embedded.

With $\mathcal{V}_{ad}$ we denominate the set of all viewpoints realized within an architectural description ad. The cohesion set $\mathcal{CS}_{ad}$ of an architectural description is defined as:

$$\mathcal{CS}_{ad} := \langle \mathcal{E}^{sem}_{\mathcal{V}_{AD}}, \mathcal{E}^{syn}_{\mathcal{V}_{AD}} \rangle$$

[10] Resorting to the argumentation at the modeling language, we abstain from detailing the notation function.

with

$$\mathcal{E}^{sem}_{\mathcal{V}_{ad}} := \{\langle s, \{i_v(s)\}\rangle \mid \exists\, v \in \mathcal{V}_{ad} : s \in \mathcal{S}_v \wedge i_v(s) \in \mathcal{R}_v\}$$
$$\mathcal{E}^{syn}_{\mathcal{V}_{ad}} := \{\langle s, \{i_v(s)\}\rangle \mid \exists\, v \in \mathcal{V}_{ad} : s \in \mathcal{S}_v \wedge i_v(s) \in \mathcal{D}_v\}.$$

Finally, we have all ingredients at hand to present the essential characterization for an architectural description $ad(a)$ of an architecture a as the unity of all its views m_v and the cohesion set $\mathcal{CS}_{ad}$, in short

$$ad(a) := \langle \bigcup_{v \in \mathcal{V}_{ad}} m_v(a),\ \mathcal{CS}_{ad}\rangle.$$

In this manner, we are constructing an architectural description strictly bottom up. For a real world case this might be insufficient, if all instances to be modeled are familiar. In this case, it would be possible to create a taxonomy (or even ontology) of instances to be modeled in advance and purport them as presetting to the architects. The conceptualization would then be known from the outset. Since this condition cannot be achieved under all circumstances, we neglected this proceeding. The implicit assumed matching process over all components in our procedure might seem fairly complex, but it assures to detect all structural relationships and semantical embodiments. Based on the cohesion set constituents, as introduced above, we can establish a mechanism to check for validity of the various semantic representations over the different views as well as the embodiment of instances into the different structural environments.

4 Conclusion and Outlook

In this paper, we provided a formal approach for describing architectures of systems. This approach builds on basic principles of conceptual modeling, as outlined by Guizzardi in [12] and by Kühn in [14]. It further accounts for the multi-perspectivity of architectural descriptions especially of complex systems, as discussed in the ISO Std. 40201 [4]. Bringing these perspectives together in a formal way seems to us valuable to allow for well-founded discussions on the nature of architectural descriptions in many application fields as e.g. software engineering or enterprise architecture (EA) management. In the former field, other formal approaches relevant to multi-perspective modeling exist, such as the "precise UML" approach initially outlined by Breu et al. in [9] and further developed in subsequent publications. This approach necessarily deals with multi-perspectivity as incorporated in the UML, but can resort itself to less general considerations than our paper, as the approach builds on the concrete language and viewpoints as defined by the modeling technique. In the light of an increasing interest in *domain specific languages* (DSLs) for software engineering, our general considerations, e.g. on compatibility and embedding of concerns, may be helpful to provide formal underpinnings usable during the development of DSLs. The field of EA management presents itself as an even more interesting field of application for our approach, as it up to this point lacks a widely-agreed technique for describing EAs. Some researchers even doubt that such a technique

fitting the needs of all companies can be found [16,17]. In this vein, our formal approach may be helpful for the advancement of the field. One may think of the construction of application- or enterprise-specific architectural description techniques and languages, whose internal consistency can be assessed in a more formal way based on the contribution of this paper.

Staying to the application field of EA management, we outline a possibly interesting direction for future research. Buckl et al. proposed in [16] a pattern-based approach to EA management that encompasses a method for constructing appropriate viewpoints for different stakeholders' given concerns in respect to the EA. As part of this approach syntax and semantics corresponding to a concern's underlying conceptualization are made explicit as object-oriented metamodels and textual descriptions, respectively. While this departs from the abstract and intentional nature of the concern, it in contrast allows for more intricate considerations on compatibility and embeddings of concerns based on this paper's considerations. Put in other words, an experienced modeler can check for the compatibility of two object-oriented metamodels in an almost algorithmic manner, while the complementing semantic descriptions ensure that no accidental synonyms invalidate the compatibility check. This formal technique for compatibility checking becomes even more interesting with later versions of the pattern-based approach, which are documented in an online catalog of patterns, of which the V-patterns describe corresponding architectural viewpoints (see [18]). In the course of the refinement of the approach, further V-pattern have been added and new concerns have been described. Complementing this increase in material, also relationships between the patterns have been established (cf. [19]) to reflect that different viewpoints build on each other, i.e., that one viewpoint is "embedded" into another. While up to this point, these whole-part-relationships are derived in an informal and subjective manner, the formal architectural description model presented in this paper may allow for a revision of the relationships. In a long term perspective, an ordering relationship on the concerns and related viewpoints could be derived. With the help of this relationship, a using company could easily understand how their currently selected concerns could be generalized to more comprehensive ones, and could determine, which additional information had to be collected to support this generalization. Thereby, the formal approach towards architectural descriptions would help in the development of a tool support for pattern-based EA management methods.

References

1. Pollio, V.: Vitruvii De Architectura Libri Decem – Zehn Bücher über Architektur – Übersetzt und mit Anmerkungen versehen von Curt Fensterbusch, 5th edn. Primus-Verlag, Darmstadt (1996)
2. Freestone, R.: Urban Planning in a Changing World: The Twentieth Century Experience (Studies in History, Planning, and the Environment). Spon Press, London (2000)
3. Reussner, R., Hasselbring, W.: Handbuch der Software-Architektur. Dpunkt Verlag, Heidelberg (2006)

4. International Organization for Standardization: Iso/iec 42010:2007 systems and software engineering – recommended practice for architectural description of software-intensive systems (2007)
5. Rechtin, E.: Systems Architecting of Organizations – Why Eagles can't swim. CRC Press LLC, New York (1999)
6. Object Management Group (OMG): Uml 2.2 superstructure specification (formal/2009-02-02) (2009), http://www.uml.org (cited 2010-02-25)
7. Burrel, G., Morgan, G.: Sociological Paradigms and Organizational Analysis. Heinemann Educational Publishers, London (1979)
8. Becker, J., Niehaves, B.: Epistemological perspectives on is research: a framework for analysing and systematizing epistemological assumptions. Information Systems Journal 17, 197–214 (2007)
9. Breu, R., Hinkel, U., Hofmann, C., Klein, C., Paech, B., Rumpe, B., Thurner, V.: Towards a formalization of the unified modeling language. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 344–366. Springer, Heidelberg (1997)
10. Dijkman, R.M., Quartel, D.A., van Sinderen, M.J.: Consistency in multi-viewpoint design of enterprise information systems. Information and Software Technology 50(7-8), 737–752 (2008)
11. Mylopoulos, J.: Conceptual modeling and Telos, pp. 49–68. Wiley, New York (1992)
12. Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, CTIT, Centre for Telematics and Information Technology, Enschede, The Netherlands (2005)
13. Gurr, C.A.: On the isomorphism, or lack of it, of representations. In: Visual language theory, New York, NY, USA, pp. 293–305. Springer, Heidelberg (1998)
14. Kühn, H.: Methodenintegration im Business Engineering. PhD thesis, Universität Wien (2004)
15. Buckl, S., Ernst, A.M., Lankes, J., Matthes, F., Schweda, C., Wittenburg, A.: Generating visualizations of enterprise architectures using model transformation (extended version). Enterprise Modelling and Information Systems Architectures – An International Journal 2(2), 3–13 (2007)
16. Buckl, S., Ernst, A.M., Lankes, J., Schneider, K., Schweda, C.M.: A pattern based approach for constructing enterprise architecture management information models. In: Wirtschaftsinformatik 2007, Karlsruhe, Germany, pp. 145–162. Universitätsverlag Karlsruhe (2007)
17. Kurpjuweit, S., Aier, S.: Ein allgemeiner Ansatz zur Ableitung von Abhängigkeitsanalysen auf Unternehmensarchitekturmodellen. In: 9. Internationale Tagung Wirtschaftsinformatik (WI 2007), Wien, Austria, pp. 129–138. Österreichische Computer Gesellschaft (2009)
18. Chair for Informatics 19 (sebis), Technische Universität München: Eam pattern catalog wiki (2009), http://eampc-wiki.systemcartography.info (cited 2010-02-25)
19. Ernst, A.M.: A Pattern-Based Approach to Enterprise Architecture Management. PhD thesis, Technische Universität München, München, Germany (2010) (in submission)

Author Index