

Springer Series in Advanced Manufacturing

Series Editor

Professor D.T. Pham
Manufacturing Engineering Centre
Cardiff University
Queen's Building
Newport Road
Cardiff CF24 3AA
UK

Other titles in this series

Assembly Line Design

B. Rekiek and A. Delchambre

Advances in Design

H.A. ElMaraghy and W.H. ElMaraghy (Eds.)

*Effective Resource Management in Manufacturing Systems:
Optimization Algorithms in Production Planning*

M. Caramia and P. Dell'Olmo

Condition Monitoring and Control for Intelligent Manufacturing

L. Wang and R.X. Gao (Eds.)

Optimal Production Planning for PCB Assembly

W. Ho and P. Ji

Trends in Supply Chain Design and Management: Technologies and Methodologies

H. Jung, F.F. Chen and B. Jeong (Eds.)

Process Planning and Scheduling for Distributed Manufacturing

L. Wang and W. Shen (Eds.)

Collaborative Product Design and Manufacturing Methodologies and Applications

W.D. Li, S.K. Ong, A.Y.C. Nee and C. McMahon (Eds.)

Decision Making in the Manufacturing Environment

R. Venkata Rao

Frontiers in Computing Technologies for Manufacturing Applications

Y. Shimizu, Z. Zhang and R. Batres

Reverse Engineering: An Industrial Perspective

V. Raja and K.J. Fernandes (Eds.)

Automated Nanohandling by Microrobots

S. Fatikow

A Distributed Coordination Approach to Reconfigurable Process Control

N.N. Chokshi and D.C. McFarlane

Machining Dynamics

K. Cheng

Vicente Botti • Adriana Giret

ANEMONA

A Multi-agent Methodology
for Holonic Manufacturing Systems

 Springer

Vicente Botti, PhD
Adriana Giret, PhD
Departamento de Sistemas Informáticos
y Computación (DSIC)
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 Valencia
Spain

ISBN 978-1-84800-309-5

e-ISBN 978-1-84800-310-1

DOI 10.1007/978-1-84800-310-1

Springer Series in Advanced Manufacturing ISSN 1860-5168

British Library Cataloguing in Publication Data

Botti, Vicente

ANEMONA : a multi-agent methodology for holonic
manufacturing systems. - (Springer series in advanced
manufacturing)

1. Computer integrated manufacturing systems - Design
2. Expert systems (Computer science)

I. Title II. Giret, Adriana
670.2'85

ISBN-13: 9781848003095

Library of Congress Control Number: 2008926604

© 2008 Springer-Verlag London Limited

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

The use of registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Cover design: eStudio Calamar S.L., Girona, Spain

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

To Mariló and Víctor

To Miguel and Marquitos

Preface

In the field of intelligent manufacturing, there is a definitive need for methodologies for holonic systems (HMS), based on software engineering principles, which assist the system designer in every development step and provide clear, unambiguous analysis and design guidelines. We believe that methodologies from multi-agent technology (MAS) are good candidates for modeling HMS. Some reasons for this are: the similarities between the holonic and agent approaches, the wide use of agents as the implementation tool for holonic systems, and the availability of complete multi-agent system methodologies. Nevertheless, there are some extensions we have to add to a MAS methodology to be able to model the HMS requirements in a proper way: holon recursive structure, systems abstraction levels, HMS specific guidelines and a mixed top-down and bottom-up approach for analysis and design steps.

In this book we propose an abstract agent notion as a modeling artifact for autonomous entities with recursive structures. The abstract agent extends the traditional definition of an agent adding a structural perspective to the agent concept: ”... *an abstract agent can be an agent; or it can be a MAS made up of abstract agents ...*”. The abstract agent is an attempt to unify the concepts of holons and agents and to simplify and close the gap between holons and agents in analysis and design steps. This will make it easier to translate modeling products, obtained from methodologies for HMS, into coding elements for the implementation of the holonic system.

This book presents ANEMONA, a MAS methodology for HMS analysis and design based on the abstract agent notion and on the HMS requirements. ANEMONA defines a mixed top-down and bottom-up development process, and provides specific HMS guidelines to assist the designer in identifying and implementing holons. In our approach the HMS is specified dividing it into more specific aspects that form different *views* of the system: agent model, organizational model, interaction model, environment model and task/goal model. The way in which the views (models) are defined is inspired by the INGENIAS methodology. The extensions we have made to the INGENIAS metamodels deal with the addition of the abstract agent notion, the redefinition of some relations to conform with the new modeling entities, the dependencies between them and real-time modeling issues from the RT-Message methodology.

The development process of ANEMONA tries to provide the HMS designer with clear, HMS-specific modeling guidelines, and complete development phases for the HMS life cycle. The first stage, *system requirements analysis* and the second stage *holons identification and specification* define the analysis phase of our approach. The aim of the analysis phase is to provide high-level HMS specifications from the problem *requirements*, which are specified by the *client/user* and that can be updated at any development stage. The analysis adopts a top-down recursive approach. One advantage of a recursive analysis is that its results, i.e., the *analysis models*, provide a set of elementary elements and assembling rules. The next step in the development process is the *holon design* stage, which is a bottom-up process to produce the *system architecture* from the *analysis models* of the previous stage. The aim of the *holons implementation* stage is to produce an *executable code* for the *setup and configuration* stage. Finally, maintenances functions are executed at the *operation and maintenance* stage.

Spain,
February 2008

Vicente Botti
Adriana Giret

Acknowledgements

The authors would like to thank Anthony Doyle who initiated the contact with Springer and invited them to write this book.

Furthermore, the authors would like to thank the “Ministerio de Ciencia y Tecnología” (Spain) that partially supported this work by CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

The second author would especially like to express her gratitude to her husband who supported her all the way, particularly when she was going through some difficult moments, and also her family; who were always there.

Finally, the authors would also like to express their thanks to the production team at Springer for their work and the rapid publication of the book.

Spain,
February 2008

Vicente Botti
Adriana Giret

Contents

- Acronyms** xv

- 1 Introduction** 1
 - 1.1 Structure of the Book 2

- Part I Backgrounds**

- 2 Holonic Manufacturing Systems** 7
 - 2.1 Holon 8
 - 2.2 Holonic Manufacturing Systems – HMS 9
 - 2.3 HMS State-of-the-Art 10
 - 2.3.1 Holon Architecture 10
 - 2.3.2 Holons Interconnection 15
 - 2.3.3 Holons Operation 17
 - 2.3.4 Holonic Control 19
 - 2.3.5 Methods for HMS Development 20
 - 2.4 Conclusions 20

- 3 Holons and Agents** 21
 - 3.1 Agents 21
 - 3.2 Holons and Agents: Two Similar Modeling Notions 22
 - 3.2.1 Autonomy 23
 - 3.2.2 Reactivity 23
 - 3.2.3 Proactivity 24
 - 3.2.4 Sociability 25
 - 3.2.5 Cooperation 26
 - 3.2.6 Openness 26
 - 3.2.7 Rationality 27
 - 3.2.8 Mental Attitudes 28
 - 3.2.9 Learning 28
 - 3.2.10 Benevolence 29

3.2.11	Mobility	29
3.2.12	Recursiveness	29
3.2.13	Physical and Information Processing Part	30
3.3	Recursiveness	30
3.4	Abstract Agent	32
3.4.1	Abstract-agent Structure	34
3.5	Conclusion	38

Part II Methodology for Holonic Manufacturing System

4	HMS Development	41
4.1	Modeling Requirements	41
4.1.1	Functional Requirements	41
4.1.2	Software Engineering Requirements	42
4.2	Holonic Manufacturing System Methodologies	44
4.3	Multi-agent System Methods	45
4.3.1	General-purpose MAS Methods	45
4.3.2	MAS Methods for Manufacturing Systems	52
4.4	Enterprise Modeling	53
4.5	Comparative Overview	54
4.6	Conclusions	57
5	ANEMONA Notation	59
5.1	ANEMONA Metamodel	60
5.2	Basic Modeling Entities	62
5.3	Agent Model	66
5.3.1	Abstract Agent and Role	67
5.3.2	Abstract Agent, Role and Goal	67
5.3.3	Abstract Agent and Belief	68
5.3.4	Abstract Agent, Role and Task	68
5.4	Task/Goal Model	70
5.4.1	Abstract Agent, Task and Goals	70
5.4.2	Task, Goals and Beliefs	72
5.4.3	Task Specification	73
5.4.4	Goal Decomposition and Goal Dependencies	75
5.5	Interaction Model	78
5.5.1	Interactions, Abstract Agents, Roles and Goals	79
5.5.2	Interactions, Interaction Units, Abstract Agents, Roles and Tasks	80
5.5.3	Interaction Specification	82
5.5.4	Interactions and Organizations	83
5.6	Environment Model	83
5.7	Organization Model	84
5.7.1	Organization Structure	85
5.7.2	Social Relations Among Autonomous Entities	86

5.7.3	Organization Functional Definition	88
5.8	Conclusions	89
6	ANEMONA Development Process	91
6.1	SPEM	91
6.2	A Simplified Supply Chain Case Study	92
6.3	The Method	93
6.3.1	System Requirement	94
6.3.2	Analysis	97
6.3.3	Design	117
6.3.4	Holon Implementation	130
6.3.5	Setup and Configuration	132
6.3.6	Operation and Maintenance	132
6.4	Conclusions	132
 Part III Evaluation and Case Study		
7	Evaluation of the ANEMONA Methodology	137
7.1	ANEMONA Applicability to Intelligent Manufacturing Problems ..	138
7.2	ANEMONA vs. State-of-the-Art Methods	141
7.3	Conclusions	142
8	Case Study	143
8.1	Requirements	143
8.1.1	Organizational Chart/Departments	144
8.1.2	Business Processes	147
8.1.3	System Scope	149
8.1.4	Processes to Control	149
8.1.5	Operation Conditions	154
8.1.6	Goals	155
8.2	Analysis	156
8.2.1	Iteration 1	156
8.2.2	Iteration 2	168
8.2.3	Iteration 3	179
8.3	Design	185
8.3.1	Holons Specification	185
8.3.2	System Architecture	195
8.4	Conclusions	200
9	Conclusions	201
9.1	Review	201
9.2	Future Work	203
References		205
Index		213

Acronyms

CIM	Computer Integrated Manufacturing
DFD	Data Flow Diagram
FMS	Flexible Manufacturing System
FIPA	Foundation for Intelligent Physical Agents
HMS	Holonic Manufacturing System
IMS	Intelligent Manufacturing System
JADE	Java Agent Development Framework
MAS	Multi-agent System
MOF	Meta Object Facility
PROSA	Product-Resource-Order-Staff Architecture
RUP	Rational Unified Process
SPEM	Software Process Engineering Metamodel
UML	Unified Modeling Language

Chapter 1

Introduction

The manufacturing sector is currently facing a fundamental change from a seller's market to a buyer's market [1]. Competition on a national scale has intensified to a global arena [2], product life cycles have shrunk, and yet there is an escalating requirement to satisfy the specific and individual needs of customers. The success of a manufacturer is no longer measured by their ability to produce a single product cost effectively, but in terms of flexibility, agility and versatility. The changes in markets, customer requirements and technology have become the competition criteria. These rapid environmental changes have forced companies to improve their manufacturing performance in conditions of increasing uncertainty. In order to survive, manufacturing systems need to adapt themselves at an ever-increasing pace to incorporate new technology, new products, new organizational structures, etc.

The trends mentioned above have motivated researchers in academia and industry to create and exploit new production paradigms on the basis of autonomy and co-operation because both concepts are necessary to create flexible behavior and thus to adapt to the changing production conditions. Such technologies provide a natural way to overcome such problems, and to design and implement distributed intelligent manufacturing environments [1]. Distributed intelligent manufacturing systems, and holonic manufacturing systems are considered important approaches for developing industrial distributed systems.

In this study we discuss the engineering of holonic manufacturing systems (HMS). Manufacturing systems are very large and complex. Accordingly, the HMSs that control or implement them are also large and complex. The development process of these kind of systems has to be guided by software engineering methods and principles in order to help the engineer in the development process of the HMS itself. To date, almost all of the applications in the HMS field are built using no design or development method. Moreover, there are probably no two applications developed with the same method. Perhaps this situation has arisen because there has been little work done on methods for HMS development.

There is a definite need for methodologies for HMS [3] that are based on software engineering principles in order to assist the system designer at each stage of development. This methodology should provide clear, unambiguous analysis and

design guidelines. In order to fill this gap in this book we present ANEMONA, a multi-agent method specifically conceived for HMS development, which attempts to fulfill all of the HMS modeling requirements.

ANEMONA is a software engineering method that merges modeling concepts from the development of distributed intelligent systems and the specific modeling requirements of holonic manufacturing systems. ANEMONA guides the system engineer in all of the development phases of the holonic system and provides HMS specific and clear modeling guidelines.

The three fundamental software engineering principles are: decomposition, abstraction and organization. These principles guide the entire development process of ANEMONA. “Divide and conquer” is a widely accepted problem-solving paradigm of computer science. Here, a centralized problem-solving entity accepts a task and separates it into subtasks, which it then distributes among decentralized problem solvers. The problem solvers produce solutions for the subproblems and send these solutions back to the centralized problem solver that integrates the solutions of the subproblems into an overall solution for the original task. Decomposition and abstraction are achieved in ANEMONA by means of the abstract agent notion, and different abstraction levels in the development phases. Abstraction is the process of suppressing, or ignoring, inessential details while focusing on the important, or essential, details. We often speak of “levels of abstraction”. As we move to “higher” levels of abstraction, we shift our attention to the larger, “more important” aspects of an item, e.g., “the very essence of the item”, or “the definitive characteristics of the item”. As we move to “lower” levels of abstraction we begin to pay attention to the smaller, “less important”, details, e.g., how the item is constructed. Finally, the principle of organization is implemented in ANEMONA by means of different analysis and design models, together with HMS-specific modeling guidelines.

1.1 Structure of the Book

The remainder of the book is structured as follows. Part I gives an overview of the state-of-the-art in the field of HMS research (Chap. 2). It also summarizes the conceptual relations among the holon and agent approaches, and presents the abstract agent notion (Chap. 3). The abstract agent notion allows the use of recursive agents in the analysis and design phases of HMS development. The abstract agent notion helps in the definition of a HMS methodology with uniformity of concepts because the same modeling notion can be used from the HMS analysis stage to its implementation.

Part II is the main contribution of this book. It focuses on engineering HMS. In Chap. 4 the HMS modeling requirements are listed and the state-of-the-art methods from the field of HMS research, multi-agent system methodologies and the enterprise modeling techniques are analyzed. Chapters 5 and 6 present the ANEMONA notation and development process respectively. ANEMONA’s central modeling entity is the abstract agent notion. On the other hand, ANEMONA’s development pro-

cess details how the different ANEMONA models are built, as well as the step-by-step activities and tasks that need to be completed to develop the HMS. The ANEMONA process detailed in Chap. 6 is illustrated with the development of a simplified case study from a supply chain management problem.

Finally, Part III focuses on the ANEMONA evaluation (Chap. 7) and a complete real case study from a ceramic tile factory is presented in Chap. 8.

Chapter 9 concludes the book and points towards future work.

Part I

Backgrounds

Chapter 2

Holonic Manufacturing Systems

It is well known that manufacturing systems are complex, large-scale systems for a number of operational and structural reasons. This complexity makes such systems difficult to control and predict. Moreover, in order to meet the challenges of “new manufacturing” these systems will need to satisfy fundamental requirements such as [1, 4]: enterprise integration, distributed organization, heterogeneous environments, interoperability, open and dynamic structure, cooperation, integration of humans with software and hardware, agility, scalability, and fault tolerance.

Many manufacturing paradigms promise to meet these challenges. Two of these paradigms, namely, distributed intelligent manufacturing systems and holonic manufacturing systems, have recently been receiving a lot of attention in academia and industry.

Techniques from artificial intelligence have already been used in intelligent manufacturing for more than twenty years [1]. However, recent developments in multi-agent systems in the domain of distributed artificial intelligence have brought about new and interesting possibilities. Distributed intelligent manufacturing systems, or agent-based manufacturing systems, are based on multi-agent system (MAS) technology [5]. MAS studies the coordination of intelligent behavior among a group of (possibly pre-existing) agents. An agent is an autonomous and flexible computational system, which is able to act in an environment [5]. Today, MAS is a very active area of research and is beginning to see commercial and industrial applications.

Over the last twenty years, researchers have been applying agent technology to areas such as manufacturing enterprise integration and supply chain management, manufacturing planning, scheduling and execution control, materials handling and inventory management, to name a few. For an extensive literature review of these applications see [1]. The mainstream applications of agent-based technology for manufacturing systems uses heterarchical architectures as the control paradigm. A heterarchical control system is a flat structure composed of independent entities (agents). These agents typically represent resources and/or tasks. Allocation of tasks to resources is done using dynamic market mechanisms. This yields a simple and fault-tolerant system, since none of the agents need *a-priori* information about the

other agents. As a consequence, several disturbances and changes can easily be coped with. Nevertheless, the basic assumption of this architecture paradigm gives rise to its principle drawbacks that impede the widespread use of these kinds of control systems in industrial environments: the independence of agents prohibits the use of global information. Therefore, global system performance is very sensitive to the definition of the market rules; the control system can not guarantee a minimum performance level in the case of unforeseen circumstances; prediction of the behavior of individual orders is impossible, etc. In order to tackle this problem the Intelligent Manufacturing Systems (IMS) consortium¹ initiated a few projects in the early 1990s to define new paradigms for the factory of the “future”. The holonic manufacturing system was one of these sets of projects.

In the following sections the holonic background is presented. The state-of-the-art in the field of holonic manufacturing systems is also studied.

2.1 Holon

The holonic concept was developed by the philosopher Arthur Koestler [6] in order to explain the evolution of biological and social systems. On the one hand, these systems develop stable intermediate forms during evolution that are self-reliant. On the other hand, in living and organizational systems it is difficult to distinguish between “wholes” and “parts”: almost everything is both a whole and a part, at the same time. These observations led Koestler to propose the word “*holon*”, which is a combination of the Greek word “*holos*”, meaning whole, and the Greek suffix “*on*”, meaning particle or part, as in proton or neutron. Koestler observed that, in living organisms and in social organizations, which are entirely self-supporting, noninteracting entities did not exist. Every identifiable unit of organization, such as a single cell in an animal or family unit in a society, comprises more basic units (plasma and nucleus, parents and siblings), while at the same time forming a part of a larger unit of organization (a muscle tissue or a community). The strength of holonic organization, or *holarchy*, is that it enables the construction of very complex systems that are nonetheless efficient in the use of resources, highly resilient to disturbance (both internal and external), and adaptable to changes in the environment in which they exist. Within a holarchy, holons may dynamically create and change hierarchies. Moreover, holons may participate in multiple hierarchies simultaneously. Holarchies are recursive in the sense that a holon may itself be an entire holarchy that acts as an autonomous and cooperative unit in the first holarchy.

The stability of holons and holarchies stems from holons being self-reliant units, which have a degree of independence and handle circumstances and problems on their particular level of existence without asking higher-level holons for assistance. Holons can also receive instructions from and, to a certain extent, be controlled by higher-level holons. This self-reliant characteristic ensures that holons are stable

¹ <http://www.ims.org/>

and able to survive disturbance. The subordination to higher-level holons ensures the effective operation of the larger whole.

2.2 Holonic Manufacturing Systems – HMS

The application of holonic concepts to manufacturing was initially motivated by the inability of existing manufacturing systems (i) to deal with the evolution of products within an existing production facility and (ii) to maintain satisfactory performance levels outside normal operating conditions [4]. Suda introduced the concept of holonic manufacturing in the early 1990s [7] to address the challenge for manufacturing in the 21st century.

Teams of industry experts, scientists, and engineers from the world's leading industrial nations worked together from 1992 to 1994 to build and test a framework for international collaboration in intelligent manufacturing systems (IMS). The experiences of teams coming together from Australia, Canada, Europe, Japan and the USA to work for one year on collaborative "test case" projects formed part of a two-year feasibility study that began in February 1992. This feasibility study proved that this kind of international collaboration could achieve significant results in a relatively short time.

A holonic manufacturing system is based on the concept of "*holonic systems*", developed by Arthur Koestler [6]. Holons in a holonic manufacturing system assist the operator in controlling the system: holons autonomously select appropriate parameter settings, find their own strategies and build their own structure.

Koestler also points out that holons are autonomous self-reliant units, which have a degree of independence and handle contingencies without asking higher authorities for instructions. Simultaneously, holons are subject to control from (multiple) higher authorities. The first property ensures that holons are stable forms, which survive disturbances. The latter property signifies that they are intermediate forms, which provide the proper functionality for the greater whole. Finally, Koestler defines a holarchy as a hierarchy of self-regulating holons that function (a) as autonomous wholes in supraordination to their parts, (b) as dependent parts in subordination to controls on higher levels, (c) in coordination with their local environment.

Work in the HMS program has translated these concepts to the manufacturing world, viewing the manufacturing system as one consisting of autonomous modules (holons) with distributed control. The goal is to attain the benefits that holonic organization provides to living organisms and societies, in manufacturing, i.e., stability in the face of disturbances, adaptability and flexibility in the face of change, and efficient use of available resources. The HMS concept combines the best features of hierarchical and heterarchical organization [8]. It preserves the stability of hierarchy while providing the dynamic flexibility of heterarchy.

The HMS consortium developed the following list of definitions to help understand and guide the translation of holonic concepts into a manufacturing setting [4]:

- **Holon:** An autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. Figure 2.1 shows the holon general architecture widely used in the field [9]. A holon can be part of another holon.
- **Autonomy:** The capability of an entity to create and control the execution of its own plans and/or strategies.
- **Cooperation:** A process whereby a set of entities develops mutually acceptable plans and executes those plans.
- **Holarchy:** A system of holons that can cooperate to achieve a goal or objective. The holarchy defines the basic rules for the cooperation of the holons and thereby limits their autonomy.
- **Holonic manufacturing system:** a holarchy that integrates the entire range of manufacturing activities from order booking through design, production, and marketing to realize the agile manufacturing enterprise.

2.3 HMS State-of-the-Art

In the last ten years, an increasing amount of research has been devoted to HMS over a broad range of both theoretical issues and industrial applications. In this section we summarize the main developments reported on specialist HMS literature covering: holon architecture, holons interconnection, holons operation, algorithms for holonic control and methodologies for HMS development.

2.3.1 Holon Architecture

A manufacturing control system for production processes is composed of software modules as well as different physical elements of the manufacturing environment: resources, products, client work orders, coordination operations, etc. The software module and the physical entity, bonded by means of an appropriate communication network, represent a holon in a manufacturing system. Every such holon will be able to reason, make decisions, and communicate interactively with other holons. The number and types of software modules, and the way this software part and the physical entities are interconnected, define the different holon architectures approaches.

The first holon general architecture was proposed by Christensen in 1994 [10]. Figure 2.1 shows the two main components of this architecture: *physical processing part* and *information processing part*. The *physical processing part* is optional. Some examples of holons without a physical processing part are work-order holons, planning holons, scheduler holons, etc. The *physical processing part* is divided into: the physical processing itself, that is the hardware that executes the manufacturing

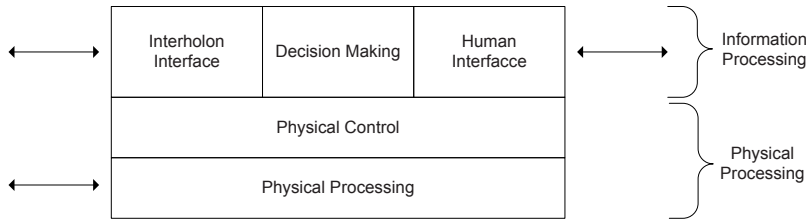


Fig. 2.1 Holon general architecture

operation; and the physical control, a controller (NC, CNC, DNC, PLC) that controls the hardware operation. The *information processing part* is made up of three modules: the holon's kernel or decision making, which is in charge of the holon's reasoning capabilities and decision making; the interholon interface, for the communication and interaction with other holons, and; the human interface, for input (operation commands) and output (state monitoring) data for humans.

An agent-based architecture for the information processing part of Christensen general architecture is proposed in [11]. This proposal is based on the holonic vision of autonomous and cooperation entities. Three main aspects guide this approach. Firstly, holons are entities with autonomous control over the machine behavior they are associated with. Holons may create and execute their own plans and follow their own strategies. This autonomous behavior implies some kind of decision-making component that guides the holon physical control. Secondly, two or more holons are able to cooperate when and wherever it is necessary. To do this, these holons are able to figure out cooperation opportunities, make cooperation or negotiation commitments, and finally to execute the cooperation committed to. Thirdly, holons are able to act in multiple organizations called holarchies and these holarchies are created and modified dynamically. Creating a holarchy means to aggregate the manufacturing process or the controlling process in order to enhance productivity. This implies work and responsibility distribution, and the definition of interaction patterns, which means that holons are able to figure out opportunities for reorganization, negotiate reorganization, and follow the interaction patterns.

The inclusion of these components into the general architecture of Christensen led the authors to propose the agent-oriented architecture in Fig. 2.2. In order to figure out physical behavior and taking into account the current situation, the holon chooses the appropriate plans and strategies in order to reach its long-term goals. These plans and strategies are communicated from the *decision-making* module to the *behavior control* in order to translate them into hardware operations. On the other hand, the cooperation interactions are initiated by the *decision-making* module and executed by the specific *cooperation techniques* using the *communication techniques* (domain ontologies and languages). In order to reorganize the manufacturing controlling processes the holon needs techniques. These techniques are used to monitor other component actions and to analyze the controlling process. In this way, the holons can figure out opportunities for improvement and can start a negoti-

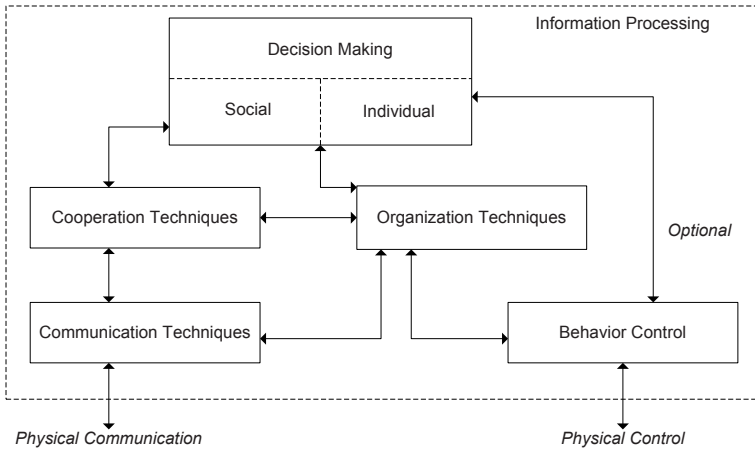


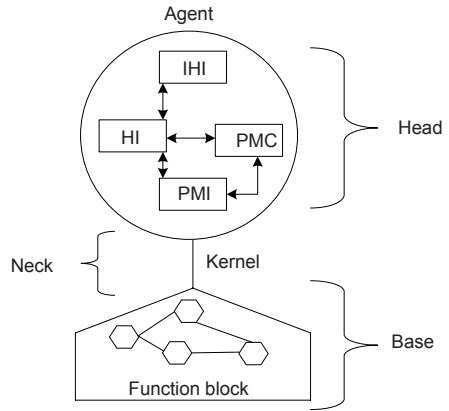
Fig. 2.2 Holon agent-based architecture

ation process for reorganization (*organization techniques*) that is executed by means of cooperation standards.

The Keele University HMS research group, proposes a holon architecture using agents and function blocks [12, 13]. A manufacturing holon is usually composed of knowledge and software modules, as well as an optional hardware component. In terms of functionality, a holon may be considered a composition of an intelligent controlling system (*head*) and a processing system (*base*), Fig. 2.3. The *head* of the holon is based on an agent architecture made up of modules defined in [10]. The elements of the intelligent controlling system are: the *PMC* (process/machine control) executes controlling plans for the running processes; the *PMI* (process/machine interface) provides the logic and physical interface for the processing system through a communication net; the *HI* (human interface) provides the human-readable interface; the *IHI* (interholon interface) is in charge of interholonic communication. The processing system incorporates all the processing modules needed to carry out the production activities. In this way, the *ICS* lets the holon supply the production facilities as autonomous subsystems in coordination with the environment and with other holons. The processing system is responsible for the manufacturing functions defined by the operation rules and strategies imposed by the *ICS*.

In the agent and function block (IEC 61499 [14, 15]) integrated architecture, agents are used to manage high-level planning strategies (*ICS-Head*), while function blocks manage real-time process/machine low-level control (*Base*). There is a holonic kernel running over the function blocks in order to provide the necessary interface between the agents and the IEC 61499. The holarchies and holons' interactions based on agents are organized by means of a structure called a cooperation domain, see Fig. 2.4. A cooperation domain (*composed holon*) is a logic space through which: (i) holons communicate and cooperate, and; (ii) an environment in which holons can find, contact and interact with other holons is provided.

Fig. 2.3 Holon architecture based on agents and function blocks



A cooperation domain is not feasible on its own but there must at least be a member holon. Cooperation domains are dynamically created by the execution of the holons' functional components. A holonic system is composed of at least one cooperation domain. A holon can simultaneously be a member of one or more cooperation domains. Every domain is led by a coordinator that is the interface with the outside (other domains). A holon can join a cooperation domain, query its attributes, interchange information with other holons, and exit the domain whenever the holon has finished its task. A cooperation protocol is executed in order to assign tasks to holons in the cooperation domain.

Deen and Fletcher propose a computational model for tasks redistribution (hierarchy reorganization). The model is based on *temperature equilibrium* concepts [16]. When a delayed task is processed, the holons may experience overloads and

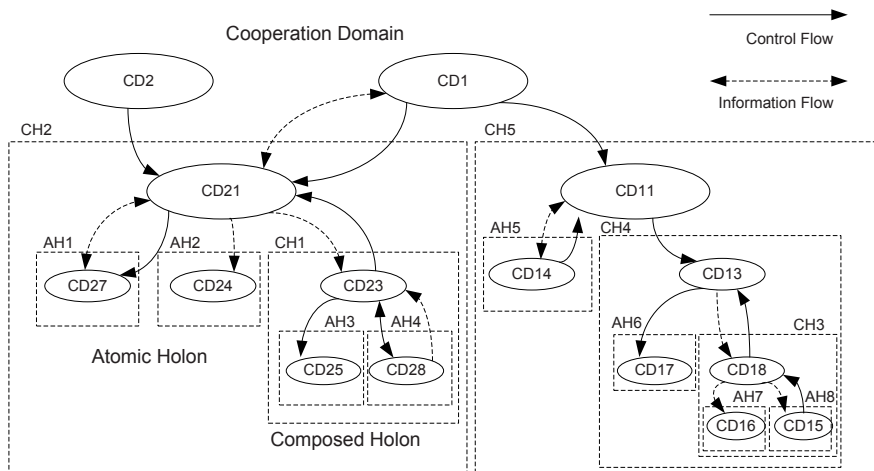


Fig. 2.4 A cooperation domain of the agent and function block integrated architecture

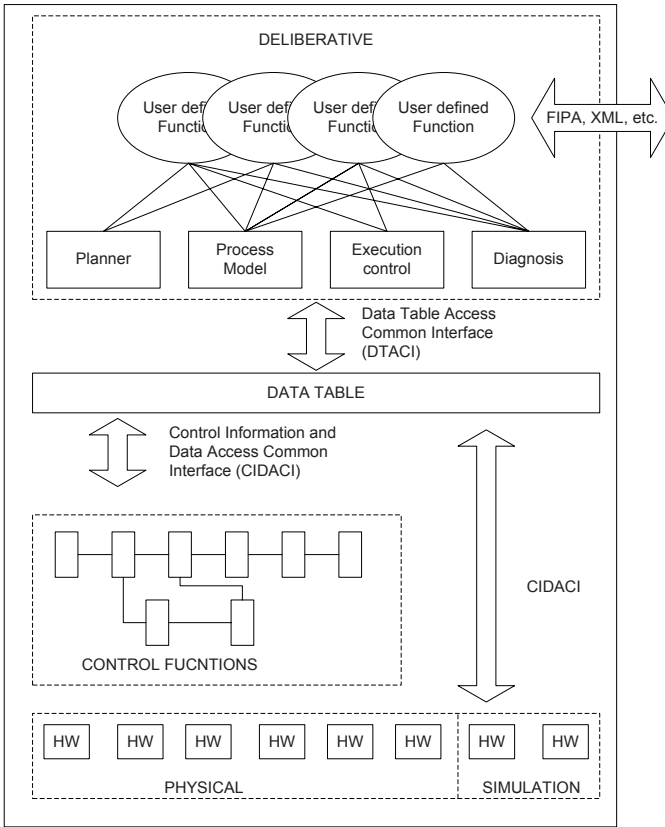


Fig. 2.5 The HCD architecture

this makes the holon “hot”. In this way, when a holon realizes that its temperature is over a predefined threshold, it informs the other holons in the holarchy of the situation. If there is a “cool” holon that can manage the task in-hand, it starts a negotiation interaction with the hot holon in order to transfer the task. The auto-organization of the entire system is achieved when the holarchies that make up the system try to maintain a temperature equilibrium.

In [17], Brennan and Norrie propose a holonic agent architecture, using agents for the deliberative layer and function blocks for the physical control layer. In [18] a holonic architecture for device control (*HDC*) is illustrated. In Fig. 2.5 we can see the architecture components. The deliberative layer has two purposes: application-domain-specific functionalities, and generic functionalities. The generic functionality is defined by the *planner*, *process model*, *execution control* and *diagnosis* modules. The deliberative layer communicates with the other layers by means of the device *data table* through the *DTACI* (Data Table Access Common Interface). The deliberative and control layers can read and write from and to the *DTACI*. The *control functions* layer is the user-defined application that controls the hardware

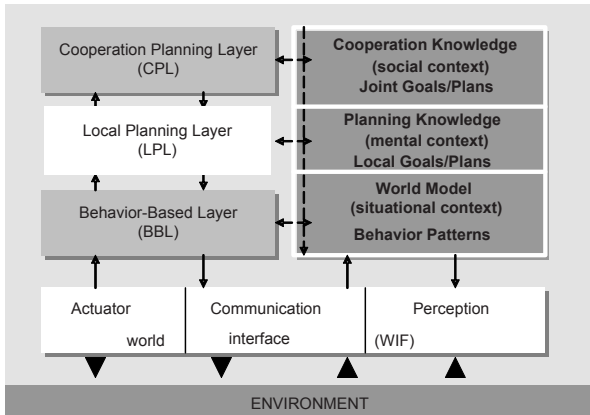


Fig. 2.6 The INTERRAP architecture

and physical process behavior of the *HDC*. On this level function blocks are used. The *physical* layer represents the hardware (sensors and actuators) controlled by the *HDC*. The *simulation* layer is the simulation of the hardware.

The German Research Center for Artificial Intelligence (DFKI) has developed an agent-based architecture to implement holonic system [19]. The architecture is based on the three concurrent layer agent architecture INTERRAP of Müller [20]. Figure 2.6 shows the INTERRAP architecture, in which the composition and configuration of the holonic structures are implemented in the *cooperative planning layer (CPL)*. The *CPL* provides the communication, negotiation and administration functionalities for the holonic structures. They have used this architecture in application domains such as: supply webs, HMS, virtual enterprise logistics and agent-based knowledge source.

In [9] Christensen overviews the holonic architecture approaches most used in the HMS field and proposes as standard an integrated architecture of function blocks and cooperation domains, very similar to the work of Fletcher et al. [12]. For low-level control he proposes the IEC 61499 standard [14, 15], while for high-level control, that is holons' negotiation/coordination in holarchies, he proposes using FIPA agents [21].

2.3.2 Holons Interconnection

The way in which the holons interconnect among themselves defines holon organization patterns that can be useful for modeling and implementing cooperation holarchies using predefined structures.

The group of Mechanics and Manufacturing Engineering of Calgary University has developed many projects related to models for manufacturing systems intelligent

control. Some of these projects are: MetaMorph I, ABCDE (agent based concurrent design environment), DIDE (distributed design environment), FBIICDE (feature-based integrated and intelligent concurrent design system) and MetaMorph II. These projects are based on a factory intelligent control distributed architecture [17, 18].

The main feature of the systems based on MetaMorph [22] is their changing structure. These systems adapt to emerging tasks and changing environments. The MetaMorph architecture uses the domain cooperation concept developed by Deen and Fletcher (see Sect. 2.3.1) but they call it a *dynamic virtual cluster*. In MetaMorph there are also types of holons or primary holons, as in PROSA (see Sect. 2.3.3). There are product holons, product model holons, and resource holons. A resource holon is dual, on the one hand it has a physical component, which is the product itself from the beginning to the end; and on the other hand, it stores information related to the process status and the product components during the manufacturing process. A product model holon stores the product life-cycle information, configuration data, design specification, process plan, materials list, quality data, etc. The resource holons are used to model manufacturing devices and operations.

The coordination and auto-organization are implemented by means of the dynamic virtual clusters (Fig. 2.7). In a dynamic virtual cluster the holons may participate dynamically in different clusters (holarchies) and may cooperate through cooperation domains. The primary holons fulfill the same task as the coordinator holons of Deen and Fletcher, and are cluster managers coordinating the holons' interactions. The cluster exists while the cooperation task is active, that is, when the task is completed the cluster disappears. The process cycle for a virtual cluster can be defined as: (1) The primary holon joins some or all the contracts (production orders, cooperation orders, etc.) in a new task. After a replanning and analysis processes, the primary holon lists the cooperation requirements as cooperation tasks. (2) A mediation holon is created in order to find a list of potential cooperating holons. (3) The potential cooperating holons are invited to enter the virtual cluster. These holons decide whether or not to participate and send proposals for all the tasks they

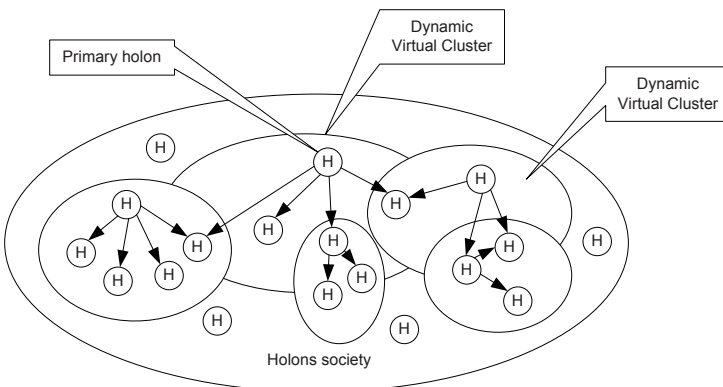


Fig. 2.7 Holons society and holarchy

are interested in. (4) All the proposals are collected and evaluated by the primary holon and the mediator holon. When the optimal tasks assignment is determined, contacts are established directly between the primary holon and the subcontracted holon. The virtual cluster is built among the primary holon and all the subcontracted holons. (5) The associated cluster, the mediator and the cooperation links disappear when the tasks are completed.

2.3.3 Holons Operation

In a manufacturing system there are many types of operation or functionalities that are crucial for the production process. The way in which these operations are identified and modeled is another research and development subject in the HMS field.

PROSA (Product-Resource-Order-Staff Architecture) [23], is the reference holonic architecture for HMS that has been widely adopted. Basically, PROSA is an interholonic architecture, which identifies the types of holons necessary for any manufacturing system, its responsibilities, and the interaction structure in which they cooperate. The architecture is made up of three basic holons, Fig. 2.8: *work-order holon*, *product holon*, and *resource holon*. These holons are specified using object-oriented concepts such as aggregation and specialization. Each of the basic holons is responsible for one of the following manufacturing controlling aspects: internal logistic, manufacturing planning, and resource management. In order to assist the basic holons, with expert knowledge, a “*staff*” *holon* can be added. The structure of the entire manufacturing system is a dual holarchy divided into one subholarchy of resources assignment (*work-order holons*, *resource holons*, and *staff holons*) and one subholarchy of process control (*product holon* and the *resource holon* controlling process parts). A *resource holon* has a physical part (that is, a production resource of a manufacturing system) and an information processing part that

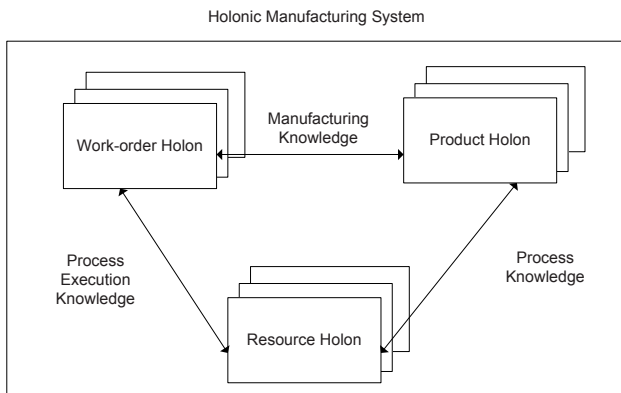


Fig. 2.8 PROSA: reference architecture

controls the resource. This holon offers production capacity and functionality to the other holons. A *product holon* stores the process and product knowledge needed to insure the correct manufacture of the product. It acts as an information server for the other holons in the HMS. A *work-order holon* represents a task in a manufacturing system. It is responsible for doing the work assigned on time and in the right way. It manages the physical products that are being produced, the product status models, and all the logistic processing information related to the task. A coordination and controlling technique for a PROSA holarchy using swarm-inspired social behavior is described in [24].

The ADACOR architecture (adaptive holonic control architecture) [25] proposes a holonic approach for the dynamic adaptation and agility in the face of disturbances in FMSs (flexible manufacturing systems). The architecture is based on a group of autonomous, intelligent and cooperative entities (holons), in order to represent the factory components. These distributed components can be either physical resources (numerical controllers, robots, programmable controllers, etc.) or logical entities (products, orders, etc.). ADACOR groups the holons of a manufacturing system into product holons, task holons, operational holons, and supervisor holons [26]. Each product is represented by one product holon that has all of the product-related knowledge and is responsible for the process planning. The product holon receives the product manufacturing orders. To this end, it stores information about the product structure and the process planning to produce it. Every manufacturing order is represented by a task holon, which is responsible for controlling and supervising the production plan execution. It includes the order decomposition, the resource assignment plan and the execution of this plan. The operational holons represent the physical resources of the factory, such as human workers, robots and machines. They manage the behavior of these resources based on their goals, constraints and capabilities, and try to optimize their agenda. The product, task and operational holons are very similar to the product, order and resource holons of PROSA [23]. The ADACOR supervisor holon is the PROSA staff holon, and is in charge of coordination and global optimization tasks, coordinating various operational and supervisor holons.

The HCBA (holonic-component-based-architecture) [27] is derived from CBD (component-based development) and HMS. HCBA defines two major holons: product and resource. The resource holon is an embedded system component that can execute operations such as production, assembly, transportation, and checking. The product holon may contain a physical part and a controlling part. The physical part can be, for example raw materials, product parts and pallets. The controlling part may represent the path controlling a production line, process control, decision making and product information. The holonic system is built associating these two types of holons, building nested structures of products and resources.

2.3.4 Holonic Control

In this section some works related to algorithm definition and implementation for holonic control are presented. These works can be grouped into four categories of controlling activities: work order programming, scheduling, work-order execution and job-shop control, and device controlling.

- *Work-order programming*: The holonic programming of production operations has been studied in [28, 29, 30, 31, 32, 33]. These approaches usually deal with an interaction scenario in which the product holon is in charge of determining the necessary parts or sub-assemblies and the manufacturing operations. The type of resources associated with every operation and the sequence is determined by means of a set of cooperation interactions with the resource holons.

The benefits of the holonic approach compared with traditional production programming approaches are due to the distributed nature of the planning process, the interactive cooperation interaction among the production components, and the easy incorporation of new products or resources.
- *Scheduling*: A significant research effort has been devoted on holonic scheduling algorithms. Much of these works focused on flexible manufacturing systems [28, 34, 35], assembly lines [36], job-shop [37], assembly and machining work-cells [38, 39, 40], continuous process lines [41, 42] and factory maintenance [43]. In [32, 44, 45] generic scheduling methods for holonic manufacturing were proposed. The main reason for the large number of activities in this field is the maturity level of the intelligent scheduling techniques [46, 47] and the algorithms for factory distributed control [8, 48, 49], due to the fact that both of them are similar to the holonic approach. They attempt to assign time and resources in a more dynamic way than can be done with offline scheduling methods.

The major feature of a holonic scheduling approach is that every holon is a problem-solver and a decision-making entity. They use cooperation strategies in order to exchange information and mutually accepted solutions. There is a mechanism to assure that global system constraints are satisfied. And finally, there is a central coordination mechanism.

The benefits of a holonic scheduling approach compared with traditional approaches are due to the computation and decision-making distribution, and the interactive nature of holons.
- *Work-order execution and job-shop control*: This activity involves the initiation, control, monitoring and termination of tasks and involves actual plans and actual production settings. Work-order execution was studied in [38, 39, 50, 51]. The new elements of the holonic approach in contrast to conventional execution controlling algorithms are [3]: the execution is implemented by means of a negotiation interaction sequence; and the resources (machines) executing the manufacturing operations are responsible for the decision-making regarding the timing and the type of execution.
- *Device controlling*: The device control – which involves actuation, sensing and feedback control of the physical operations that support a machine – has been

largely studied in the HMS field as a conventional control problem [52, 53, 54, 55, 56]. These studies focus on achieving an effective device interface.

2.3.5 Methods for HMS Development

Manufacturing systems are very large and complex. Accordingly, the holonic systems that control or implement them are also large and complex. The development process of these kinds of systems has to be guided by software engineering methods and principles in order to help the engineer in the development process of the HMS itself. To date, almost all of the applications in the HMS field have been built using no design or development method. Moreover, there are probably no two applications developed with the same method. Perhaps this situation has come about because there has been little work done on methods for HMS development.

In [57], a *formal specification approach for HMS control* is presented, but it is still in a developmental stage. There are no defined development phases, and no detailed descriptions to explain how to model issues such as cooperation in the holarchy, holon autonomy and system flexibility. In [58], an agent organization is proposed to model each holon/holarchy that is independent of any holon architecture. However, it is focused only on the holarchy definition and does not define the development phases. The development of HMS is studied in detail in Chap. 4.

There is a definite need to have methodologies for holonic systems [3], that are based on software engineering principles in order to assist the system designer at each stage of development. This methodology should provide clear, unambiguous analysis and design guidelines. In order to fill this gap, we present in this book, ANEMONA, a multiagent method specifically conceived for HMS development, which tries to fulfill all the HMS modeling requirements.

2.4 Conclusions

In this chapter we have discussed the background of HMS and analyzed the state-of-the-art in HMS. We have attempted to present a global overview of the field, covering the different studies worked on: architecture (Sect. 2.3.1), holons interconnection (Sect. 2.3.2), holons operation (Sect. 2.3.3), holonic control (Sect. 2.3.4), and methods for HMS development (Sect. 2.3.5).

The more active fields are those related to developing holonic control systems. From these developments we can conclude that currently multi-agent system technology is the tool most utilized for developing HMS. Nevertheless, there is very little work on methods for HMS development. The focus of this book is HMS development. To this end, in Chap. 4 we study HMS development requirements in depth and present ANEMONA, a HMS development methodology.

Chapter 3

Holons and Agents

In this chapter we study the conceptual similarities between holons and agents. Despite these two modeling approaches being very similar, there are some peculiarities when analyzing their characteristics. In this chapter we review a study on every characteristic of holons and agents. The complete analysis can be found in [59].

Before analyzing the characteristics of agents and holons, we introduce the concepts of agent and multi-agent systems. Finally, we present the abstract agent notion as a modeling artifact for autonomous entities with recursive structures [60]. The abstract agent extends the traditional agent definition, adding a structural perspective to the agent concept: "... an abstract agent can be an agent; or it can be a MAS made up of abstract agents ...".

3.1 Agents

An agent is an autonomous and flexible computational system that is able to act in an environment [5]. Flexible means that the agent is:

- *Reactive*: it reacts to the environment it is in.
- *Proactive*: it is able to try to fulfill its own plan or goals.
- *Social*: it is able to communicate with other agents by means of a language.

Some properties that are usually attributed to agents to a greater or lesser degree, for solving particular problems are [61, 62]:

- **Autonomy**: agents can operate without the direct intervention of humans or other agents.
- **Social ability**: agents are able to interact with other agents (human or not) through an agent communication language.
- **Rationality**: an agent can reason about perceived data in order to compute an optimal solution.
- **Reactivity**: agents are able to perceive the environment's stimulus and these stimuli guide the agents' actions in their environment.

- Proactiveness: agents are not only stimulus-reacting entities, but also have an enterprising character and can act guided by their own goals.
- Adaptability: related to the learning that an agent may do and its capabilities to change its own behavior based on this learning.
- Mobility: the capability of an agent to move through a network.
- Veracity: an agent cannot deliberately provide false information.
- Benevolence: an agent is willing to help other agents as long as it is not contrary to its own goals.

A multi-agent system (MAS) is a computational system where two or more agents interact (cooperate or compete, or a combination of the two) to achieve some individual or collective goals, and the achievement of these goals is beyond the individual capabilities and individual knowledge of each agent.

MAS studies the coordination of intelligent behavior among a group of (possibly pre-existing) autonomous intelligent agents. It is focused on the individual behavior from which the system behavior follows. Today, MAS is a very active area of research and is beginning to see commercial and industrial applications. MAS is centered on the social behavior of intelligent entities and focuses mainly on investigating behavior models, cooperation and coordination strategies, intelligent brokerage, task-performance optimization, learning from own experiences, coalition formation, etc.

In summary, MAS is a general software technology motivated by fundamental research queries on subjects like autonomy, cooperation, group formation, etc. It has focused on answering questions like “*what can be done?*” and “*how can it be done?*”, and is applicable to a large range of domains: e-commerce, intelligent manufacturing control, robotics, information recovery, etc.

3.2 Holons and Agents: Two Similar Modeling Notions

In this section, we review the characteristics of holons and agents and analyze the similarities among them. In the first case, following the generalized trend in the field of intelligent manufacturing systems, we indicate the motivation of both approaches; then we identify those properties that define both approaches in order to make an analysis of each one.

As we mentioned in the previous section, agents are: autonomous, social, reactive, proactive, rational, mobile, able to learn, etc. [5]. Holons, as a paradigm, have the following basic characteristics: autonomy, cooperation and reorganization (openness). In addition to these characteristics, which could be called “behavioral properties”, holons have “structural properties”. One of them is “recursiveness”, which allows holons to be made internally of self-similar entities (holons), which again can be made of holons, and so on (until we reach an atomic level in which a new subdivision is impossible or useless for the domain application). Another important structural property, as was defined by the HMS consortium [4], is that

holons usually comprise an information processing part with an optional physical processing part.

In the following subsections we analyze and compare each property separately.

3.2.1 *Autonomy*

The behavior of autonomous entities can be based on both their own experiences and the built-in knowledge used in constructing the entities for the particular environment in which they operate. Agents have been successfully used in domains where the degree of uncertainty and unpredictability requires processing units that are capable of autonomous action, without the direct intervention of humans or others.

Manufacturing control systems are large and complex systems, designed to carry out a clearly defined task. However, in a manufacturing system, things rarely go as expected. The system may be asked to do additional tasks that were not anticipated and is sometimes allowed to omit certain tasks. The resources available to perform tasks can become unavailable, and additional resources introduced. The start time and processing time of a task are also subject to variation. A task can take more time than anticipated or less time than anticipated, and tasks can arrive early or late. These are some of the reasons why holons are, by definition, autonomous entities that must be able to create, control and monitor the execution of their own plans and/or strategies, and to take suitable corrective actions against their own malfunctions [4]. In this sense, it can be said that agents and holons share this property.

3.2.2 *Reactivity*

An agent is responsive to events that occur in its environment, where these events affect either the agent's goals or the assumptions that underpin the procedures that the agent is executing in order to achieve its goals. So, the effects of environment stimulus may be changes in the agent's goals or assumptions, or the agent's actions effecting changes in the environment. In the same way, holons need to react to changes in their environment. Such changes may affect their initial goals or prevent the execution of the current or future planned tasks.

Let's suppose a simplified manufacturing system (see the gray area in the manufacturing processing section in Fig. 3.1). There are three types of holons: (i) order holons that represent tasks in the manufacturing system; (ii) resource holons, which are production resources in the manufacturing system, and; (iii) product holons, the products themselves. OH1 (order holon 1) processes a task to produce PH1 (product holon 1). The production process is carried out in processing steps assigned to resource holons RH1, RH2 and RH3, such that RH1 is the first processing step, RH2 is the second and RH3 the third. For some reason, RH1 takes more time for process-

ing than the initial estimated time. RH2 and RH3 have to be aware of this situation, which prevents the execution of their current plans. They must react in some way, by searching for other order holons to take advantage of their production power, for example, or stopping until RH1 finishes its processing (obviously, the second alternative is less productive for overall system performance, but it is a reactive action too). Despite the fact that this example is a simplified and reduced one, it reflects the reactive property of holons.

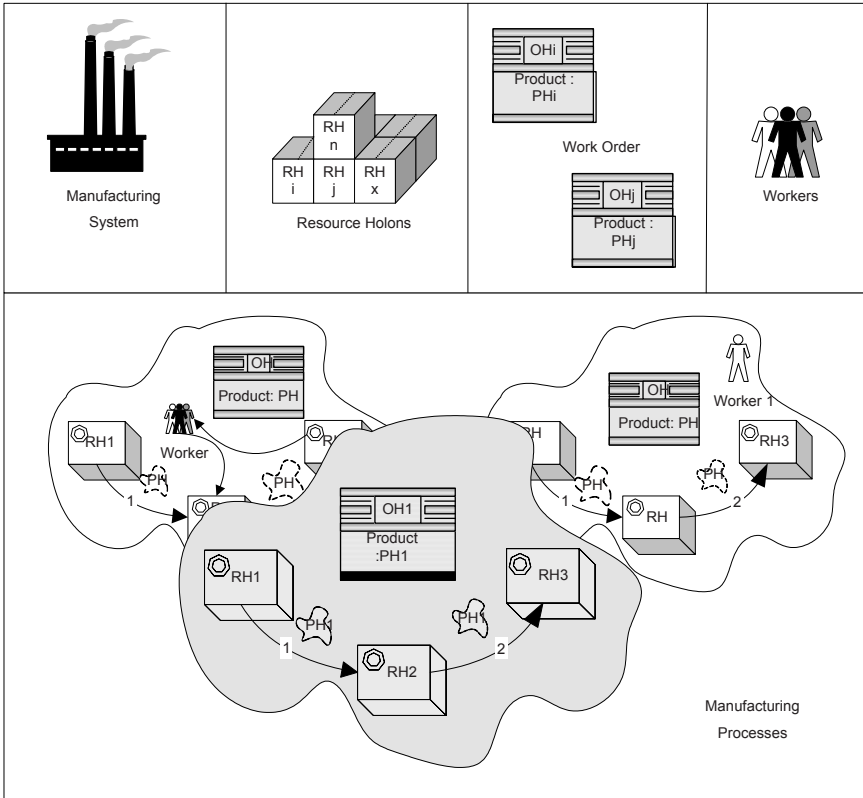


Fig. 3.1 A simplified HMS

3.2.3 Proactivity

Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by *taking the initiative* [5]. It is not hard to build a system that exhibits goal-directed behavior, as pointed out by Wooldridge and Jennings.

When we write such a procedure, we describe it in terms of its precondition and postcondition. The effects of the procedure are its goal. If the precondition holds when the procedure is invoked, then we expect the procedure to execute correctly: the goal will be achieved. But systems of this kind assume that the environment does not change while the procedure is executing. Similarly, it is assumed that the goal, that is, the reason for executing the procedure, remains valid at least until the procedure terminates.

As we pointed out in previous sections, the domain application area of agents and holons changes constantly, preventing them from blindly executing a procedure without regards to whether the assumptions underpinning the procedure are valid. This implies that agents and holons need a balance between goal-directed (proactive) and reactive behavior. They will attempt to achieve their goal systematically, but they will also be able to react to a new situation in time for the reaction to be of some use. Let's suppose that in the previous example, RH1, RH2 and RH3 have as a goal: "to maximize their utilization". Before RH1 begins to slow down its processing, RH2 and RH3 have a certain slot time assigned to PH1, so they look for other products to get a greater utilization of their time and processing power, i.e. to achieve their goal. When RH2 and RH3 realize that RH1 is taking more time than was expected, RH2 and RH3 must reschedule the slot time assigned to PH1 as well as the slots time and the processing power for the new products found in their attempt to achieve their goal. This simple example reflects the proactive property of holons.

3.2.4 Sociability

As agents do not usually act in isolation but in the presence of other agents or humans, they need sociability and interactive behavior to communicate, cooperate, coordinate and negotiate with them. One question immediately arises: *How are they enabled to interact and especially communicate with other agents?* The answer is: *By agent communication languages (ACL)*. But what is an ACL? An ACL, as proposed by Austin [63], "...is the medium through which the attitudes regarding the content of the exchange between agents are communicated; it suggests whether the content of a communication is an assertion, a request, a query, etc.". Common standards for ACL include the Knowledge Query and Manipulation Language (KQML) [64] and a proposal by the Foundation for Intelligent Physical Agents (FIPA), based on the language Arcol – ARtimis COmmunication Language, [65]. These developments are, in part, rivalled by increasingly sophisticated Internet markup languages like XML [66], which can also be used by agents. In addition, interacting agents need to have the same understanding of a particular vocabulary. This problem is solved by ontologies that cover every domain where agents operate. An ontology is a computerized representation or model of a specific part of the world.

A holon's social ability is included in its capacity for cooperation (see the next subsection), since holons need a means of communication with other holons to be

able to cooperate. In the example above, the three types of holons need social abilities to exchange information about the manufacturing process, thus enabling the execution of tasks. For example, OH1 needs resources to produce PH1, so OH1 interacts with resource holons RH1, RH2 and RH3 to obtain processing functionalities and properties specific to the operation, such as high quality or high throughput. On the other hand, RH1, RH2 and RH3 try to maximize their utilization, and PH1 focuses on the following operations in order to get processed by RH1, RH2 and RH3. As can be deduced from this simple example, holons need to interact with each other. Therefore, as agents, they need social abilities with all their associated problems.

In manufacturing systems, people and computers need to be integrated, with access to required knowledge and information, in order to work collectively at various stages of product development [4]. This requirement led Christensen to propose an integrated human-interface block (see Figure 2.1) on his holon general architecture [10]. Each holon must always be able to cooperate with humans, whereas in MAS, the human interface is implemented by one or several specialized agents who provide communication services as a whole. Nevertheless, nothing in the definition of agents, prevents them having an integrated human-interface block.

3.2.5 Cooperation

Cooperation is a means of social ability. Manufacturing enterprises have to fully cooperate with their suppliers and customers for the supply of materials, parts fabrication, final product commercialization, and so on. Such cooperation should be efficient and with a quick response. Cooperation is an imperative requirement for any complete functional model for advanced manufacturing systems [4]. Moreover, all manufacturing units cooperate in order to achieve the overall manufacturing goals. With respect to these goals, a holon never deliberately rejects cooperation with another holon. Only when the requested actions are impossible or strongly disadvantageous to the manufacturing process, does it refuse their execution [11]. Coordination, negotiation, bargaining, and other cooperation techniques allow holons to flexibly interact with other holons in an abstract way. Regarding cooperation, a great number of specific approaches exist in MAS: see [67] and [68] for an overview. A sample cooperation scenario was presented in the social ability subsection.

3.2.6 Openness

Koestler [6], defines a holarchy as a hierarchy of holons that function (a) as autonomous wholes in supraordination to their parts, (b) as dependent parts in subordination to controls on higher levels, (c) in coordination with their local environment. This definition leads to a heterarchy of holons that is a mixture of hierarchy and hor-

izational organization. HMS widely explores holarchy principles to create integrated collections of several lower-levels holons. As indicated by Brennan and Norrie [69], the holarchy notion can be implemented using several MAS architecture approaches for federations such as facilitators, brokers, or mediators.

Holons are able to act within multiple organizations (holarchies), which are created and changed dynamically. Real-world manufacturing environments are highly dynamic because of diverse, frequently changing situations: bank rates change overnight, materials do not arrive on time, power supplies break down, production facilities fail, workers are absent, new orders arrive and existing orders are changed or canceled, etc. [4]. Such changes lead to deviations from existing plans and schedules. It is necessary, therefore, for the working system to adapt to such a changing environment. That is why holarchies must be open organizations, which must be able to accommodate the incorporation of new holons, the removal of existing holons, the reorganization of the work load, etc. Apart from cooperation, holons require techniques to reorganize the control of the manufacturing processes. These techniques monitor other component actions, that is, physical and communication actions, and analyze the control process. In this way, they may identify the possibilities for improvement and start a negotiation process for the organization using standard cooperation techniques. The implications of the agreed reorganization are distributed to the other holons' components. In MAS, while there are many studies regarding cooperation, there is little work on reorganization [11].

3.2.7 Rationality

A rational agent is one that does the right thing, i.e., an action that causes the agent to be the most successful [70]. The actions an agent performs can be understood as goals. Galliers proposes, in [71], a definition of rationality: "...*(crudely) the assumption that an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved – at least insofar as its beliefs permit*". A rational agent acts according to: the percept sequence, what the agent knows about its environment, and the actions that the agent can perform. These three things will determine the success of the agent.

Although this property does not explicitly appear in the holon definition, it can be derived from a stronger definition of autonomy: "*a system is autonomous to the extent that its behavior is determined by its own experiences*" [70]. And the general assumption is that a holon always attempts to obtain the best overall system performance.

3.2.8 *Mental Attitudes*

In recent years, a number of approaches have been proposed to specify rational agents in terms of mental attitudes such as knowledge, beliefs, wants, goals, commitment, and intention. However, there is no clear consensus in the agent community about precisely which combination of mental attitudes is best suited to characterizing agents. However, there seems to be an agreement that beliefs (or knowledge) should be taken as one of the basic notions of agent theory [5].

As was pointed out in previews sections, although manufacturing processes experience several changes and disturbances, the degree of uncertainty and unpredictability is not comparable to that of other domain applications of agents. As a consequence, manufacturing applications require less mental and social deliberation than typical applications of multi-agent systems [11]. Manufacturing control units (holons) must reason about the behavior of the manufacturing system, but not about their own mental attitudes or that of other control units (holons).

3.2.9 *Learning*

When designing multi-agent systems, it is often unfeasible to foresee all of the potential situations an agent may encounter, and optimally specify an agent's behavior in advance. In order to overcome these design problems, agents have to learn from and adapt to their environment. Sen and Weiss pointed out, in [72], some ML (machine learning) standard classifications for the different forms of learning. One of them is as follows:

- According to the method or strategy of learning there are: rote learning, learning from instruction and advice taking, learning from example and by practice, learning by analogy, and learning by discovery.
- According to the learning feedback that is available to a learning entity and that indicates the performance level achieved so far: supervised learning, reinforced learning, and unsupervised learning.

Manufacturing control units (holons) must be able to adapt to changing environments and handle emergent contexts. As we pointed out in the openness subsection, holarchies may be reorganized to cope with unforeseen situations. This capability can be improved with learning. For example, as pointed out by [73], some learning targets are: combinations of manufacturing resources for specific tasks, manufacturing system behavior, support in favor of or against a decision, preconditions and postconditions for actions and tasks, types of conflicts, heuristics to solve conflicts and to negotiate, etc.

3.2.10 Benevolence

The property of benevolence is one by which agents cooperate with other agents whenever and wherever possible. Blind benevolence has no place in modeling autonomous agents for whom cooperation will occur only when it is considered advantageous in terms of motivation to do so. The agent cannot spend all its time in new cooperations with other agents, without taking into account its current commitments and motivations. We can say that holons are benevolent entities, because when they discover a possible cooperation scenario they will cooperate.

3.2.11 Mobility

Mobile agents extend the capabilities of distributed systems by code mobility. Mobile agents are programs that can wander through a computer network and contact other agents and agent places to perform their task. The mobile agent paradigm offers a number of advantages: mobility and autonomy make permanent connections unnecessary. The use of agents is also appropriate in scenarios where large volumes of data have to be shipped over the network while the processing code itself is rather small. In such a case, it is worthwhile considering moving the code to the data. Some network management tasks fall into this category. For more information about mobile agents see [74].

Manufacturing control units are dedicated to the continuous control of physical manufacturing components. The relationship $\langle \text{controller}, \text{controlled unit} \rangle$ is assigned beforehand and fixed throughout the manufacturing process. Moreover, all of the necessary control information is in the controlled unit. Therefore, holons rarely need mobility for the execution of their tasks [11].

3.2.12 Recursiveness

The holonic systems' basic condition is that a holon is simultaneously a whole and a part of some whole/part [4]. This means that holons can contain other inferior levels of holons, that can also be contained in another superior level of holons, being in a recursive architecture.

In multi-agent specialized literature, we have found few little references to recursive agent architectures. Nevertheless, in the definition of agents there is nothing to prevent having agents whose internal structure is composed of self-similar entities. This property is analyzed in depth in Sect. 3.3.

3.2.13 Physical and Information Processing Part

In 1994 Christensen proposed a general holon architecture [10] (Fig. 2.1). Almost everyone in HMS adopts this general architecture. An explicit separation between information processing and physical processing can be seen in this general architecture. In the MAS field, there is no such explicit separation, since an agent in its internal structure has software components that can do any kind of processing. Bussmann [11] proposes the use of multi-agent systems as the allowing technology for information processing in HMS. Starting with the holonic vision, multi-agent systems can provide the necessary reasoning techniques to develop the information processing architecture of a holon and the necessary cooperation techniques so that holons can interact with other holons forming holarchies. For the physical processing part, Fletcher and Deen [75] propose functional blocks to manage real-time control for low-level process/machine interaction.

3.3 Recursiveness

One of the most difficult challenges for automated systems is scalability and adaptation. In life systems there are many useful concepts, including examples on how to scale up, evolve, adapt, interoperate, organize, and so on. Complex and adaptive life systems are large, intricate and require active autonomous entities. Koestler called these entities holons. The strength of holonic organization, or holarchy, is that it enables the construction of very complex systems from simpler self-similar entities. Within a holarchy, holons may dynamically create and change hierarchies. Moreover, holons may participate in multiple hierarchies at the same time.

As pointed out by Gasser [76], almost all of the proposals for agent architectures have failed to address the general problem of how to treat collections of “agents” as higher-order entities – e.g., how to treat organizations as agents. In this section, we present an agent definition that deals with organizations of agents. Several difficult challenges for automated systems may be tackled by giving full meaning to the agent concept: adopting a recursive definition of agents and allowing the dynamic creation of agents (organization of agents) by agents themselves.

The need for some kind of hierarchical aggregation in real-world systems has been recognized in the intelligent manufacturing field. These systems have to remain readable while they are expanded on a wide range of temporal and spatial scales. For example, a modern automobile factory incorporates hundreds of thousands of individual mechanisms (each of which can be an agent) in hundreds of machines that are grouped into dozens or more production lines. Engineers can design, build, and operate such complex systems by shifting from the mechanism to the machine or to the production line (depending on the problem at hand) and by recognizing the agents of higher levels as aggregations of lower-level agents. Also, in e-commerce applications, an enterprise is a legal entity that is independent of the individual people who are its employees and directors.

The question arises as to whether an agent can be a collection of several interacting agents, a hierarchy, or some other type of organization. In [76], Gasser pointed out that almost all of the proposals for agent architectures have not addressed the general problem of how to treat collections of agents as higher-order entities – e.g., how to treat organizations as agents.

In the agent-specialized literature, we have found very little work about agent architectures and methodologies that allow us to carry out a recursive and dynamic analysis, design, and implementation of MAS. Most of the current approaches start from an atomic agent definition such as an indivisible entity and build MASs as compositions of interacting agents. Most of the approaches do not deal with systems in which their components may be MASs themselves. The only work we have found about recursive agent model is by Ocelllo. In [77], he proposes a recursive approach to build a hybrid MAS. From a given set of elementary agents, Ocelllo proposes a recursive agent structure definition and two recursive functions, to build a higher-level agent. His work is based on a rigorous analysis of recursive properties in MAS structures, such as agent and environment, and two functions defined in them, interaction and organization. A recursive agent is a MAS, that is, a set of (recursive) agents and (recursive) environment objects. The interaction function allows us to model all of the communication acts that can occur either with other agents or with the environment (perception, action and cognitive interaction). The organization function is modeled as a set of relations between agents. These relations can be of three types: acquaintance, communication and subordination. However, in Ocelllo's work, there is no formal definition of the recurrence property to define the behavior of one level of recursion with regard to another. In our work [60], as in Ocelllo's work, a MAS can be viewed as a set of agents at a given level and as a whole agent at an upper level. We call it an abstract recursive agent. It is abstract because it only exists at the analysis and design phases and is not a real executing agent (at the coding stage it is replaced by its constituent elementary agents). Unlike Ocelllo, we will not build an engine to manage the interaction for a MAS at run time, because this interaction is already managed by its constituent agents. To define the behavior of the abstract recursive agent, instead of using an interaction function (as in Ocelllo's work), we do it with the reactive and intentional behavior of its constituent agents. We believe that these definitions will make the formal definitions of the recurrence property straightforward.

It is important to point out that recursive modeling in the context of our work is different from the work done by Gmytrasiewicz and Durfee [78] and by Tambe [79]. Gmytrasiewicz and Durfee proposed a recursive modeling method as a theoretical framework for representing and using the knowledge that an agent has about its expected payoffs and those of other agents. That is, a representation of the benefits an agent expects to get given the combination of actions chosen by all the agents. On the other hand, Tambe proposed a different approach for an agent's models of other agents' behaviors. He proposed the combination of architectural features that enable an agent to generate flexible and reactive behaviors of other agents. Note that in both of these studies, the recursive model is not a modeling artifact for representing MAS.

3.4 Abstract Agent

The abstract agent is an attempt to unify the concepts of holons and agents and to simplify and close the gap between holons and agents in the analysis and design steps. This makes it easier to translate the modeling products that are obtained from methodologies for HMS into coding elements for the implementation of the holonic system. Thanks to the integration of the holon recursive property into an abstract agent, the abstract agent is useful not only for HMS but for the modeling of complex systems as well. An abstract agent that acts in organizational structures encapsulates the complexity of subsystems (simplifying representation and design) and modularizes its functionality (providing the basis for the integration of pre-existing multi-agent systems and incremental development). The abstract agent facilitates the modeling of organization of organizations as well as, multi-agent systems of multi-agent systems.

Definition 3.1. An abstract agent [60] is a software system with a unique entity, which is located in an environment, which as a whole, perceives its environment (environment sensitive inputs). From these perceptions, it determines and executes actions in an autonomous and flexible way – reactive and proactive. These actions allow the abstract agent to reach its goals and change its environment. From a structural point of view, an abstract agent can be an agent (atomic entity); or it can be a multi-agent system (with a unique entity) made up of abstract agents that are not necessarily homogeneous.

An abstract agent is on a higher conceptual abstraction level than an agent. An abstract agent can be seen as a MAS, an organization, a federation or an institution with the added value that it can also be a composition of all of these abstraction models. Furthermore, when we define two interacting abstract agents, we could also be modeling two interacting organizations, federations, MASs or institutions. An abstract agent will exist only at modeling stages. In the end (at coding stages) it may be replaced by a group of agents or also by a single agent.

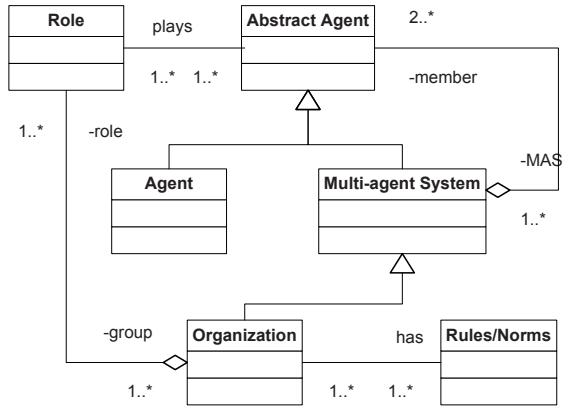
Definition 3.1 provides a functional and structural abstract agent perspective. The functional perspective is based on the widely known agent definition of [5], in which an agent is an autonomous, reactive and proactive entity. On the other hand, the structural perspective introduces an indirect recursion when indicating that an abstract agent may be a MAS, which in turn is made up of Abstract Agents, each one of which may be a MAS or an agent.

Definition 3.2. A multi-agent system is made up of two or more abstract agents that interact to solve problems that are beyond the individual capabilities and individual knowledge of each abstract agent.

Definition 3.2 extends the traditional notion of multi-agent systems when indicating that a MAS is made up of abstract agents. This could be a very useful property because with this we could have a MAS made up of interacting MASs.

Figure 3.2 illustrates the essential abstract agent structure.

Fig. 3.2 Abstract agent



There are two levels in an abstract agent. The abstraction level and the recursion level. The abstraction level is used in the analysis and design phases. When we begin to analyze a MAS A we identify the group of agents $\{a_1, a_2, \dots, a_{n'}\}$. Agents $\{a_1, a_2, \dots, a_{n'}\}$ are said to be on a lower abstraction level than A . Let m be the abstraction level of A , then $\{a_1, a_2, \dots, a_{n'}\}$ are in $m - 1$ abstraction level. Subsequent analysis will "open" these agents, for example when analyzing a_1 , we could have that $a_1 = \{a_{11}, a_{12}, a_{13}\}$. Then the abstraction level of each agent in $\{a_{11}, a_{12}, a_{13}\}$ will be $m - 1$, and so on. The recursion level is defined as follows:

Definition 3.3. Let a be an agent and A and A_i be abstract agents. The recursion level of an abstract agent is:

$$LevelR(A) = \begin{cases} 0 & A = a, \\ \max\{LevelR(A_i)\} + 1 & A = \{A_1, A_2, \dots, A_k\}, 1 \leq i \leq k. \end{cases}$$

From definition 3.3 we have:

- Abstract agent of recursion level 0 is an agent.
- Abstract agent of recursion level 1 is a MAS made up of interacting agents.
- Abstract agent of recursion level $n > 1$ is a MAS made up of interacting abstract agent of recursion level $< n$.

The designer's point of view will determine the nature of what is being observed at each moment. From the outside, a system can be considered as an abstract agent since it has agenthood characteristics. On the other hand, from the inside, that is, from the internal structure, the abstract agent can be considered as being composed of a group of interrelated abstract agents (MAS). When there are no more subdivisions, the abstract agent can be considered as being a simple agent. The end of the recursion is defined by the designer since the subdivision exists whenever it is useful for the definition of the problem being modeled. In the end, at the lowest abstraction level, only the agents that make up the global MAS will be apparent, but

as the abstraction levels go up, there will be some agents and some abstract agents that are refined as MASs.

An extremely useful feature in terms of the reduction of complexity for the designer of a MAS is that an overall task can be broken down into a variety of specific subtasks, each of which can be solved by a specific agentified problem solver. Divide and conquer is a widely accepted problem solving paradigm in computer science. In this study we attempt to define a modeling structure to apply the divide and conquer paradigm to the analysis of Multi-agent Systems.

3.4.1 Abstract-agent Structure

In [60] we proposed an abstract agent as a modeling artifact for the analysis of large-scale multi-agent systems. With this approach, to develop multi-agent systems as systems in which their components may be MASs themselves, the idea is as follows [80]. When we begin to analyze a group of agents (MAS) A , we identify the agents $\{a_1, a_2, \dots, a_n\}$ that execute certain functions. These agents may encapsulate individual persons, physical, or software entities (agents). They may also be other groups of MAS, say B , so we can have $a_i = B_i$, which we treat as black boxes. We can take this perspective as long as our analysis can ignore the internal structure of the member groups (MAS). However, subsequent analysis generally needs to “open” these black boxes and look inside them to see the agent components and their corresponding functions; for example, when analyzing B we have that $B = \{b_1, b_2, \dots, b_m\}$.

The following example illustrates how the abstract-agent structure is used for multi-agent system modeling.

Let us suppose a multinational company, called AG, which has different national companies distributed among different countries. The objective is to model the multinational as a MAS.

Each national company can be an abstract agent since it has agenthood characteristics. The national company is autonomous in its national environment; it acts in the national market with its own market and production rules. At the same time, it must be able to interact with other national companies to exchange materials, personnel, knowledge, etc. The national company, is also governed by the rules and norms of the multinational for its international relations (other national companies).

The international companies’ relationships define the rules, norms and policies of the multinational. In Fig. 3.3, geographical areas can be observed in which the relationships among the national companies are narrower. In addition, the commercial agreements among the different countries define new interrelation rules among the national companies of these zones. For example, in Europe, the European Union countries are governed by certain standards and norms of the community; and in South America they are governed by the Southern Cone Common Market – MERCOSUR (Paraguay, Argentina, Chile, Brazil, Uruguay and Bolivia) and by the Andean Community (Bolivia, Colombia, Ecuador, Peru and Venezuela). The relation-

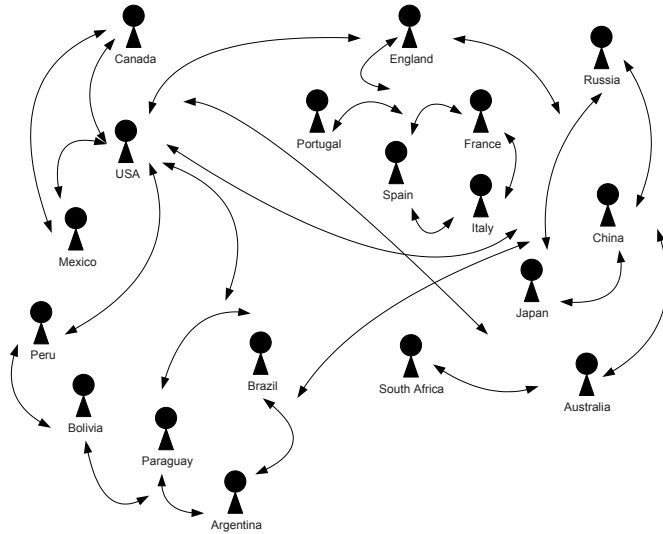


Fig. 3.3 National companies of the multinational AG

ships of the countries of these markets with other countries or regional markets are managed by their local market rules. Each market can be modeled as an abstract agent. This generalization is shown in Fig. 3.4. It is important to note that Bolivia, as a National Company, belongs to two regional companies (MERCOSUR and the Andean Community).

Up to this point, we have identified 4 levels of abstraction (Fig. 3.5): the multinational company, the regional companies, and the national companies. We have been able to model the multinational as a MAS, which is composed of abstract agents that

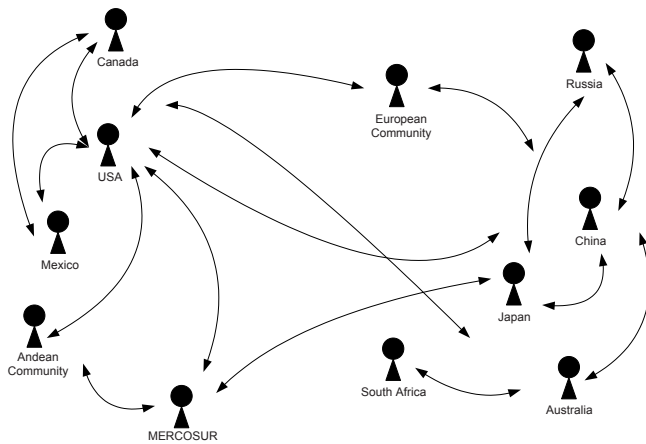
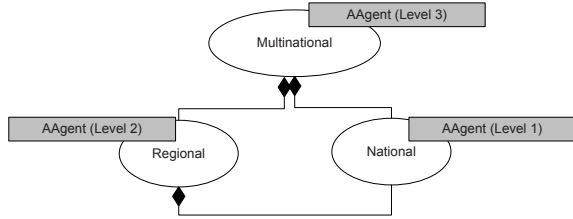


Fig. 3.4 National and regional companies of the multinational AG

Fig. 3.5 Three recursion levels of the AG example



are related to each other with certain behavior patterns that define the multinational company. If the national companies are made up of agents (abstract agent of recursion level 0), we can think of a national company as a traditional MAS (abstract agent of recursion level 1), the regional companies as abstract agents of recursion level 2 and the multinational as an abstract agent of recursion level 3.

Apart from modeling the outside relationships, if the designer’s interest is also to model the internal structure of each national company, the national company should be observed from the inside. Inside each national company there would be new companies located in different cities or with autonomy for certain activities. In turn, each local company is subordinated to the national company and each national one to the multinational. Thus, we have a new level of abstraction, the local company as an abstract agent of recursion level 1, the national company as an abstract agent of recursion level 2, the regional company as an abstract agent of recursion level 3 and the multinational as an abstract agent of recursion level 4 Fig. 3.6.

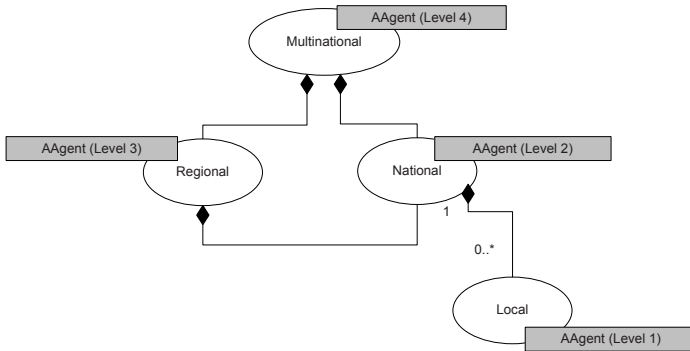


Fig. 3.6 Four recursion levels of the AG example

If the national company is not subdivided into city companies or autonomous companies, then the national company is a traditional MAS composed of national domain-specific agents (abstract agents of recursion level 0), which are interrelated agents and carry out specific functions. These national domain-specific agents define the services provided by the national company inside the country and outside the country. This very same analysis should be made for each Local company until we reach the agents, which define and implement the activities of the company as a

whole. In summary, the final result of the analysis should be similar to Fig. 3.7. In Fig. 3.7, it can be observed that the national company is composed of zero or more local companies, and each local company, in turn, is an abstract agent of recursion level 1.

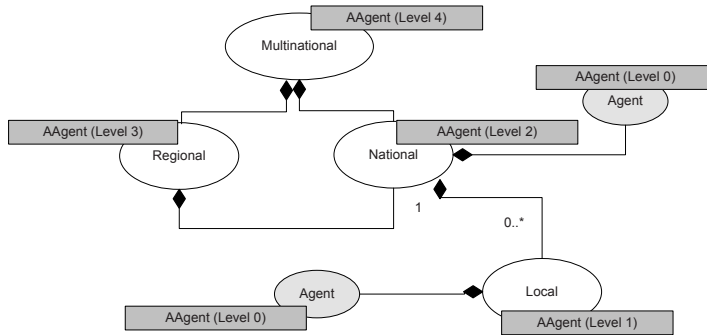


Fig. 3.7 Five recursion levels of the AG example

Again, the multinational can be considered from the outside as an abstract agent, since it is located in an environment – the world market; it is autonomous; it has its own economic and market policies; it is social, i.e., it interacts with other entities for purchasing, selling, recruiting, leasing, etc.; it is proactive, since, for example, according to world market trends it is able to modify its current market policies, etc.

An abstract agent is specially tailored for: specification of interaction between groups of agents, integration of pre-existing multi-agent systems and modeling of holonic manufacturing systems [4], among others.

An abstract agent is a kind of virtual entity because it is not executable (it will not exist at the coding stage, it will possibly be replaced by a group of agents or also by a single agent). On the other hand, an abstract agent has agency properties, it is autonomous, reactive and proactive [5]. An abstract agent may play roles (see Fig. 3.2). A role is a description of an agent's abstract behavior. A role describes the constraints (obligations, requirements, skills) that an agent will have to satisfy to obtain a role, the benefits that an agent will receive in playing that role, and the responsibilities associated to that role.

An agent is a special case of an abstract agent (it is executable). A multi-agent system is also an abstract agent, but it is a composite entity. A multi-agent system is made up of two or more abstract agents that interact to solve problems that are beyond the individual capabilities and individual knowledge of each abstract agent.

A multi-agent system is a group of agents that are related via interaction patterns. Groups of agents may be explicitly specified by the system designer or they may be emergent. When a group of agents has norms and rules that govern the interaction among its members we call it an organization (see Fig. 3.2). An organization is a special case of a multi-agent system in which there is a set of norms and rules that regulate the interaction among the roles (played by agents) of the organization. The

structural aspect of an organization is made up of two parts: a group of abstract agents that make up the organization, and a set of roles assigned to the organization and their relationships. The rules and norms of the organization are defined by a set of institutionalized patterns of interactions that are defined between the roles assigned to the organization.

An abstract agent acting in organizational structures can encapsulate the complexity of subsystems (simplifying representation and design) and modularize its functionality (providing the basis for integration of pre-existing multi-agent systems and incremental development). The abstract agent structure facilitates the modeling of organization of organizations (as well as, multi-agent systems of multi-agent systems). This is so because an organization is an abstract agent and therefore may play roles in a wider organization, interact with other abstract agents (be it an agent or an organization as well), pursue some goal, etc.

In [60], we proposed a formalization of MAS behaviors in terms of their constituent agent behavior. In summary, the reactive behavior of a MAS is determined by its perception that is defined as the union of the set of perceptions of its agents. It is also defined by its actions, which in turn are defined as the union of the group actions executed by its member agents and the union set of the primitive actions carried out by each of its constituent agents. The intentional behavior of a MAS, considering a BDI agent architecture, is determined by its goals, desires and intentions. The goals of a MAS can be defined using two different approaches, depending on the problem at hand. The top-down approach is defined for situations where the set of goals of the MAS is given and the objective is to determine which group of agents could reach it. On the other hand, the bottom-up approach is defined for situations where there is a group of interacting agents and the objective is to find out what the goals of the emergent MAS are.

3.5 Conclusion

In this chapter, we have analyzed the characteristics of holons and agents. We have also presented an abstract recursive agent definition (abstract agent). This definition provides a functional and structural abstract agent perspective. The functional perspective is the well-known agent definition of [5], in which an agent is an autonomous, reactive and proactive entity. On the other hand, the structural perspective introduces an indirect recursion when indicating that an abstract agent may be a MAS, which is at the same time made up of social abstract agents, each one of which in turn may be a MAS or an agent. This definition allows us to carry out a dynamic and recursive analysis and design of a multi-agent system. In Part II the abstract agent notion is used to define the ANEMONA methodology and in Part III its practical usage is explained.

Part II
Methodology for Holonic Manufacturing
System

Chapter 4

HMS Development

In this chapter we will discuss state-of-the-art techniques and methods for modeling manufacturing systems. Firstly, we will summarize the modeling requirements of the new generation of manufacturing systems. Then we will discuss the studies carried out in the areas of HMS, MAS and enterprise modeling, and conclude with a comparative overview of the different approaches.

4.1 Modeling Requirements

A manufacturing system is a complex, global system that encompasses the entire set of activities in a manufacturing company. The manufacturing system has a set of characteristics imposed on it by the “new manufacturing” approach (Chap. 2). From these characteristics a series of requirements arise, which we can cluster into two large groups: functional requirements and software engineering requirements.

4.1.1 *Functional Requirements*

Manufacturing control systems are large-scale complex systems designed to carry out a clearly defined task in a standardized and well-structured environment. Although manufacturing processes undergo several changes and disturbances, the levels of uncertainty and unpredictability are not comparable to spatial, traffic, or service application systems.

The modules, or entities (holons), which implement the control of these systems should cooperate in order to achieve the global manufacturing objectives. With regard to these objectives, a holon never rejects the cooperation of another holon deliberately. It only rejects their execution, when the actions requested are impossible or strongly unfavorable for the manufacturing process. In this sense, manufacturing holons are semi-autonomous. This characteristic is defined implicitly in the way that

holons are organized to cooperate. A holarchy is a “loosely coupled” and temporal hierarchical structure in which there is a certain level of subordination to the holon that manages it (for example, the staff holon of the PROSA architecture, or the coordinator holon of the cooperation domains, see Sect. 2.3.3). At the same time each holon is an autonomous entity that acts in cooperation interactions. These issues impose the following functional requirement.

Requirement 1: Manufacturing control systems require autonomous entities to be organized in hierarchical and heterarchical structures.

The second requirement is related to the kind of behavior that the control unit at the factory level should exhibit. Manufacturing control units are continually handling a high number of repeated events that are known, but unpredictable. This flow of events should be handled in an effective way and with temporal constraints. The handling of the events can consequently be fixed *a priori* by routines, while the beginning and execution of these routines should be performed in real time. The set of events and their occurrence patterns change over time.

Requirement 2: Manufacturing control units require routine-based behavior that is both effective and timely [81].

4.1.2 Software Engineering Requirements

Besides functional requirements, any control system (which is used in a manufacturing environment) should satisfy general industrial standards. These standards specify, among other things, requirements for reliability, fault tolerance, diagnosis, and maintenance. The control systems should achieve certain reliability levels that guarantee a continuous operation. This is also true for the control software. However, product dependability is only achieved if the software development process is carried out following an engineering methodology, instead of developing it in an *ad-hoc* way with no engineering methods or techniques.

From fundamental software engineering principles the following requirements are derived:

Requirement 3: Programming methods should provide data and process encapsulation.

Requirement 4: Control programs should have clear semantics.

Specialized literature in the field of intelligent manufacturing basically takes two approaches to problem decomposition. (i) Physical decomposition (the most obvious): agents are used to represent the entities of the physical world, such as workers, machines, tools, schedules, products, orders, attributes and operations. This approach defines different sets of state variables that should be handled by the agents in an efficient way and with a limited number of interactions. However, with this approach a great number of agents per resource are required. A common example

of this approach is the use of order agents and machine agents for the planning and scheduling of manufacturing [48, 82, 83]. (ii) In the functional decomposition approach the agents are used for encapsulating functionalities such as work order acquisition, planning, scheduling, material handling, logistics, etc. In this approach the agents do not have an explicit relationship with the physical entities. Examples of this approach include the use of agents for encapsulating special functionalities (e.g., facilitator agents [84], broker agents [85], mediator agents [86]) and the use of agents for integrating pre-existent systems (e.g., ARCHON [87], EXPORT [88] and CIIMPLEX [85]). From these approaches the following requirement is derived.

Requirement 5: A methodology for HMS should lead to straightforward translation from the control task on a factory resource or factory function to autonomous entities [4].

In the field of intelligent manufacturing a kind of “loosely” hierarchical aggregation for real-world systems has been recognized. These systems have to remain readable while they are expanded into a wide range of temporal and spatial scales. For example, a modern automobile factory, incorporates hundreds of thousands of individual mechanisms (each of which can be an agent) into hundreds of machines that are grouped into dozens or more production lines. Engineers can design, build, and operate such complex systems by shifting from the mechanism, to the machine or to the production line (depending on the problem at hand) and by recognizing the higher-level agents as aggregations of lower-level agents. This implies the following requirement:

Requirement 6: A methodology for HMS should define a development process that is guided by abstraction levels, and should also provide modeling artifacts, tools and guidelines to manage this process

The traditional methods and techniques for manufacturing system modeling, such as CIM, are mainly based on a top-down approach. The user’s requirements and the global conceptual design constitute the whole set of modeling constraints. With these approaches very rigid hierarchical architectures are built [4]. On the other hand, HMS modeling requires a mixed development process, bottom-up and top-down depending on the level being modeled. It is not necessary to define the whole set of constraints at the beginning. A mixed development process allows the generation of reconfigurable and scalable architectures. This characteristic implies the following requirement.

Requirement 7: A methodology for HMS should define a mixed top-down and bottom-up development process.

Finally, the following requirement is inferred from the characteristics of “new manufacturing” defined by the HMS consortium.

Requirement 8: A methodology for HMS should integrate the entire range of manufacturing activities (from order booking through design, production, and marketing) to model the agile manufacturing enterprise [4].

Taking these requirements into account we will now analyze the different methodologies reported in: (i) the HMS field; (ii) the MAS field (due to the widespread

tendency to use MAS technology as the implementation tool for HMS, see Chap. 2), and finally; (iii) the enterprise modeling field, because a manufacturing system is embedded in a manufacturing company. The goal of this study is to determine if these methodologies are suitable for modeling HMS. Firstly, we present a brief summary of the different methodologies (more details can be found in specialized literature). Finally, we will make a comparative summary based on the requirements we have cited in this section.

4.2 Holonic Manufacturing System Methodologies

In HMS-specialized literature there are few studies on HMS development methods. The reason may be that research efforts have been centered around the development of holonic control systems and in the definition of architectures (Chap. 2). There is a recognized need, however, for design methodologies that provide clear, specific and unambiguous development processes and guidelines [3].

Leitão and Restivo in [57], propose *a formal approach for holonic control system specification*. This proposal attempts to formalize the structure and behavior of a HMS. It combines UML notation to specify the structure and static aspects of the system, and Petri nets to model behavioral aspects. The methodology is based on the ADACOR architecture [25].

The static structure of the system is defined by the specification of the system object classes, attributes, methods and relations set. This structure is similar to conceptual models (diagram of classes) in object-oriented methodologies.

The modeling of the system is based on the development of a behavioral model for each of the holon classes.

- Each product is represented by a product holon that is responsible for process planning and contains all the knowledge related to the product. The behavior is modeled by means of a *product holon model*. This model consists of a Petri net that specifies the different states of the product holon and the preconditions and postconditions of each status. It is also used for synchronizing different holons with each other.
- Each production order is represented by a task holon, which is responsible for the control and supervision of production order execution and contains the dynamic information. The behavior of a task holon is modeled by means of a *task holon model*. The Petri net specifies those functions for order decomposition, the planning of resource assignment and the execution of these plans.
- The operational holons represent the manufacturing physical resources, such as workers, robots and machinery. The behavior is modeled with the *operational holon model*. This holon acts as a reactive server that waits for new operations (proposed by the supervisor holon or by the task holons).
- The *supervisor holon model* specifies the behavior of the supervisor holon in charge of coordinating the activity of the holons under its control. It provides

coordination and global optimization and it can coordinate several operational and supervisors holons.

Leitão and Restivo's proposal is very preliminary and is lacking in several ways. For example, it does not describe how to specify the relationships among the system holons, nor what the relationship is among the static structure of the system and the different holon behavior models. The development steps are not defined. It does not specify how each holon model is translated into an entity of an implementation platform. Finally, the most critical aspect is that using this approach, neither holon cooperation, autonomy nor flexibility are modeled, these being the basic holon characteristics as defined by the HMS consortium.

Another proposal is given in [58]. Here, an agent organization is proposed for modeling each holon/holarchy. The holon characteristics are specified as agent organization characteristics that model them. No holon architecture is used. It proposes different types of organization that can be used to model autonomous and self-interested entities (heterarchy systems), and centralized hierarchical control systems, and the holarchy structures where the semi-autonomous entities are managed by a "head" of the organization. The approach of Fisher et al. also lacks different development steps, and is only focused on the definition of the holarchies (organizations) and their different configurations. In a previous study [89], a design based on the architecture INTERRAP (Chap. 2) is proposed. In this study an FMS is modeled using layers. Five layers are identified: manufacturing control and planning, shop-floor control, cell control, autonomous system control, and machine control. A hierarchical structure between layers and a horizontal cooperative structure among the components of the same layer are defined. There is no development process defined and there is no modeling notation. Nevertheless, it is the only study that considers the global manufacturing system integrated in the company.

4.3 Multi-agent System Methods

In this section we present a summary of the better-known MAS methodologies in the agent field. We classify them into two big clusters: general-purpose MAS methods¹ and MAS methods for manufacturing systems.

4.3.1 General-purpose MAS Methods

In this section we summarize some of the better-known MAS methodologies.

¹ In this cluster of general-purpose methodologies we can find different clusters, for example, role-based methods, agent-based methods, organization-based methods, system-based methods, interaction-based methods, behavior-based methods, extensions of knowledge-based methodologies, etc. This study is beyond the scope of this book.

Modeling and design of BDI multi-agent systems. This method [90] extends the OO modeling techniques to agent systems based on BDI architecture [91]. It proposes two levels of abstraction: external and internal. In the external view, the system is modeled as a hierarchy of agent classes (agent model). The classes of agents are characterized by their purpose, their responsibilities, the services that they offer, the information about the world that they require and store, and their external interactions (interaction model). From the internal point of view a set of models is used (belief model, goal model and plan model) that allow the specification of the motivation and information state of the agents, as well as the control structures that determine their behaviors. It is based on the identification of the key role in the application and their interrelations, which serve as guidelines for the definition of the agent class hierarchy. The analysis of the responsibilities of each role, represented by an agent class, results in the identification of the services that each agent provides and uses, external interactions and objectives and events that it should respond to.

In [92], Burmeister presents a method for MAS development in which three models are specified for the agent-based analysis and some construction guidelines are defined. The method is based on OO techniques. It divides the analysis into three sub-models: (i) Agent model, which contains the agents and their internal structure (mental states). (ii) Organization model, which specifies relationships, mainly the hierarchical ones, among the agents and the agent types. (iii) Cooperation model, which describes the cooperation among agents. The conjunction of these three models completely defines the system to be developed. The design phase consists of the refinement of these models for the subsequent implementation of the system by means of a specific tool.

The MAS-CommonKADS methodology [93] is based on CommonKADS [94], and provides a set of models for developing the MAS analysis and design phases. Its main characteristic is the incorporation of object-oriented techniques to CommonKADS. In MAS-CommonKADS different views of the system are proposed and these views are translated into different phases. In the conceptualization phase a user-centered analysis is proposed for the requirement elicitation (use cases are used). The organization model analyzes the company (organization) in which the system will be implemented, and is useful for describing the relationships among the agents in the MAS and the relationships of the agents with their environment. The agent model describes the properties and characteristics of each agent. It is one of the most important phases, consisting of an agent identification sub-phase (with different options) and a detailed description sub-phase of each agent. The task model describes the functions or tasks (cognitive, man-machine communication, agent communication) that the system provides. In the experience model the way in which the cognitive tasks are carried out is described. The communication model is focused on describing how the man-machine interaction tasks are carried out. The coordination model develops and describes the interactions among agents in the multi-agent system. In the design model a design of the network, the agents (agent's architecture) and the platform (operating system and hardware) is developed.

GAIA [95] is focused on the idea that the construction of agent-based systems is a process of organizational design. An organization in GAIA is a collection of

roles, which maintain certain relationships with other roles and participate in institutionalized patterns of interaction with other roles. The roles contain four aspects: responsibilities of the agent, the resources that it is allowed to use, the associated tasks, and the interactions. GAIA intends to work initially with a high-level analysis. In this analysis two models are used: the role model for identifying the key roles in the system together with their properties, and the interaction model that defines the interactions by means of a reference to a model of message exchange. The following step is the high-level design, whose objective is to generate three models: the agent model, which defines the existing agent types, how many instances of each agent type exist and what roles each agent plays; the service model, which identifies the services (agent functions) associated to each role; the acquaintance model, which defines the communication links that exist among the agents. In the following steps, classic OO design techniques would be applied. However, this is outside of the scope of GAIA.

GAIA v.2 [96] extends GAIA, taking into account the fact that an organization is more than a single collection of roles, which is how it was considered in the first version of GAIA, and therefore it introduces new organizational abstractions. Besides the role and protocols, the environment in which the MAS is immersed constitutes the main analysis and design abstraction. This new version incorporates rules and organizational structures as necessary elements for the definition of the organization. This extensions attempt to adapt the original version of GAIA for the design and construction of open systems.

ROADMAP [97] began as an attempt to extend the original version of GAIA with: a dynamic hierarchy of roles (one way to handle open systems), additional models for explicitly describing the agent environment (just as GAIA v.2 does). Besides the basic models, ROADMAP inherits from GAIA the organizational view of the MAS, and the basic definitions of roles, protocols, agents and services. In ROADMAP, a system is seen as an agent organization, and it consists of a hierarchy of roles and a hierarchy of agents. The hierarchy of roles is the specification of the system, and it represents the correct behavior of agents. The hierarchy of agents is the implementation of the system, and it provides their current functionality. The environment model and the knowledge model contain reusable domain information. The use case model, the interaction model, the role model, the agent model and the acquaintance model are specific to the application. The protocol model and the service model describe the low-level software components that are potentially reusable.

The multi-agent system development method, MASSIVE (multi-agent systems iterative view engineering), developed in the DFKI [98] is made up of a set of different views of the system, where an iterative development process is followed. In this method a re-engineering process together with an improved cascade method, which allows refinements to be carried out, are combined. The different views are: tasks, environment, roles, interactions, society, architecture and system. In the task view the functional aspects of the system are analyzed. A task hierarchy is generated, and this hierarchy is used to determine the basic problem-solving capabilities of the final system entities. The environment view basically consists of carrying out

an analysis of the system from two points of view: from the developer's point of view and from the system's point of view. The role view begins applying the multi-agent paradigm as a problem solution. This view consists of determining those role abstractions necessary for covering with the functionality and physical restrictions of problem specification. The question of how to assign roles to agents is also considered. In the interactions view the system interactions are modeled. In MASSIVE the system interactions are considered a form of general conflict resolution and they are not limited to a particular form, as can be the case with the communication. In the society view the goal is to classify the society (set of agents), which is desirable from the point of view of the developer. In the architecture view the specification of the rest of the views is transformed into one system architecture where the structural attributes of the system are defined. The system view copes with those system aspects that affect the rest of views or the whole system. Examples of the aspects handled in this view are the design of user interfaces, the strategy of error handling or the actual system setup.

In Tropos [99] an agent-based software development methodology is presented. This methodology uses extensions of UML and a modeling environment called *i** [100]. The first step of the methodology is to elicit the requirements by means of the modeling of objectives, tasks, resources and system actors, as well as the dependencies among the actors. During the design stage, the requirement model is elaborated by means of (i) the system actors refinement, (ii) the identification of the necessary capabilities for satisfying the system objectives, and (iii) the assignment of these capabilities to agents. In the first design step, the actor diagram of the system can be enlarged with the help of design patterns, although the way in which the appropriate patterns are identified are not identified. For the other design steps, the methodology does not include guidelines for translating the elements identified in the previous models into entities that can be implemented in an agent platform.

MaSE (multi-agent system engineering) is a methodology developed at the Ohio Air Force Institute of Technology by Wood and DeLoach [101]. MaSE uses a specification language that is based on UML+OCL [102] and a development tool called AgentTool. The systems designed with this methodology should be closed systems. MaSE only permits the building of systems with a maximum of ten agent classes. The methodology does not support mobility characteristics. Dynamic systems, where the agents can be created and destroyed, are not allowed. The conversations among agents are "one to one" and multicast is not allowed. From a general point of view MaSE is divided into two big steps: analysis and design. The analysis is divided into three phases: objective capture, role transformation and use-case application. The design is divided into four phases: agent-class creation, conversation construction, agent-class assembling and system design. The first MaSE step is to identify the objectives and use cases. Later, the objectives form the roles that incorporate the tasks for obtaining the system goals. The use cases become sequence diagrams in order to take into account the sequences of events to be designed. Next, the roles are integrated into agent classes connected by means of conversations, which are specified using specific diagrams. Finally, an agent diagram is obtained from the agent classes and from this diagram the system can be generated automatically.

MESSAGE [103, 104] (methodology for engineering systems of software agents) is an agent-based methodology that incorporates software engineering techniques that cover the analysis and design phases of multi-agent systems. The methodology provides a language, a method and some guidelines for applying the methodology, and is focused on the analysis and design phases, plus it outlines some ideas about the rest of the steps such as implementation, tests and installation. The notation used is UML and the modeling process is RUP (rational unified process) [105]. Its development process consists of 4 phases: beginning, elaboration, construction and transition. For developing a MAS, MESSAGE offers some guidelines to determine if it is appropriate to describe the entities (or at least some of them) of the proposed system as agents [106]. Requirement step: according to MESSAGE, the characteristics of the initial requirements for MAS are no different from other systems. In the analysis step: an agent-analysis model is provided, which introduces agent abstraction as the one of the fundamental construction blocks. Other elements of the analysis are: tasks, objectives, roles and interactions. In this step, models are built of: organization, goals and tasks, agent, interaction and domain model [107]. In the design step, the design model provides constructors for describing the communication among agents and between an agent and their environment. It also allows the definition of the internal agent structure. For the implementation step, guidelines for the selection of agent platforms that support the developed system, have been provided.

RT-MESSAGE [108] proposes an extension of MESSAGE for the modeling of real-time MAS. It is based on the real-time MAS platform SIMBA [109]. The main component of SIMBA is the real-time agent architecture ARTIS [110]. RT-MESSAGE defines the analysis, design and implementation steps. In the analysis step the MESSAGE models are used, but these models are extended to be able to model real-time system characteristics. In the design step all of the artifacts generated in the analysis are the inputs, and this step is focused on transforming the entities specified in these artifacts to computational entities. To do this, SIMBA architecture is used. SIMBA allows the design of the interactions, organizations and internal structures of the system's real-time agents.

INGENIAS [111] is presented as an evolution of MESSAGE. INGENIAS goes deeper into the elements of MESSAGE for the specification and development process, and it incorporates new support tools and examples of developments. INGENIAS, as MESSAGE, defines a set of metamodels (high-level description of the elements in a model) for describing the system. The metamodels allow the description of: isolated agents, agent organizations, the environment, interactions among agents or roles, tasks and objectives. The process of metamodel instantiation (creation of models) is not trivial. There are many entities and relationships to identify, besides dependencies among different models. To this end, INGENIAS defines a set of activities whose completion results in a set of models. These activities, in turn, are organized following a software engineering paradigm, RUP (rational unified process) [105]. The execution of the activities for building models is based on the INGENIAS IDE tool, a tool for visual modeling. This tool stores the specifica-

tion of the system using XML. From this specification the IDE tool generates code and documentation.

MASB is a methodology proposed in [112]. It identifies agent (human or artificial) roles in an application by means of the analysis of scenarios in which the agents interact with a potential user. In the analysis step, the user describes (in text) a typical scenario emphasizing the roles carried out for human and artificial agents, the typical exchanges of messages, the events that happen during the execution of the scenario and the actions carried out by the agents. A role is characterized by means of a behavior diagram specifying the activities, knowledge and interactions in which the role participates. The analysis step concludes with the modeling of the local data, the static and dynamic description of the world, and interactions between the system and the users. In the design step, the agents are identified and the roles are assigned to these agents. Once the agents have been identified, the designer specifies the knowledge structures that characterize each agent (beliefs, decisions, actions and reasoning) for carrying out a role. Finally, the conversations among agents are specified taking into account the plans that an agent is able to execute.

The Prometheus method [113] begins with the analysis step in which the goals and basic functionalities of the system to be developed are identified. To identify goals, functionalities, as well as the interactions among those functionalities, the methodology proposes the analysis of typical use cases. Once the functionalities have been identified, they are grouped and assigned to agents according to coherence and coupling approaches of traditional software engineering. In particular, it provides the following strategies for grouping functionalities: (i) to group, if the functionalities use the same data or require the same information; (ii) to group, if this implies fewer interaction links among the agents; (iii) not to group, if the functionalities are not related or if they should reside in different hardware platforms; (iv) not to group, if the data of a functionality should not be available for another functionality due to security or privacy reasons; and (v) not to group, if the functionalities will change, or they will be modified by different people. To apply these strategies, Prometheus provides a coupling diagram of data and a diagram of “acquaintance”. The clustering process is based only on the coupling of data and interaction.

Elammari and Lalonde proposed a method that goes from specifications of high level to implementable model through discovery and definition phases [114]. It generates five models: (i) the high-level model identifies agents and their high-level behaviors; (ii) the internal agent model describes the internal structure and the agents’ behavior; (iii) the model of relationships captures the dependencies and the jurisdictional relationships; (iv) the conversational model describes the coordination among agents; and finally (v) the contract model defines a structure of agreements among the agents. The discovery phase develops the high-level model with the help of maps from use cases [115]. Using these use-case maps, the agents are identified taking into account the actors that play active roles in the application. It also identifies the roles and responsibilities. The definition phase produces the four remaining models.

Miles et al. proposed a design method, called “analysis of interaction among agents” [116]. This method derives the required interactions from an analysis of

goals and preferences of the system requirements. The methodology begins with the identification of the system goals and it converts these goals into a hierarchy of goals. It is assumed that each goal is solved by means of the interaction of some agents. In a second step, the designer chooses a mechanism of interaction for each independent goal of the hierarchy of goals using roles to refer to the agents that are able to participate in the interaction. During the execution of the system the goal is taken on by a “real” agent, which takes charge of looking for other agents that play the other roles in the interactions. Should the agent fail in the search, it can use the hierarchy of goals to decompose the goal into sub-goals and then propose the initiation of the interactions associated to them. The agents that are needed in execution time are derived from the roles required in the interactions. These roles are grouped taking into account the preferences identified in the definition of the problem, as well as general considerations, such as complexity or the agents’ limited functionality, or optimization of the computational load of an individual agent.

Collinot et al. proposed Cassiopeia [117] as a design method to derive the individual behavior of agents from the global specification of a collective task. Cassiopeia focuses on the organization of the MAS for connecting individual behavior to group behavior. It distinguishes three behavior levels: (i) elementary, (ii) relational, and (iii) organizational. Cassiopeia begins with the elementary behaviors and builds the system defining more complex behaviors. Each level is designed in a different step: a) Identification of the elementary behaviors. The elementary behaviors are those behaviors that are required to be able to execute a collective task. b) Specification of the relationships among behaviors. Analyzing the organizational structure with regards to the dependencies among the elementary behaviors. c) Specification of the organizational behavior. The organizational behaviors are those that allow the agents to handle the formation, duration or breakup of the groups.

The environment agent societies [118] proposes an organization-based method for developing MAS. It describes: (i) the structure and the global characteristics of a domain from an organizational perspective (organization model); (ii) it defines the population of agents by means of social contracts that regulate the execution of roles by individual agents (social model); and (iii) the agent interactions by means of interaction contracts (interaction model).

The civil agent societies method [119] is based on human civil societies, in which the social institutions settle and impose the laws, and monitor the fulfillment of these laws in order to respond to possible emergencies. It is based on: (i) socialization services, in which social contracts between agents and society are settled, indicating that the agents belongs to the society; (ii) a notary’s office service, that verifies that the interactions among agents are legal and to generate an appropriate private contract; and (iii) the exception handling service which initializes sentry agents when new contracts are generated. These sentry agents try to avoid exceptions and detect their symptoms.

4.3.2 MAS Methods for Manufacturing Systems

In this section we summarize some studies about MAS methods for manufacturing systems that have been reported on in specialized literature.

Kendall et al. [120] proposed a methodology for the development of an agent-based system that is built on the base of object-oriented methodologies, just like OMT and OOSE [121], and the IDEF method [122] for modeling manufacturing systems (see Sect. 4.4). The method begins by creating an OO model and an IDEF model (IDEF0) of the system that is being modeled, and then identifies the agents and the interactions among these two models. The agents are seen as the autonomous decision makers and they are identified in the OO model when an actor appears, and in the IDEF model when a function creates control information as output. The agents identified are modeled as BDI agents, in particular as procedural reasoning systems [123]. The interactions are identified when two or more resources of IDEF appear in an information exchange. The interaction patterns are defined from the corresponding use cases [121]. The definition of the agents and the interactions are completed with the help of OO design techniques. In spite of being defined as a method for the development of agent-based systems, this proposal is limited by its own base (OO methods and IDEF0 method) since it bases the whole identification process and definition of agents on techniques that do not explicitly model the autonomous behavior, nor decision making. Also, the object-oriented concepts present several limitations when agents are identified and defined [124].

Ritter et al. [125] proposed a method for the identification of agents in a manufacturing system. The process is based on the physical aggregation or logical dependence of manufacturing objects (obtained from an inventory list). However, the method does not provide precise criteria on how to define those physical and logical dependencies (it only presents an example). Also, the method does not include any guidelines for specifying which manufacturing objects/entities to apply the aggregation process. The agent-identification process therefore is quite subjective and intuitive.

Colombo et al. [126] have extended the approach of Petri-net-based modeling [127] for the modeling of agent-based manufacturing systems. In their methodology, the agents are identified on the base of a Petri-net model of the production processes and the relevant control decisions. Although this method offers a more rigorous model for the agent-identification stage than the previous method, it does not present any defined approach for the identification of the agents in the Petri-net model (the only guide is the following one, “the agents are decision motors and therefore they are responsible for the decision control”).

Bussmann et al. [81] proposed the methodology DACS (methodology for the design of agent-based manufacturing control systems). The methodology starts with the specification of the manufacturing control problem, which, among other things, includes a specification of the manufacturing components to be controlled and their physical behavior. The first phase, *decision-making analysis*, analyzes those decisions necessary to the operation of the manufacturing system, observing the local decision tasks that appear in each manufacturing component and identifying any

dependence of decisions that could exist among these tasks. The second phase of the methodology is *agent identification*, which has been developed based on the decision model derived in the first phase and produces clusters of decision tasks with the goal of assigning each cluster to an agent. For the cluster process, the method provides some rules that guide the process. This phase offers a set of operations that modify the network of decisions, either separating the decision tasks or introducing new tasks. These operations allow the designer to reorganize the decision model and improve the cluster process. Finally, the third phase of the methodology *selection of interaction protocols*, provides a mechanism for reusing existing interaction protocols with the goal of solving any dependence of decisions among different agents. Once the dependencies have been classified the interaction protocols (which will be used by the agents) are decided. In summary, the successive application of the three phases results in an agents-based design for solving a manufacturing control problem. The resulting design consists of a list of agents with their decision responsibilities (in terms of a set of decision tasks), and a set of interaction protocols for each dependence among different agents. The method proposes, as an implementation step, the independent development of each agent. However, it does not specify how this step is carried out, nor which agent architecture or agent platform can be used. The method is only centered around the control of manufacturing processes without considering the other components of a manufacturing system.

4.4 Enterprise Modeling

In this section we present several studies related to notations and methods for enterprise modeling. A manufacturing system belongs to a manufacturing company, therefore, the system modeling must integrate the enterprise modeling.

In the enterprise modeling two levels are identified [128]: (i) business level, which consists of partially ordered sequences of company activities, activated by the occurrence of events that produce identifiable final results; (ii) company activity level, which represents the set of partially ordered sequences of basic operations executed by active resources of the company. There are several studies in the specialized literature that describe these levels.

CIMOSA [129] is the result of the AMICE project, one of the biggest projects in the European Community in the domain of computer-integrated manufacturing and engineering (CIME). Some of their results have been used as standards for the enterprise modeling domain. CIMOSA allows the definition of the business processes in the life cycle of the system (company engineering environment) and in the life cycle of the product (product engineering environment). It classifies the models and modeling elements into three categories: (i) generic blocks of construction, which are the basic elements of modeling for any manufacturing system; (ii) partial models, which represent instances of generic blocks or aggregates of instances of generic blocks; and (iii) particular models, which are instanced constructions that represent a particular manufacturing system or a particular company. CIMOSA defines four

modeling views: (i) *functional* view: describes the functionalities and tasks of the company; (ii) *information* view: models the information used in the company processes; (iii) *resource* view: specifies the capacities and resources required for executing the processes of the company; (iv) *organization* view: defines the authority and responsibilities of the entities in the organization. The disadvantage of CIMOSA for the modeling of new-generation manufacturing systems resides, logically, in their CIM orientation. However, it is a good reference for the modeling of enterprises due to the conceptual elements that it defines. In fact, the majority of notations are based, or include, concepts defined in CIMOSA.

UEML (Unified Enterprise Modeling Language) [130] arose from a project financed by the European Community for the unification of the different notations of enterprise modeling. This notation integrates, among other things, the following well-known notations: IDEF0, IDEF1x and IDEF3 [122], GRAI net [131], CIMOSA [129], IEM [132], ARIS method [133], etc. UEML 1.0 defines the basic set of concepts and modeling elements. It is based on the identification of common and non-common concepts in the different notations of Enterprise Modeling. It includes the following elements: activity, which is a company behavior that produces outputs from inputs; flow, which represents the flow of an object from an origin toward a destination; anchor, which is the origin or destination of a flow; material resource or human resource, which is a special type of object needed for the execution of an activity; information object, which is an object that can be annexed to a flow. CIM also defines the attributes, associations and restrictions of each one of these elements. UEML is used in the specification of the business level without taking into account the internal specification of the activities of the company.

4.5 Comparative Overview

From the previous sections it can be observed that in the field of MAS a great deal of effort has gone into researching the development of multi-agent methodologies. The large number of studies to be found in the specialized literature is proof of this fact. Most of the MAS approaches are general-purpose approaches, while only a reduced number of MAS methods are defined as specific for the manufacturing system domain. MAS technology has been defined as a promising approach for “new manufacturing” due to its characteristics: distribution, autonomous behavior, cooperation capacity, to name a few [81]. In fact, the majority of the developments in the HMS field use MAS technology to implement their systems (Chap. 2).

The specific MAS approaches for manufacturing systems have some limitations. The proposal of [120] does not use notation, nor the appropriate technology for the modeling of agents. It is based exclusively on OO concepts and on the IDEF model, due to the fact that it loses expressiveness and rigor when modeling agents. Also, IDEF0 is insufficient for modeling all of the aspects of a manufacturing system [81]. On the other hand, the proposals of [125] and of [126] only take into account a small phase of the development process of manufacturing systems. They are fo-

cused on the control of production processes and the identification of agents that will control the process. Neither proposal is sound nor complete, since they are limited to present examples without providing a general definition applicable to a specific manufacturing control problem. DACS [81] is a method specifically for the design of agent-based systems for manufacturing system control. Therefore, it is focused only on the controlling elements of the manufacturing system. The output of this method is an agent-based design, but it does not offer development guidelines or a tool for the implementation of the agent-based system that it designs. However, it is a rigorous method and is easy to use for designers without previous knowledge of agent technology.

In the HMS field there is a recognized need for specific, sound and complete methodologies with uniformity of notation and concepts, for guiding the development of these systems [4]. Two studies that attempt to fulfill these objectives have been reported, but both are very preliminary and therefore lack some key elements for software engineering methods: definition of the development process, notation and tools.

Finally, methods in the area of enterprise modeling are mainly centered on the definition of the business processes without taking into account the modeling of the internal activities of the company. These methods do not have tools for handling the requirements of the “new manufacturing”. Nevertheless, it is very important to have standardized notations, such as the UEML [130].

Finally, we present in Table 4.1 a summary of the comparative study. In this table we include all the methodologies presented in the previous sections, indicating, for each methodology, how it copes the different HMS modeling requirements.

In the following paragraphs we present a discussion on the results of Table 4.1.

- It is evident that *requirement 1* is satisfied by all MAS-based methodology. On the other hand, it is logical that the approaches from the enterprise modeling field do not cope with it due to their motivation.
- *Requirement 2* refers to real-time characteristics for the modeling of control systems in factories. RT-MESSAGE is the only MAS method to consider it. On the other hand, it is surprising that neither the methods of Kendall, Ritter, Colombo nor the DACS method consider this requirement.
- *Requirement 3* is a fundamental requirement of any software engineering method, therefore all of the methods that recognize the implementation stage support it. CIMOSA and UEML only include the modeling of the business processes of the company without going down to the level of modeling and implementation of the tasks inside the company, which is why they do not support this requirement.
- *Requirement 4* is similar to requirement 3.
- *Requirement 5* refers to the mechanisms or procedures defined by a methodology to translate the entities of the manufacturing domain to entities with autonomous behavior. The methods from the HMS field correctly support this requirement, since they are based on holonic architectures that have a direct correspondence with entities from the manufacturing system domain. The methods MESSAGE, RT-MESSAGE and INGENIAS define some guidelines to help the designer in

Table 4.1 Development methods and modeling requirements for HMS

Method	Requirements							
	1	2	3	4	5	6	7	8
Leitão and Restivo method	✓		✓	✓	✓			
Fischer et al. method	✓		✓	✓	✓			✓
Kinny and Georgeff method	✓		✓	✓				
Burmeister method	✓		✓	✓				
MAS-CommonKADS	✓		✓	✓				
GAIA	✓		✓	✓				
GAIA v.2	✓		✓	✓				
ROADMAP	✓		✓	✓				
MASSIVE	✓		✓	✓				
Tropos	✓		✓	✓				
MaSE	✓		✓	✓				
MESSAGE	✓		✓	✓	✓	~		
RT-MESSAGE	✓	✓	✓	✓	✓	~		
INGENIAS	✓		✓	✓	✓	~		
MASB	✓		✓	✓	✓			
Prometheus	✓		✓	✓	✓			
Elammari and Lalonde method	✓		✓	✓				
Miles method	✓		✓	✓				
Cassiopeia	✓		✓	✓				
Agent societies	✓		✓	✓				
Civil agent societies	✓		✓	✓				
Kendall et al. method	✓		✓	✓	~			
Ritter et al. method	✓		✓	✓	~			
Colombo et al. method	✓		✓	✓	~			
DACS	✓		✓	✓	~			
CIMOSA								✓
UEML								✓

the agent-identification process. These methodologies do not completely fulfill this requirement since they propose general guidelines (due to their general orientation) and consequently they do not approach, in a specific way, the specific characteristics of the manufacturing problems and functionalities. The MAS methods for manufacturing systems take into account the specific characteristics of the entities of the manufacturing control systems domain. However, they do not take into account the other components of the manufacturing system.

- Only the approaches of enterprise modeling consider *requirement 6*. The methods from the HMS field do not take this requirement into account.
- *Requirement 7* refers to mixed development methods (bottom-up and top-down). All of the methods analyzed follow one approach or another without combining them.
- Finally, *requirement 8* is only considered in an explicit and specific manner in the method of Fischer. It is true, on the other hand, that with some MAS methodologies large-scale systems (which represent a whole manufacturing system) can be

modeled, but the guidelines defined in them are not specific and therefore in the analysis and design phases the effort and the designer's attention may be focused on less important elements for these domains [81].

4.6 Conclusions

The modeling of holonic manufacturing systems constitutes the fundamental interest of this book. There are few studies in the HMS field related to modeling methods and techniques. The preliminary state of the studies regarding HMS methodologies [19, 58, 57], is evidence of the immaturity of this area. These facts motivated us to study development methods from HMS, MAS and enterprise modeling, and to compare them in order to obtain a qualitative measure of the completion of each one for HMS development. To carry out this comparison we have defined eight HMS modeling requirements. These requirements have been divided into two groups: functional requirements and software engineering requirements. The first ones refer to the type of programs that should be developed when applying a methodology, while the second refers to the properties of the software engineering method. All of these requirements are specific for HMS. The MAS methods may seem suitable for developing HMS because there are several methods, many of which are general purpose and a reduced group of them is specific for the manufacturing domain. And more importantly, the agent technology is the implementation tool used the most in the HMS field. Finally, the field of enterprise modeling offers interesting studies and proposals for the standardization of notation for modeling company business processes. These studies can be applied (and, in fact, have been applied) as modeling notation for manufacturing companies. With these three groups: HMS methods, MAS methods and enterprise modeling, we have carried out the comparison based on the eight HMS modeling requirements. The result is Table 4.1. From this study it is evident that there is a need to develop a methodology for HMS. This is the main contribution of this book, and we present the results in Chaps. 5 and 6.

Chapter 5

ANEMONA Notation

In this chapter we present the ANEMONA notation. The ANEMONA development process is presented in Chap. 6. ANEMONA is a MAS methodology for HMS analysis and design based on the abstract agent notion and HMS modeling requirements (Chap. 4). ANEMONA integrates features from HMS, MAS and enterprise modeling techniques [129, 130] (Chap. 4).

In ANEMONA the HMS is specified by dividing it into more specific characteristics that form different *views* of the system. These views are defined in terms of MAS technology; therefore, we talk about agents, roles, goals, beliefs, organizations, etc. We use abstract agent and holon as similar notions [59] (Chap. 4).

In ANEMONA there are five views or models. The *agent model* (Sect. 5.3) is concerned with the functionality of each abstract agent: responsibilities and capabilities. The *organization model* (Sect. 5.7) describes how system components (abstract agents, roles, resources, and applications) are grouped together. The *interaction model* (Sect. 5.5) addresses the exchange of information or requests between abstract agents. The *environment model* (Sect. 5.6) defines the non-autonomous entities with which the abstract agents interact. The *task/goal model* (Sect. 5.4) describes relationships among goals and tasks, goal structures, and task structures.

In order to introduce the ANEMONA process we present here a short overview of it. The complete process is explained in detail in Chap. 6. In Fig. 5.1 we can see the development stages of ANEMONA¹. The first stage, *system requirement analysis* and the second stage *holon identification and specification* define the analysis phase (Sect. 6.3.2). The aim of the analysis phase is to provide high-level HMS specifications from the problem *requirements* (Sect. 6.3.1). The analysis of ANEMONA is a top-down recursive approach. The next stage in the development process is the *holon design* stage (Sect. 6.3.3) which is a bottom-up process to produce the *system architecture* from the *analysis models* of the previous stage. The aim of the *holon implementation* stage (Sect. 6.3.4) is to produce an *executable code* for the *setup and configuration* stage (Sect. 6.3.5). Finally, maintenance functions are executed

¹ The specification of the development process of ANEMONA is presented using SPEM diagrams [134].

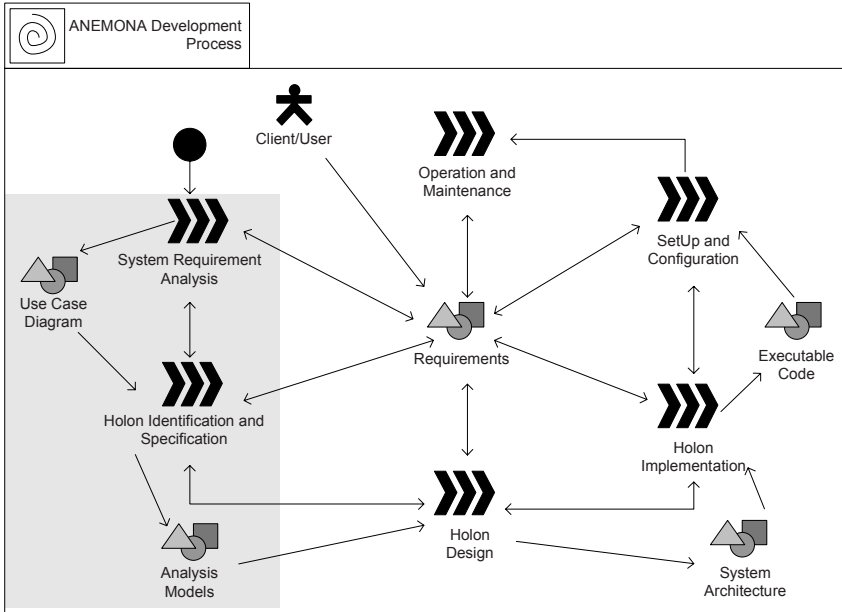


Fig. 5.1 The ANEMONA development process

in the *operation and maintenance* stage (Sect. 6.3.6). The ANEMONA notation defines the models that the software engineer has to build in the different development phases.

The ANEMONA notation is based on two complete MAS methodologies for general-purpose domains. These are INGENIAS [135] and RT-MESSAGE [136]. An analysis on the HMS adequation of these two methods can be found in Chap. 4. Both methods are extensions of the MESSAGE methodology [103, 104] so they share their basis and MAS conceptual models.

The ANEMONA notation is inspired on INGENIAS models and metamodels and the real-time specification elements of RT-MESSAGE. The abstract agent notion described in Chap. 3 is the ANEMONA central modeling entity.

5.1 ANEMONA Metamodel

The notation of ANEMONA is specified using metamodels. Metamodeling is a mechanism to formally define modeling language. A language metamodel is a precise definition of its elements using concepts and rules from a metalanguage. These concepts and rules are necessary for building models in such a language. A metamodel defines the primitives and the syntactic and semantic properties of a model. In other words, a model is described using models.

The ANEMONA metamodel is defined using the UML [137] language with the constraints defined in [138]. In this way the metamodel definition uses UML entities such as: objects, relationships, roles, and properties. All these entities are graphically represented as boxes (Fig. 5.2a), which contain the entity stereotype within $\ll \gg$, and the ANEMONA modeling entity name. In addition to these entities, in the ANEMONA metamodel definition we use the following UML relations.

- The inheritance relation, Fig. 5.2b, denotes that the entity in the arrow ending is the generalization of the entity in the other ending. That is, for example in Fig. 5.2b, object “A” generalizes object “B”, or at the same time object “B” inherits the definition of object “A”, or object “B” is an object “A”. For instance, let’s suppose we have two entities, “Person” and “Worker”, an inheritance relation from “Worker” to “Person” denotes that “Person” is a generalization of “Worker”, a “Worker” is a “Person”, etc.
- The association relation, Fig. 5.2c, denotes that there is a conceptual dependency among the entities linked by the association line. For example, let’s assume two entities “Factory Cell” and “Machine”. An association relation linking these two entities may represent a conceptual dependency: a “Machine” is in a “Factory Cell”. The cardinality attributes of the association relation may describe relations with one-to-one, one-to-many and many-to-many cardinality.
- The aggregation relation, Fig. 5.2d, denotes that one entity is part of another entity. For example in Fig. 5.2d, object “B” is part of object “A”, in this case object “A” is the whole and object “B” is the part. The cardinality attributes of the aggregation relation may describe relations with one-to-one, one-to-many and many-to-many cardinality.

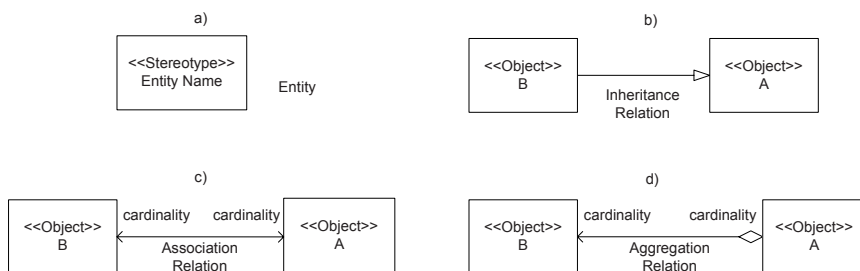


Fig. 5.2 UML entities

In order to facilitate the reading of the metamodel diagrams presented in this chapter we use the mnemonic rules of Fig. 5.3. In addition, for every association relation between a relationship entity and an object entity there is an instance of the UML *role* primitive. This instance is named with the same name of the relationship entity but with the prefix *R* and ending with the letter *D* or *O*, in order to indicate the destination of the relationship (*O* for origin, *D* for destination).

```

RelationIdentifier ::= WorkflowRelation | AgentRelation | InteractionRelation |
                    InteractionUnitRelation | MetaTaskRelation | SocialRelation |
                    OrganizationRelation | EnvironmentRelation
WorkflowRelation ::= WFIdentifier
AgentRelation ::= AIdentifier
InteractionRelation ::= IIdentifier
InteractionUnitRelation ::= UIIdentifier
MetaTaskRelation ::= GTRelation
SocialRelation ::= AGORelation
OrganizationRelation ::= ORelation
EnvironmentRelation ::= ERelation

BinaryAssociationRollIdentifier ::= RRelationName(OID)
NaryAssociationRollIdentifier ::= RRelationNameIdentifier(OID)
    
```

Fig. 5.3 Mnemonic rules for roles and relations

The following sections describe the ANEMONA notation. These sections are sketched, presenting first the metamodel defining the model, and secondly the description of how to build the model.

5.2 Basic Modeling Entities

The basic modeling entities of ANEMONA are presented in Fig. 5.4. These entities are used to build the different ANEMONA analysis and design models. The way in which these entities can be related to each other and the meaning of the allowed model structures of ANEMONA are detailed in Sections 5.3 to 5.7.

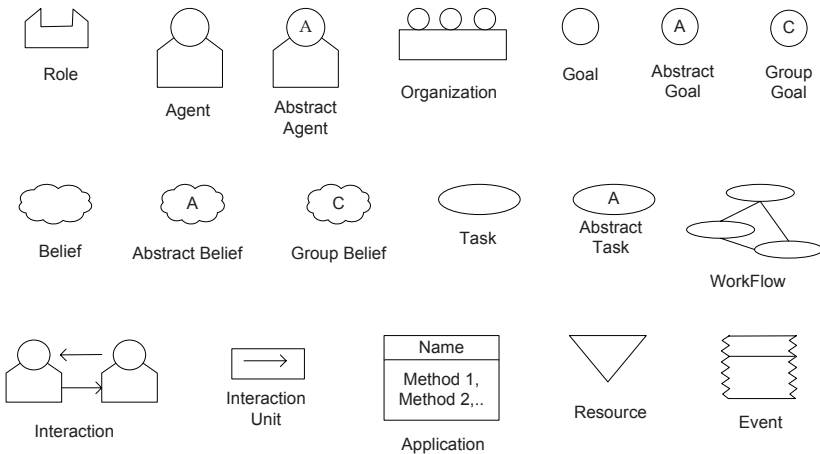


Fig. 5.4 The ANEMONA graphical notation

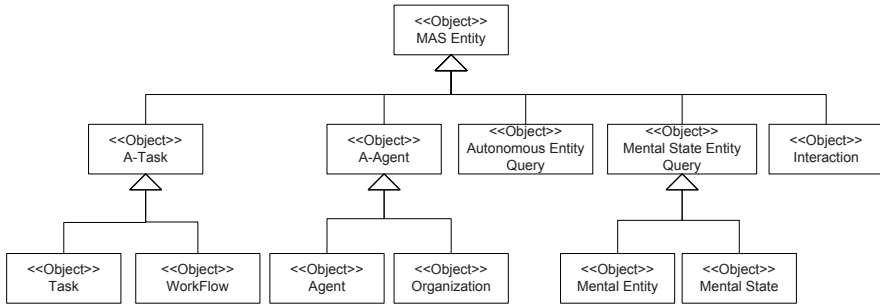


Fig. 5.5 The metamodel specification of the basic entities of ANEMONA

The metamodel of Fig. 5.5 defines the hierarchical relationship among the basic modeling entities of ANEMONA.

Role: a role is an encapsulation of certain attributes and behaviors of the abstract agent it is bound to. In a company, for example, there are different roles such as, the company president, the company supervisor, the company worker, etc. In the ANEMONA graphical notation the role is depicted as in Fig. 5.4 labeled with the role name.

An abstract agent can have more than one role at a time, and can also dynamically change its roles. A role does not take action, however, an abstract agent does. Roles are not isolated: there must be other roles related to them. A role acts as a “window” of an abstract agent, through which other abstract agents know the way to interact with the abstract agent. Roles provide a facility for efficient reuse. In other words, a role groups a set of functions, responsibilities, interactions, and represents a given position in an organization structure.

Abstract agent: an abstract agent (A-Agent for short in the metamodel diagrams) is an autonomous entity that is abstract and complex. An abstract agent represents non-atomic holons that are in turn composed of holons. The designer’s point of view will determine the nature of what is being observed at each moment. From the outside, a system can be considered an abstract agent since it has agenthood characteristics. On the other hand, from the inside, that is, from the internal structure, the abstract agent can be considered as being composed of a group of interrelated abstract agents (holons). When there are no more subdivisions, the abstract agents can be considered as being a simple agent (atomic holons). The end of the subdivision is defined by the designer since the subdivision exists whenever it is useful for the definition of the problem being modeled. In the end, at the lowest abstraction level, only the agents that make up the global MAS will be apparent, but as the abstraction levels go up, there will be some agents and some abstract agents that are refined as MASs.

For example, in a manufacturing company, let’s imagine a manufacturing company’s production department that interacts with a sales department. In order to focus only on specifying the interaction cooperation among the departments, without worrying about their internal structures both departments may be modeled as

interacting abstract agents. On the other hand, when analyzing, for example, the production department abstract agent is apparent that it is a holarchy composed of other holons, such as, machines, products, production planning, raw materials, parts, etc. In the ANEMONA graphical notation the abstract agent is depicted as in Fig. 5.4 labeled with its name.

Agent: an agent is an autonomous, reactive, proactive and social computational entity, which is able to act in an environment. In a HMS an agent may represent atomic holons, for example a press machine and the controlling system associated with it, a worker, an assembling part with its assembling rules and its quality specification, etc. In ANEMONA it is graphically represented as in Fig. 5.4 labeled with its name. An agent is a specialization of an abstract agent (Fig. 5.5).

Organization: an organization models a group of abstract agents which cooperate to achieve common goals. An organization consist of a group of roles, a communication structure among these roles, and a social structure defined by the cooperating roles. In a HMS an organization is used to represent the internal structure of holarchies. For example, let's imagine a factory production line with machines, workers, products, parts, supervisors, work orders, etc. These holons cooperate to fulfill the production line goal. They are related by means of social relations, client-server relations, etc. The production line with all of its members and their relations is modeled as an organization. In ANEMONA an organization is graphically represented as in Fig. 5.4 labeled with its name. An organization is a specialization of an abstract agent (Fig. 5.5).

MAS technology uses the notion of *mental state* (Fig. 5.5, and Sect. 5.3), in order to be able to define the autonomous behavior of autonomous entities. That is, what an autonomous entity is doing at a given time, what it is looking for, and what idea it has about its environment. In ANEMONA the mental state of an autonomous entity is defined using the concepts of goals and beliefs.

Goal: a goal describes what an agent (atomic holon) is trying to fulfill, what it is looking for, the reason for its execution, etc. It is graphically depicted as can be seen in Fig. 5.4.

Abstract goal: an abstract goal is used to represent the goal of an abstract agent (non-atomic holon). An abstract goal bounded to an abstract agent may be decomposed into goals of agents (members of the abstract agent) and/or into group goals of a group of agents. Examples of goals for a factory cell could be to maximize production performance, minimize waiting time and raw materials, etc.

Group goal: a group goal is used to represent a goal that emerges from a group of agents in an abstract agent and so can not be assigned directly to any of those agents. For example, let's consider a machine holon with the goal $o_1 = \text{to produce a new product A every 3 minutes}$, and a seller holon with the goal $o_2 = \text{to sell a product A 2 minutes after it was produced}$. And let's suppose the two holons are in a group with the group goal $o_{group} = \text{to lead the market sell of product A}$. This group goal is achieved by goals o_1 and o_2 but cannot be associated to either of the two holons.

Belief: a belief is the mental entity that models the idea that an agent has about its surroundings. Examples of beliefs in a machine holon are: the next product in

the waiting queue, the time elapsed from the last product change, the amount of materials left, etc. A belief is graphically depicted by a cloud in Fig. 5.4.

Abstract Belief: an abstract belief is used to model the belief of an abstract agent. It may be decomposed into agents beliefs or group beliefs.

Group Belief: a group belief is used to model the belief of an abstract agent when it is an organization (a holarchy).

Task: a task is used to model the capabilities of an agent. A task represents a functionality of the agent. An agent can modify its environment by means of tasks. Example of tasks associated with a work-order holon could be to get production resources, to control work orders, to resume a production plan, to execute a production order, etc. A task is graphically represented as depicted in Fig. 5.4.

Work Flow: a work flow is a set of tasks that are executed in a given order by many agents. A work flow is defined within an organization. It is used to model the way in which the organization or its member agents implement the functionality they have. It can also model a plan to fulfill some goal. Example of work flows associated with a warehouse organization are, to track raw materials, to control raw material order, etc.

Abstract Task: an abstract task is used to represent an abstract agent capability. It can be a task (when the abstract agent is a single agent) or a work flow (when the abstract agent is a group of agents). An abstract task is graphically represented as depicted in Fig. 5.4.

Interaction: an interaction is used to specify dependencies among abstract agents and to define their behavior. An interaction shows what the reaction of an abstract agent is to a given event, message, environment status, etc. It also shows how an abstract agent's behavior is related to its goals and tasks. An interaction is specified using the following elements:

- The participating actors, that is the different abstract agents involved in the interaction. In any given interaction there is an initiator actor and one or more collaborator actors.
- The definition of the *interaction units*. An interaction unit can be a message or an event. An interaction unit details who executes it (the abstract agent that initiates it), who collaborate with it (the abstract agent to which the interaction unit is sent), and what tasks are executed in the interaction unit.
- An ordering definition on the interaction units that may up the interaction.
- The actions (tasks) that are executed in the interaction. This involves the specification of when a task has to be executed, and the consequences of executing it.
- The interaction context, that is the specification of what happens in the system when the interaction starts, while it is executing, and when it has finished.

The complete specification of an interaction is detailed in Sect. 5.5.

Application: an application is used to model all those services that are outside of the HMS, and are not implemented by any autonomous entity. An application is

used to interact with software systems that are already implemented or are outside of the HMS being modeled. Examples of applications are, a database system that manages the bank accounts of the company, a communication system that is used for intercompany information exchange, the Internet, etc.

Resource: a resource is a software/hardware element that is necessary for HMS execution. Resources can be: a data file, an execution thread, memory, input devices, output devices, information storage units, sockets, bandwidth, etc.

Event: an event models changes in the environment that were perceived by an abstract agent. An event is graphically represented as in Fig. 5.4. Examples of events associated with a conveyor belt are: the product *x* is entering the belt, the product *x* is just in front of machine *z*, the product *x* is exiting the belt, etc.

5.3 Agent Model

The agent model specifies all the details of the abstract agents that make up the HMS. It is used to model the abstract agents' autonomy, intelligence and conceptualization. The agent model of a given HMS is a set of agent diagrams. Every abstract agent is specified by means of an agent diagram. The allowed entities in an agent diagram are: abstract agent, agent, role, abstract goal, goal, abstract belief, belief, abstract task, and task. Also, the relationships among these entities are specified.

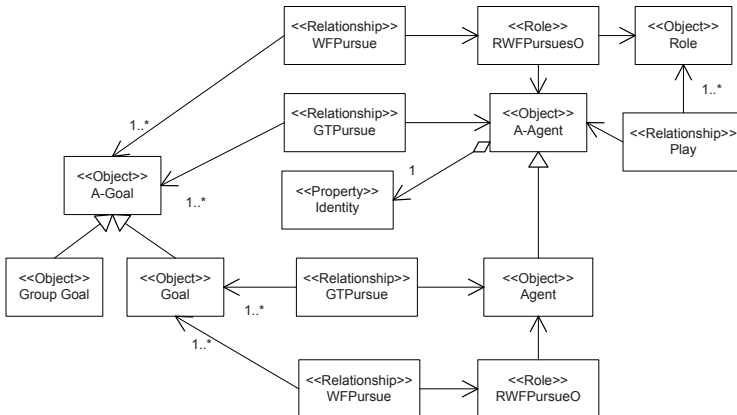


Fig. 5.6 Agent metamodel: role, abstract agent and goals

5.3.1 Abstract Agent and Role

A role is used to group a set of autonomous entity functionalities or responsibilities. A role has a name and a set of goals and tasks associated with it (Sect. 5.7). The *play* relation is used to bind a role to an abstract agent or to an agent (Fig. 5.7). This relationship means that the abstract agent is responsible to fulfill the goals, to execute the tasks, and has the mental state associated to the given role. Also to these entities inherited from the role it plays, the abstract agent may have its own goals, tasks, and mental state (in the following sub-sections these relationships are explained). Figure 5.6 shows the metamodel specification of the *play* relation between an abstract agent and a role. An abstract agent can play one or more roles when participating in organizations.

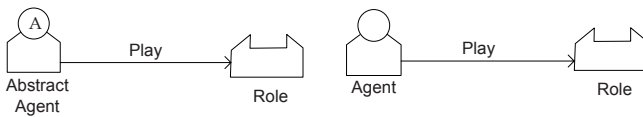


Fig. 5.7 Play relation

Figure 5.12 shows the *play* relation between a *factory holon* and a *production manager role*.

5.3.2 Abstract Agent, Role and Goal

The abstract agent autonomy is specified by means of abstract goals. An abstract agent acts in order to fulfill its goals. Figure 5.6 shows the metamodel specification of the relationship *GTPursues* among goals and agents (Fig. 5.8). Figure 5.6 also states that to be autonomous an abstract agent must at least have an abstract goal associated with it.

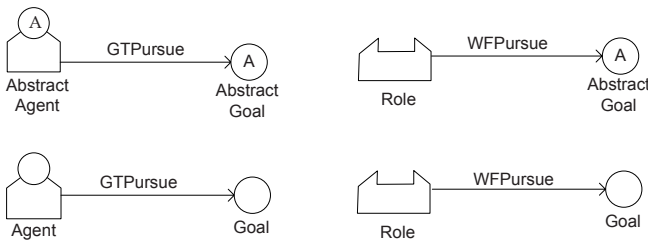


Fig. 5.8 GTPursue and WFPursue relations

5.3.3 Abstract Agent and Belief

The abstract agent information structure is built specifying the beliefs it has. In order to do this, the abstract agent is related to belief entities by means of the *AContainE* relation (Fig. 5.9). Figure 5.10 details the metamodel specification of the relation *AContainE* among a mental state entity and abstract agents.

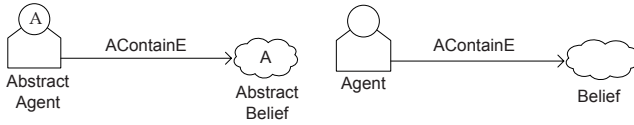


Fig. 5.9 AContainE relation

5.3.4 Abstract Agent, Role and Task

The abstract agent’s capabilities are specified with a set of tasks that it is able to execute. The abstract agent and the tasks bound to it are modeled by means of the *AResponsible* relation (Fig. 5.11). Figure 5.10 details the metamodel specification of the *AResponsible* relation among an abstract task or task entity and an abstract agent or agent. In order to be an autonomous entity it has to be related to at least one task or abstract task. The task entity is explained in detail in Sect. 5.4.

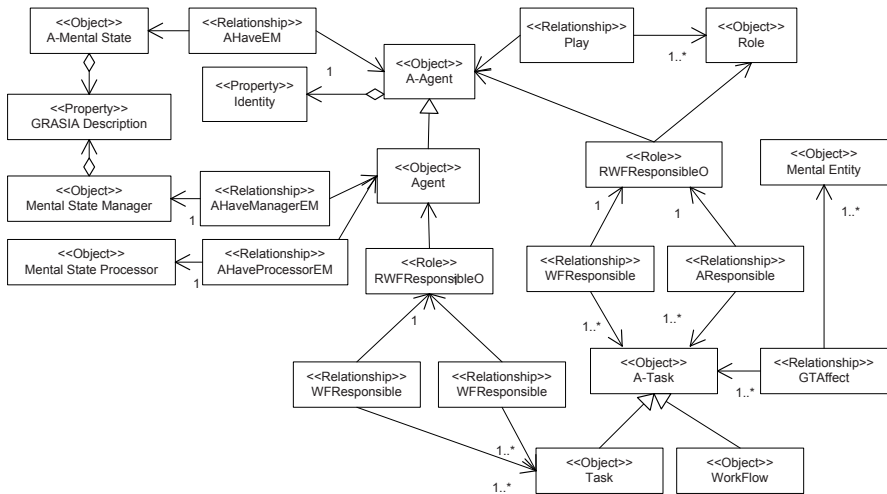


Fig. 5.10 Agent meta-model: roles, abstract agents, beliefs, and tasks

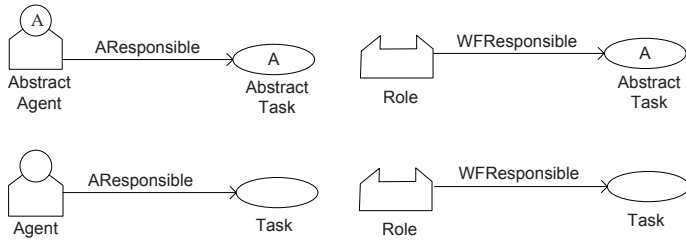


Fig. 5.11 AResponsible and WFResponsible relations

A role entity may have responsibilities as well, which are specified using the *WFResponsible* (Fig. 5.11). This relation models the functions of the role into the organization it is involved in.

Figure 5.12 shows a sample agent diagram for a general and simplified *factory holon*. The *factory holon* is modeled as an abstract agent because it is complex. It models the whole factory that is composed of a set of lower-level agents (the details of the *factory holon* is discussed in Chap. 8). The *factory holon* *Plays* the *production manager* role and pursues a set of abstract goals by means of the *GTPursue* relation. These abstract goals will be decomposed or refined in more specific goals, when the internal structure of the *factory holon* is described. The *factory holon* groups its beliefs into the *information structure* mental state entity by means of the *AContainE*

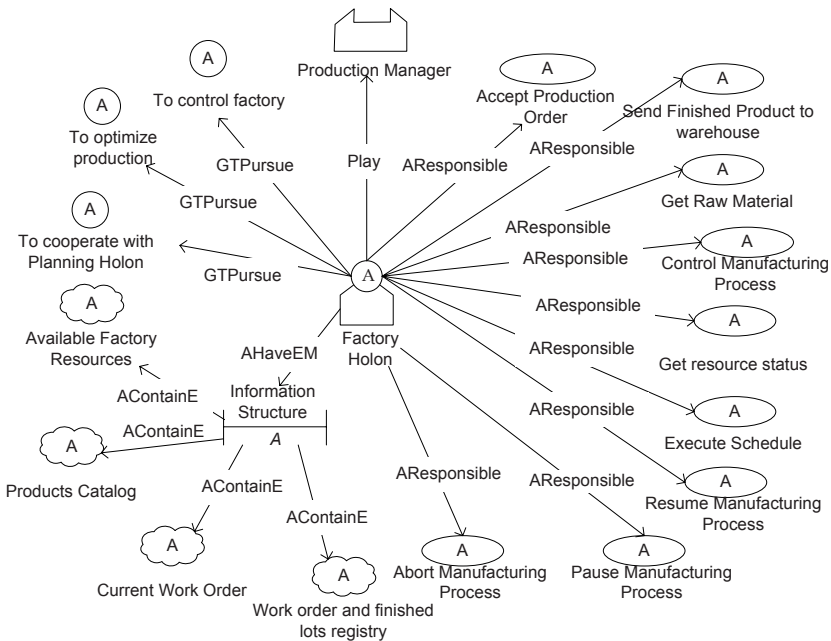


Fig. 5.12 A sample agent diagram

relation. A mental state entity is used to group the beliefs of an autonomous entity. A sample list of abstract tasks related to the *factory holon* can also be seen in this figure. Chapter 8 presents more sample agent diagrams.

5.4 Task/Goal Model

The task/goal model is used to specify the motivation of the HMS. It defines the actions identified in the organization model (Sect. 5.7), the interaction model (Sect. 5.5) and the agent model (Sect. 5.3). The task/goal model tries to specify the consequences of executing these tasks and why they have to be executed. The actions executed by an autonomous entity are motivated by the goals it pursues. This model is intended to specify the relations among the goals of an autonomous entity and the tasks this autonomous entity can perform.

A task can be seen as a transformation in the global system state, an event response, a process, a physical action or a command. A goal is used in ANEMONA to specify reasoning processes for decision making on different alternatives for agent actions in a given situation.

The task/goal model of a given HMS is a set of task/goal diagrams. In the task/goal model all of the HMS identified tasks have to be specified, stating the goal or goals that motivate the task, the effects of executing the task, the decomposition or not of the task, the goal composition or decomposition and the dependencies among the goals. The entities allowed in a task/goal diagram are: abstract task, task, work flow, abstract goal, goal, group goal, agent, abstract agent and role. The following relations can also be used.

5.4.1 Abstract Agent, Task and Goals

Figure 5.13 presents the relations allowed among abstract agents, roles, abstract tasks, tasks, abstract goals, goals, and group goals. These relations have already been described in the agent model and are used in the same way. The real-time characteristics of the functioning of a HMS are specified by means of real-time task and goals.

A real-time task [136], Fig. 5.13, is usually associated to the factory floor machine resources, and its controlling operation. A real-time task is one in which its successful execution is not only determined by its correct results but also by the time in which those results are obtained. In order to model a real-time task in a task/goal diagram the following properties must be specified: the *task type*, its *Max deadline*, and if it is *periodic*. These properties are optional and are used only when the task has real-time features. For a nonreal-time task these properties are left blank. The *Task type* defines if the task has *hard* real-time characteristics or (*soft*) real-time characteristics and, if it is *periodic* or *aperiodic*. A *hard* real-time value of this prop-

erty means that if the time constraints are not adhered to the results of the task are not useful so far. In this case the system may collapse. A *soft* real-time value means that it is desirable to adhere to the time constraint but if it cannot it only means that the result will be poorer than the first case. The *Max deadline* defines the max time allowed to get a result from the task. The *Period* property defines the time interval in which the task is executed when it is a *periodic* task.

A real-time goal [136], Fig. 5.13, is used to specify goals associated to holons that control machine resources or communication interactions with time constraints. In order to model a real-time goal the following properties are defined: *max deadline* and *goal type*. The *max deadline* property is used only when the goal has real-time constraints, otherwise it is left blank. It is used to specify the maximum time available in order to fulfill the goal. The *goal type* property may have the values: *hard*, *soft* or *normal*. Its default value is *normal* and is not depicted graphically. It is used when there is no time constraint associated with the goal. The *hard* value means that it is a goal with hard real-time characteristics, and the *soft* value means that the time constraint is only desirable but not crucial. When working with real-time goals and real-time tasks, the *max deadline* of a real-time goal constrains the *max deadline* of the task that satisfies that goal. In other words, the task *max deadline* has to be lower than or equal to the goal *max deadline*.

Let's take, for example, a controlling hard-time constraint is associated to the heating control function: "The product's cooking temperature of 100 degrees Celsius cannot

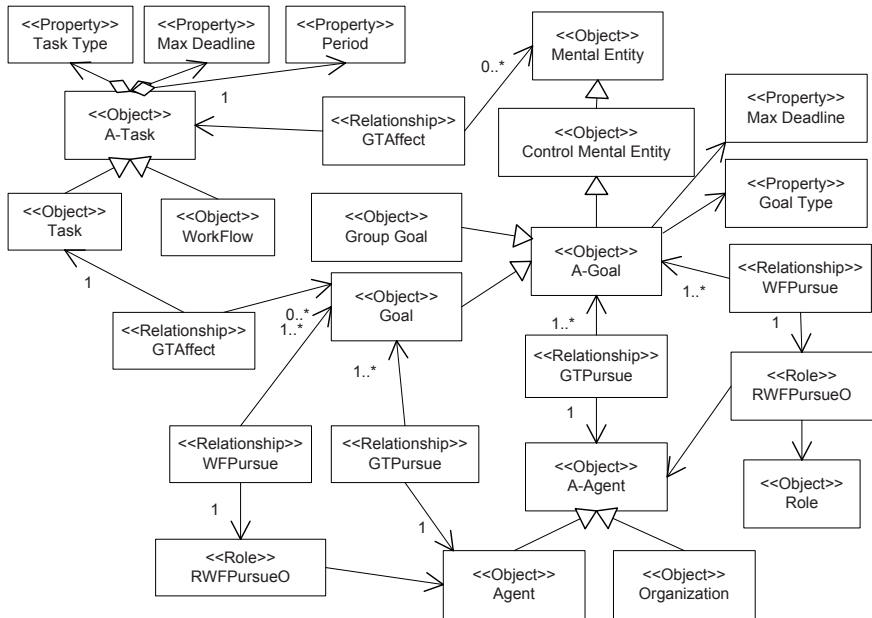


Fig. 5.13 Task/Goal metamodel: abstract agent, task and goal

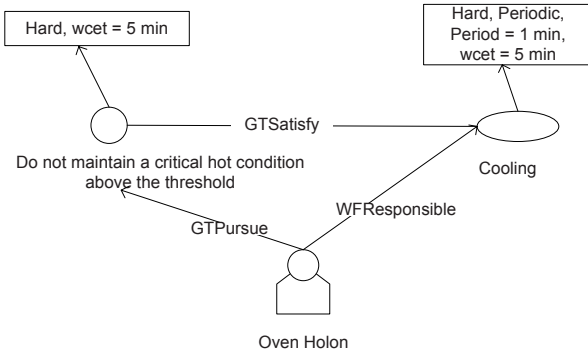


Fig. 5.14 A sample task/goal diagram with one real-time task and one real-time goal associated to an oven holon

be exceeded for more than 5 minutes in the cooking process”. In order to model it, a *cooling* task is defined as *hard* and *periodic*, with the *max deadline* (wcet) of 5 min, and a *period* of 1 min. The real-time goal associated to this task is “do not maintain a critical hot condition above the threshold”. This goal is defined as *hard*, and with the *max deadline* (wcet) of 5 min. The specification of this example is illustrated in Fig. 5.14.

5.4.2 Task, Goals and Beliefs

Figure 5.13 defines the relation *GT Affect* among abstract tasks and abstract goals. *GT Affect* is used in order to model the autonomous behavior of an autonomous entity. In other words, the tasks are executed in order to “affect” the mental entities

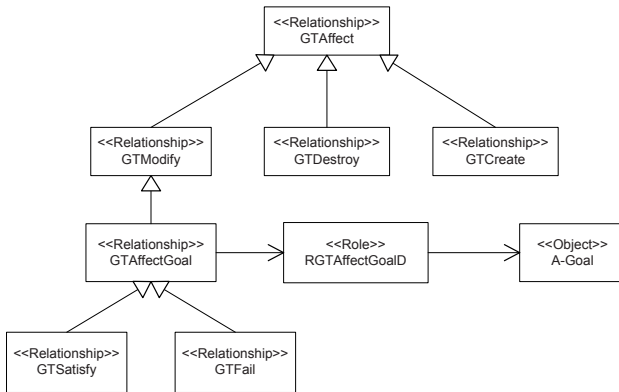


Fig. 5.15 Task/goal metamodel: task, goal, and belief relations

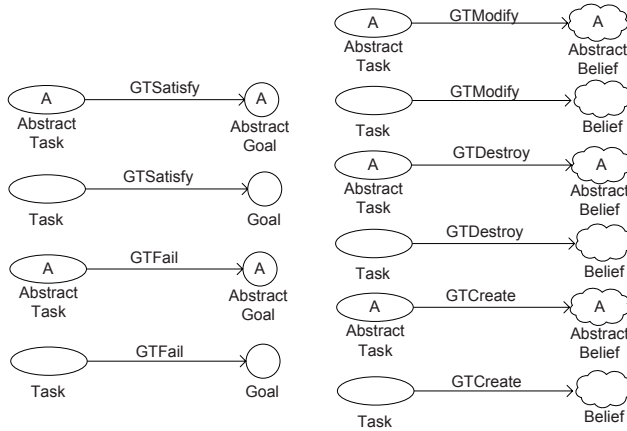


Fig. 5.16 GTSatisfy, GTFail, GModify, GCreate and GDestroy relations

of the abstract agent that executes them. The mental entities modification of other abstract agents is modeled by means of interaction diagrams (Sect. 5.5).

The *GTAffect* relation is specialized in the relations presented in Fig. 5.15. These relations are graphically represented as in Fig. 5.16. *GTSatisfy* is used to specify that a task/abstract task satisfies a given goal/abstract goal. *GTFail* is used to model the situation when the execution of a task/abstract task causes the goal/abstract goal to fail. On the other hand, the relations *GModify*, *GCreate* and *GDestroy* are used among tasks and beliefs. *GModify* specifies that a task/abstract task modifies the value of a given belief/abstract belief. *GCreate* means that the execution of a task/abstract task creates a given belief/abstract beliefs, and *GDestroy* represents the destruction of the given belief/abstract belief by the task/abstract task. Figure 5.23 illustrates these relations in a sample task/goal diagram.

5.4.3 Task Specification

In order to specify a task or an abstract task the software engineer may use the previously defined relation *GTAffect* and the relations *WFConsume*, *WFUse* and *WFProduce* (Fig. 5.17). These relations are used as shown in Fig. 5.18.

WFConsume is used to model preconditions. Its semantic is that in order to execute a given task, abstract task or work flow, it needs a given belief, abstract belief and/or resource as a precondition. The relation *WFUse* is also used to specify task preconditions. In this case it models the fact that there is a need for an application in the task, abstract task or work-flow execution. On the other hand, the *WFProduce* relation is used to specify the effects of a task. These effects could be a resource availability, or an interaction execution. The *GTAffect* relation can also be used to specify postconditions. Its semantic is defined in the sub-section above.

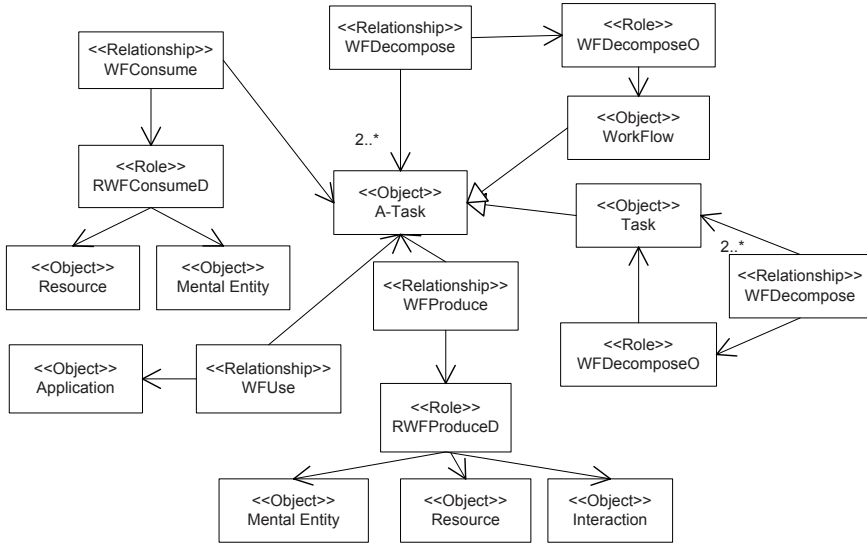


Fig. 5.17 Task/goal metamodel: task specification

A complex task or an abstract task can be decomposed into simpler sub-tasks in order to make the process of implementing it easier. A complex task can be useful

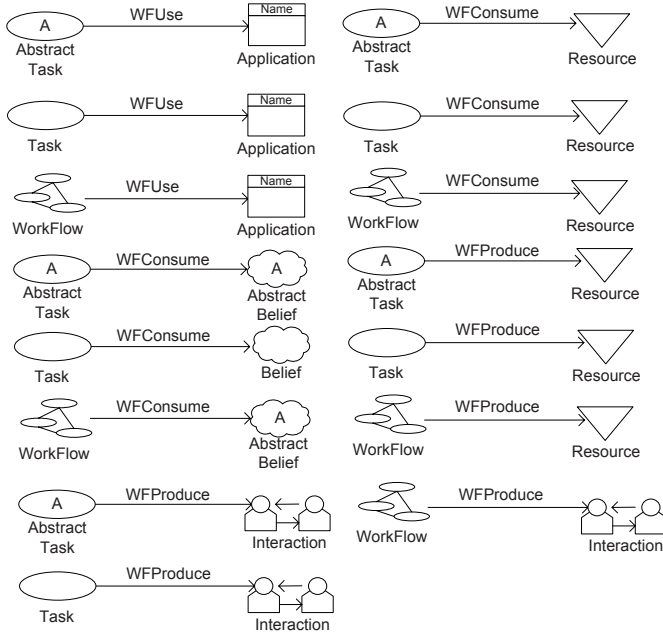


Fig. 5.18 WFConsume, WFUse, and WFProduce relations

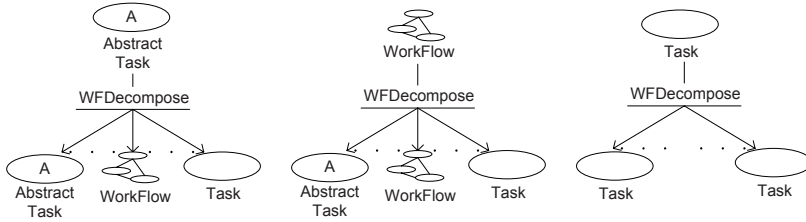


Fig. 5.19 WFDecompose relation

at a given abstraction level in which the focus is not task specification but task interaction with other modeling elements such as goals, agents, interaction scenarios etc. Nevertheless, when the modeling focus is task specification, the complex task has to be decomposed in order to define its insights. This is done using the relation *WFDecompose*. Figure 5.17 shows that this relation is allowed between abstract tasks. That is, an abstract task can be decomposed into abstract tasks, tasks or work flows. A work flow can also be decomposed into abstract tasks, tasks, and work flows, while a task can only be decomposed into other tasks. Figure 5.19 shows the graphical representations of all these cases.

5.4.4 Goal Decomposition and Goal Dependencies

Goal specification plays a very important role in the modeling of autonomous entities. In order to specify goals the system engineer needs a mechanism to build

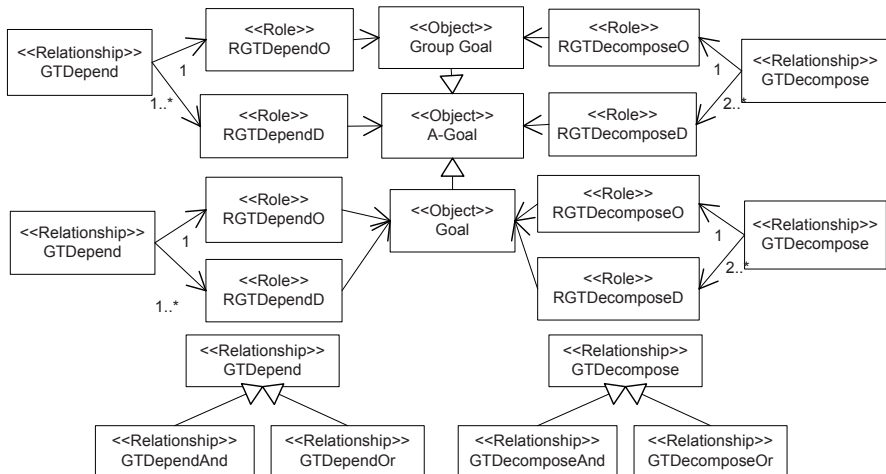


Fig. 5.20 Task/goal metamodel: goal specification

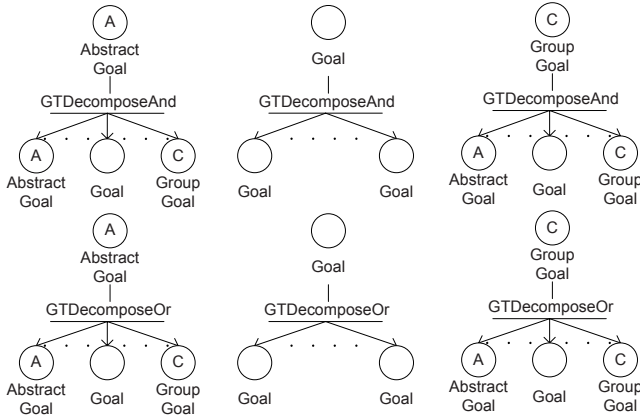


Fig. 5.21 GTDecomposeAnd and GTDecomposeOr relations

the structural aspects of the goals and the dependencies among them. In addition to the relations already described, there are goal-associated relations that can be used to specify the goal structure that makes up the motivational mental state of an autonomous entity. These relations are described in Fig. 5.20. There are basically two relations: *GTDepend* and *GTDecompose*. They are used among abstract goals, group goals, and goals.

GTDecompose is used to decompose complex goals into simpler sub-goals. This relation is a little different from the *WFDecompose* relation for complex tasks. *GTDecompose* is refined into *GTDecomposeAnd* and *GTDecomposeOr* (Fig. 5.21). These relations are used to specify that a complex goal can be implemented by different sub-goals. *GTDecomposeAnd* is used to specify that the fulfillment of the complex goal imposes the fulfillment of all the sub-goals into which it is decomposed. While *GTDecomposeOr* is used when, in order to fulfill the complex goal, only the fulfillment of a set of sub-goals is required, not all of them. The sub-goals that configure this set are specified using the *GTDepend* relation.

The relation *GTDepend* is used to specify that the fulfillment/failure of one goal affects the fulfillment/failure of another goal. In other words, when a goal o_a is fulfilled/failed, the goal o_b associated with it by means of *GTDepend* is fulfilled/failed automatically. *GTDepend* is refined in *GTDependAnd* and *GTDependOr* (Fig. 5.22). *GTDependAnd* is used to specify that when all the goals $\{o_1, o_2, \dots, o_i\}$ associated with a given goal o_x are fulfilled then the goal o_x is also fulfilled. *GTDependOr* is used to specify that in order to fulfill o_x the fulfillment of only one goal of $\{o_1, o_2, \dots, o_i\}$ is needed. These relations are also used to specify failures in the goal o_x . When the relation used is *GTDependAnd*, the semantic is as follows: if any goal of $\{o_1, o_2, \dots, o_i\}$ fails this causes o_x to fail. On the other hand, the relation *GTDependOr* implies that for o_x to fail, it is necessary for all of the $\{o_1, o_2, \dots, o_i\}$ goals to fail.

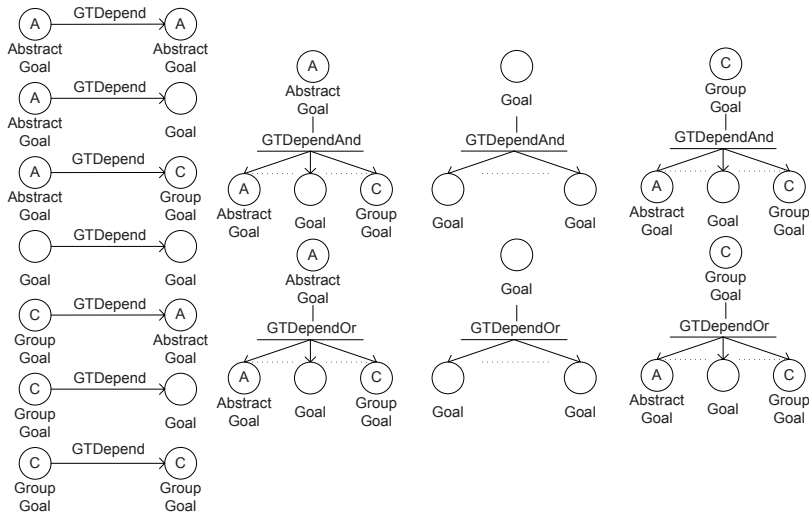


Fig. 5.22 GTDepend, GTDependAnd and GTDependOr relations

Figure 5.23 shows a sample task/goal diagram of a general *factory* holon. In this figure we can see some sample instances of the relations described above for tasks and goals.

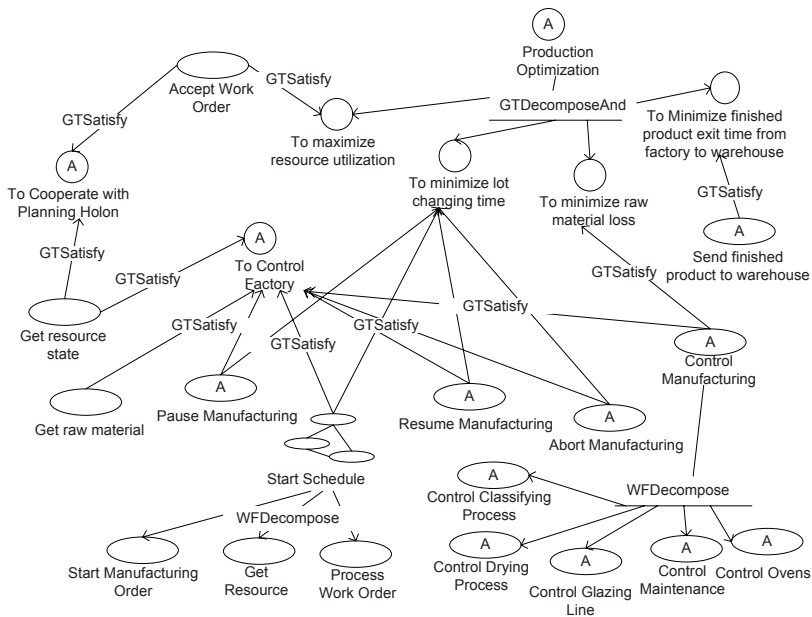


Fig. 5.23 A sample task/goal diagram of a general factory holon

5.5 Interaction Model

The interaction model is used to specify the dynamic behavior of the HMS. This behavior is social, autonomous, reactive and proactive. In order to model this kind of HMS behavior an interaction model is composed of abstract agents, roles, goals, tasks, interactions and interaction units. The abstract agents and the roles are the interactions actors. In an interaction, interaction units, in which an initiator (sender) cooperates with a set of collaborators (receivers), are executed. Moreover, the actors participate in interactions in order to fulfill their goals. The interactions specify behavior of the abstract agents. This behavior may be: an abstract agent's response to a given action on itself (reactive behavior), or; a function on the abstract agent's goals and tasks (proactive behavior).

The interaction model is very important and useful in ANEMONA. Firstly, it is a tool by which the system engineer may express: the dynamics of the modeled system, the communication among the different system entities, the coordination and behavior of these entities. Secondly, the system engineer can define interaction scenarios in which the participating entities (abstract agents) may have different complexity levels (agent, agent group, organization, multi-agent system). Thirdly, the interaction scenarios can be used as integration patterns of HMSs encapsulated into abstract agents. Fourthly, the abstract agent encapsulation allows us to abstract away from complex details in order to facilitate and focus attention on the interaction's context, nature and execution depending on the problem at hand. Fifthly, the interaction scenarios associated to an organization, constitute the abstract agent behavior specification that, in a higher abstraction level, represent that organization.

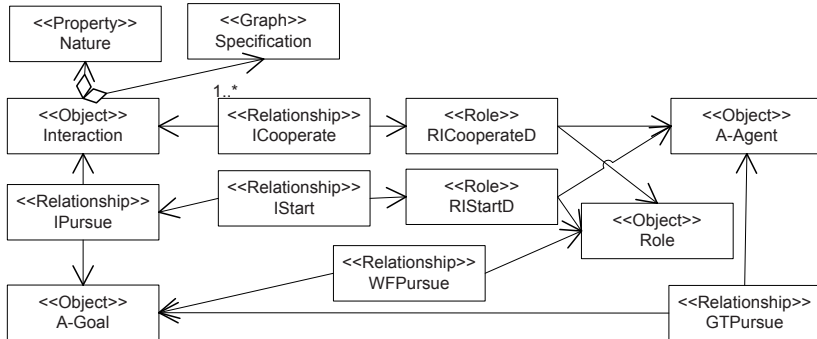


Fig. 5.24 Interaction metamodel: interactions, abstract agents, roles and goals

5.5.1 Interactions, Abstract Agents, Roles and Goals

The metamodel of Fig. 5.24 defines some of the entities, and relationships among them, allowed in an Interaction diagram. In this figure we can see that in order to specify an interaction the system engineer has to model the goals/abstract goals that the interaction pursues (*IPursue*), the participating entities (roles or abstract agents) by means of the *IStart* and *ICooperate* relations, and, at the same time, the goals that make these autonomous entities participate in the interaction (*WFPursue* and *GTPursue*). These relations are described in Sections 5.3 and 5.4).

The relation *IStart* states which role/abstract agent initiates the interaction. The motivation of this autonomous entity to do this is modeled by the relations *WFPursue* and *GTPursue*. The relation *ICooperate* specifies which roles/abstract agents cooperate in the interaction. These autonomous entities may also fulfill their goals participating in the interaction. This fact is modeled by the relations *WFPursue* and *GTPursue*. These relations are graphically represented, as depicted in Fig. 5.25.

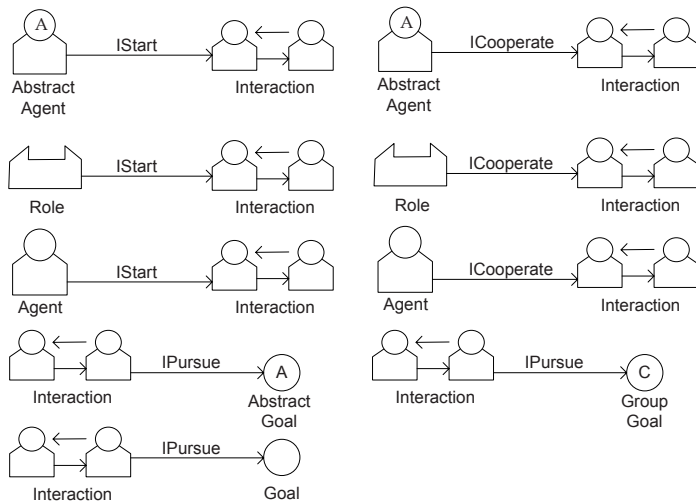


Fig. 5.25 *IStart*, *ICooperate*, and *IPursue* relations

Let’s imagine a simplified interaction on a factory floor for a *work-order scheduling* (Fig. 5.26). This interaction is initiated by a *work-order holon* (relation *IStart*) that is in charge of the execution of the work order, and has the goal “to minimize the work order starting time” and the goal “to get the work order produced on time”. These goals motivate the *work-order holon* to start the interaction (relation *GTPursue*). At the same time the interaction pursues the goal “to get the work order produced on-time” (relation *IPursue*). In order to fulfill the goals in this interaction the cooperation of the *machine controller* role, the *product manager* role and the *raw material* role (relation *ICooperate*) is needed. At the same time, these roles have their own goals that motivate them to cooperate in the interaction.

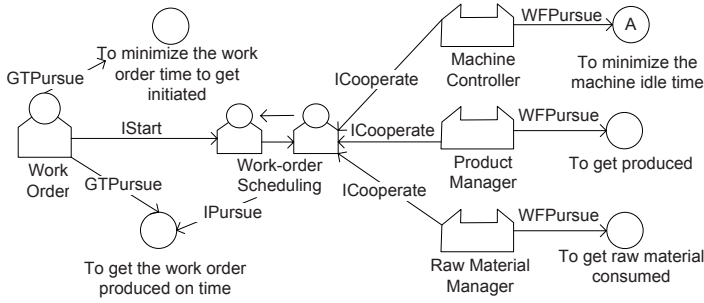


Fig. 5.26 A sample interaction for a simplified work-order scheduling scenario

5.5.2 Interactions, Interaction Units, Abstract Agents, Roles and Tasks

In order to communicate with each other, the participating entities of an interaction exchange interaction units. Figure 5.27 summarizes the constraints and relations used to model an interaction unit.

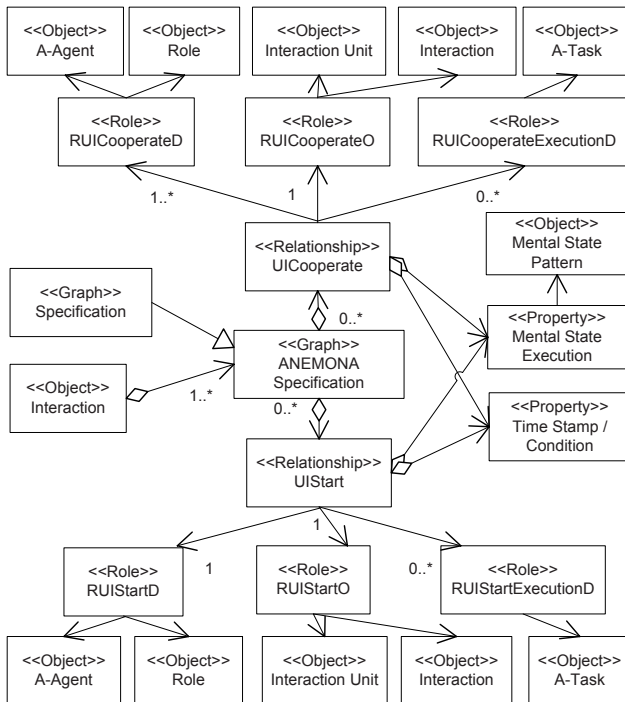


Fig. 5.27 Interaction metamodel: interactions, interaction units, abstract agents, roles and tasks

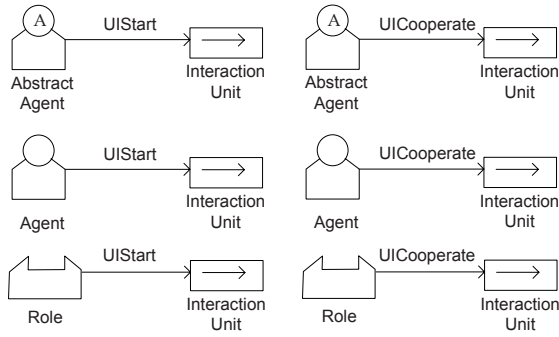


Fig. 5.28 UStart and UICooperate relations

An interaction unit can be a message-passing, or remote procedure call. Interaction units and tasks/abstract tasks are used to specify the execution of the interaction. An interaction is executed when the participating entities communicate with each other by means of interaction units and when the participating entities execute their related task/abstract tasks. Figure 5.28 illustrates the graphical representation of the relations *UStart* and *UICooperate*. These relations are used to identify the interaction unit sender and receiver entities. A sample interaction diagram is presented in Fig. 5.30.

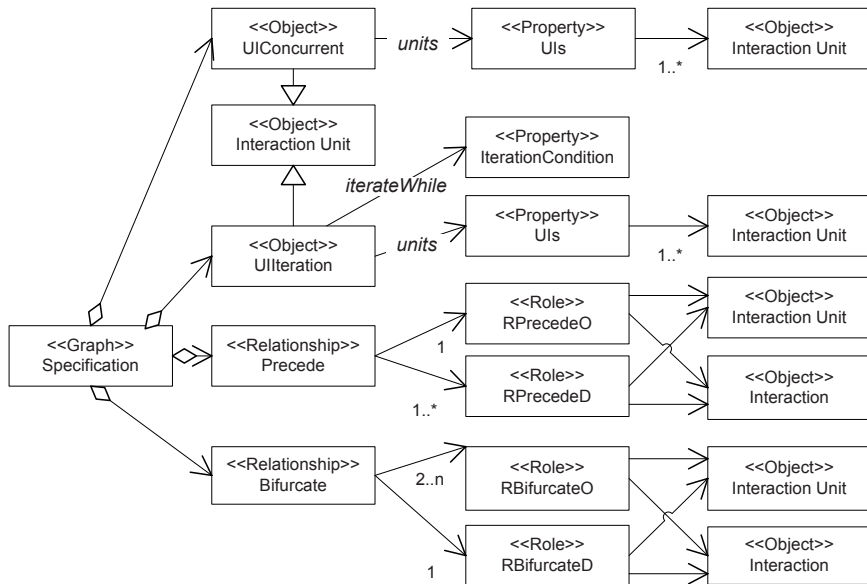


Fig. 5.29 Interaction metamodel: interaction specification

5.5.3 Interaction Specification

In order to complete the specification of an interaction, an order of execution of the set of interaction units that made up the interaction has to be defined. This is done as detailed in the metamodel definition of Fig. 5.29.

There are basically four types of relations for defining the interaction units' execution order. The relation *precede* defines interaction units' sequences. Let A and B_1, \dots, B_n be interaction units. A *Precede* relation among A and (B_1, \dots, B_n) implies that the interaction unit A is always executed before any B_i . The relation *bifurcate* is used to model precedence conditions among a group of interaction units. That is, a *Bifurcate* relation among A and (B_1, \dots, B_n) implies that only one B_i will be executed after A . Which B_i will be executed depends on the mental state of the interacting autonomous entities. The *UIIteration* defines the iterative execution of a set of interaction units. When the concurrent non-deterministic execution of a set of interaction units is needed, *concurrent* is used. A sample interaction diagram is presented in Fig. 5.30.

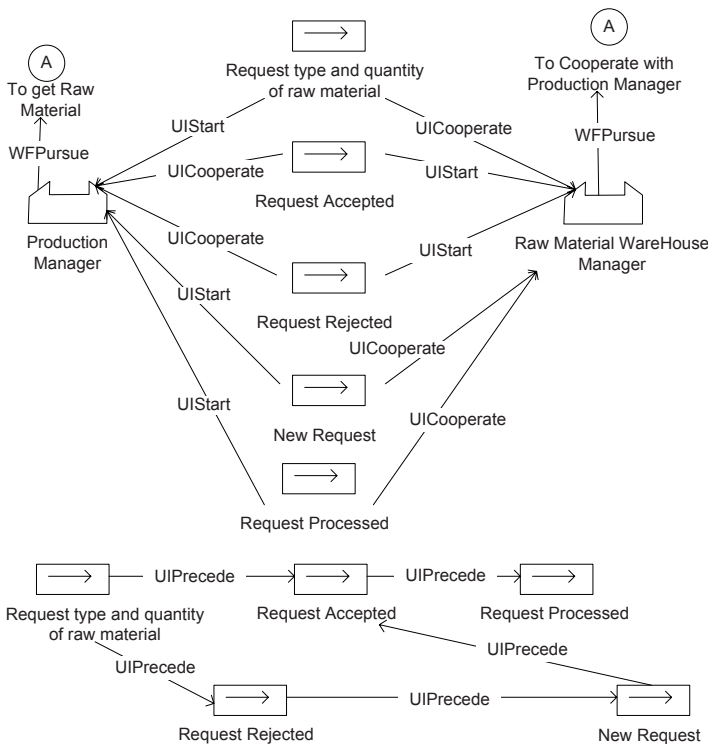


Fig. 5.30 A sample specification of the simplified interaction *raw material request*

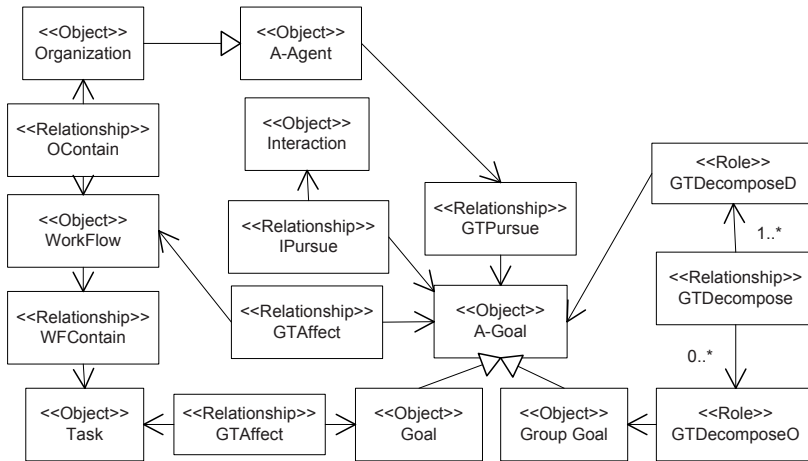


Fig. 5.31 Interaction metamodel: interactions and organizations

These relations are also used to concatenate interactions that have to be executed in a given order. Building complex interaction scenarios by composing simpler pre-defined interactions can be useful.

5.5.4 Interactions and Organizations

Interactions are executed within the context of organizations structures. In other words, autonomous entities populate organizations and participate in interactions in order to fulfill their own goals, as well as the organization’s goals. Figure 5.31 shows the metamodel specification of the relations in an organization and the interactions that are associated to its autonomous entities. Here we can see that the nexus is defined by the goals that are bound to the organizations and to the interactions.

5.6 Environment Model

The environment model is used to specify the environment entities with which the HMS autonomous entities may cooperate. It also specifies the constraints associated to these interactions. The environment model is defined by resources, applications and abstract agents, and focuses on the abstract agent’s perceptions and actions in its environment.

An environment resource is defined by the metamodel diagram of Fig. 5.32. A resource can be consumable or not consumable. A resource may belong to an abstract agent (relation *EResourceBelongsTo*). An abstract agent may use applications

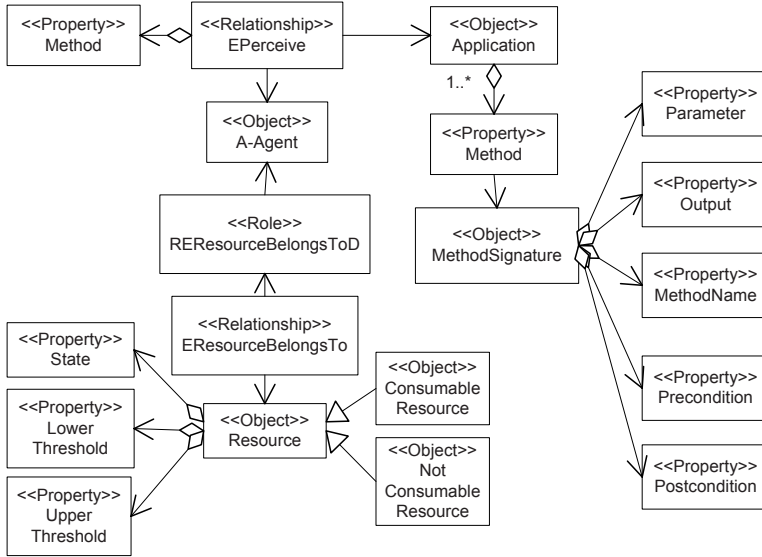


Fig. 5.32 Environment metamodel: resource

in order to fulfill its goals (relation *EPerceive*). In order to specify how an abstract agent uses an application the system engineer has to state the application methods used, including the method name, parameters, output, and pre- and postconditions. Figure 5.33 shows the graphical representation of these relations.

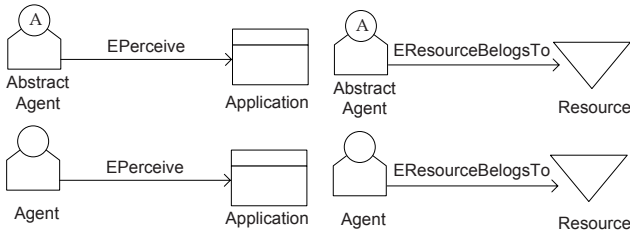


Fig. 5.33 EPerceive and EResourceBelongsToTo relations

5.7 Organization Model

The organization model is used to specify the architecture of the HMS. It is composed of a set of organization diagrams in which the organizational structures of the HMS are described. In an organization diagram the main elements are the work

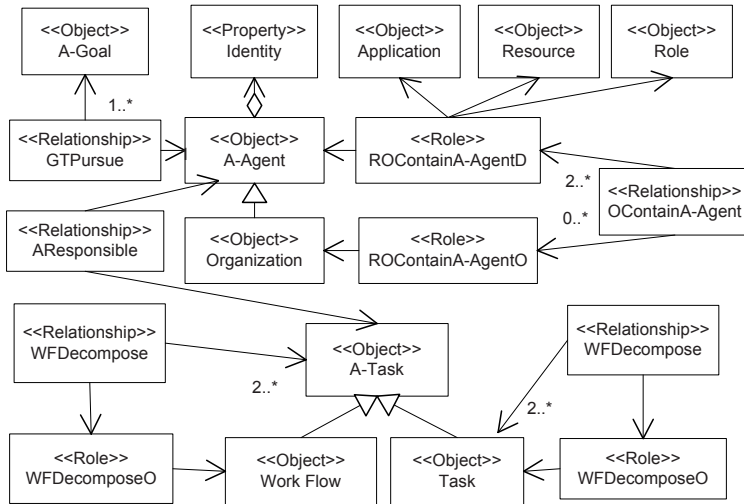


Fig. 5.34 Organization metamodel: organization structure

flows that detail how the organization’s members coordinate their actions in order to fulfill the system goals. The organization model also defines the constraints in the holons behavior by means of subordinate relationships.

The HMS structure is modeled specifying the following three complementary views. The organization structure defines the main elements that make up the organization and how it is built from them. High-level social relationships constrain its member’s behavior. The functional definition states what the organization offers to its users and how it is executed.

5.7.1 Organization Structure

Figure 5.34 shows the main entities and relationships with each other used to specify the structure of an organization. The fundamental relations are *OContainA-Agent* and *AResponsible*. The *OContainA-Agent* relation is used to specify the entities that make up the organization. An organization structure is made up of abstract agents, roles, resources and applications. These entities populate an organization in order to execute the work flows that fulfill the organization (system) goals, relation *GTPursue* (Sect. 5.4). These relations are graphically represented as depicted in Fig. 5.35. The relation *WFDecompose* is used to state what tasks/abstract tasks or in turn work flows define a given work flow.

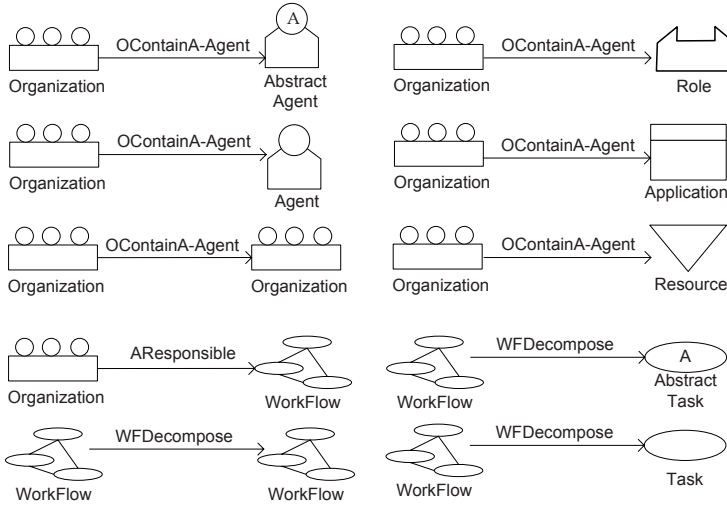


Fig. 5.35 OContainA-Agent, AResponsible and WFDecompose relations

5.7.2 Social Relations Among Autonomous Entities

The relations *AGOSubordination* and *AGOClientServer* (Fig. 5.36) are used to configure social constraints that limit the interactions among the autonomous entities of an organization. The entities that can be related to these social relations are organizations, roles, abstract agents and agents.

Relation *AGOSubordination* forces the subordinate to always obey its major entity (Fig. 5.37). At the same time the major entity can give any order to its subordinate. This relation can be conditional *AGOConditionalSubordination* or unconditional *AGONotConditionalSubordination*. The first one is used in order to specify relations in which the subordinate is always forced to, regardless of its actual goals. While the unconditional relation is used in cases in which there is a contract (condi-

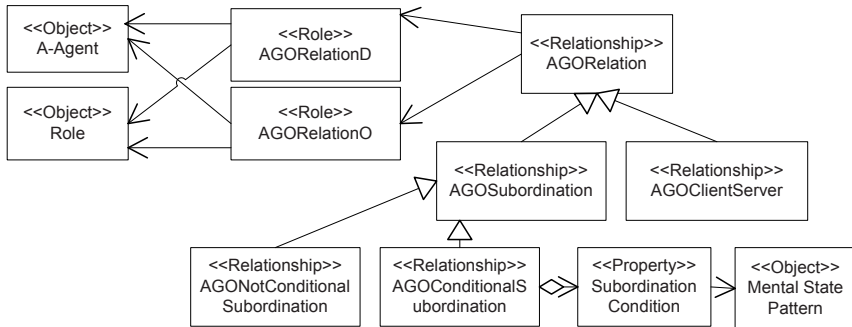


Fig. 5.36 Organization metamodel: social relations among autonomous entities

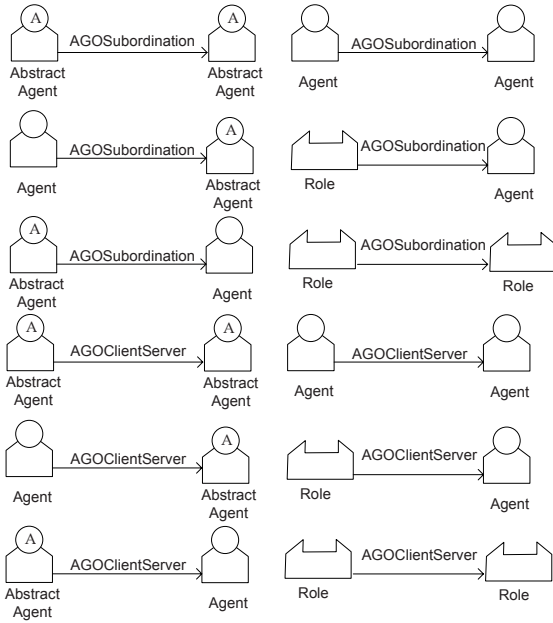


Fig. 5.37 AGOSubordination and AGOClientServer relations

tions). When the contract conditions are not valid then the subordinate is not forced to obey. The *AGOCientServer* relation (Fig. 5.37) is used to model relations in which there is a service provider (server) and a service requester (client).

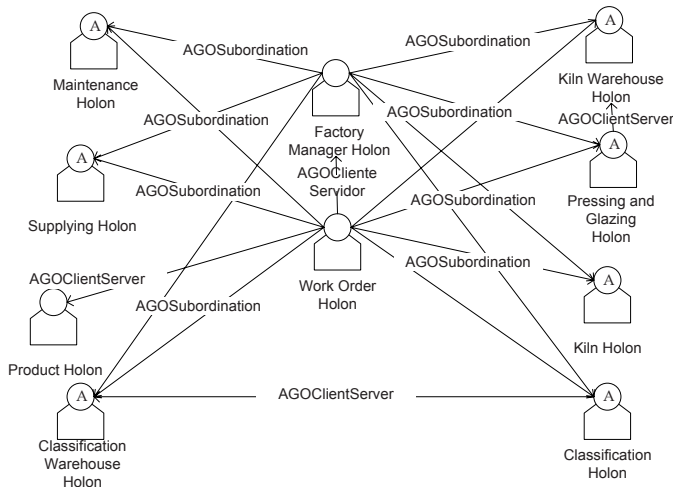


Fig. 5.38 A sample organization diagram of a factory holarchy

Figure 5.38 shows a sample organization diagram in which a *factory* holarchy is specified. In this holarchy there are different holons that offer services (client/server relations) and maintain subordination relations.

5.7.3 Organization Functional Definition

The organization functional definition is specified using work flows. A work flow specifies how resources are assigned, which steps (tasks/abstract tasks) are necessary to achieve a given goal (*GTPursue*), and which autonomous entities are responsible for executing them (*AResponsible* or *WFResponsible*). Figure 5.39 summarizes the metamodel specification of these relations.

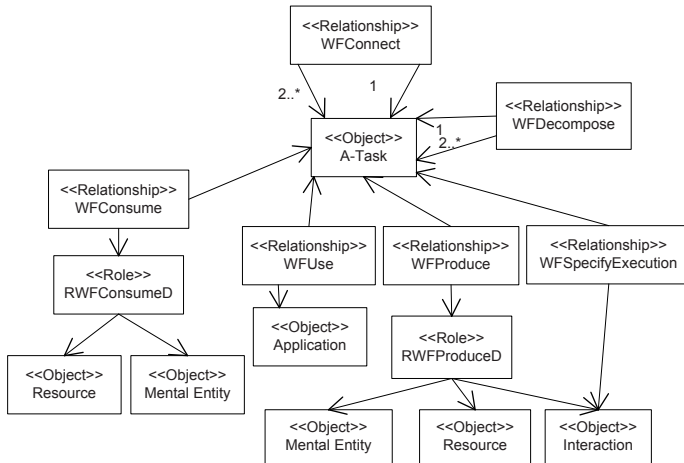


Fig. 5.39 Organization metamodel: work-flow specification

In order to concatenate tasks/abstract tasks and work flows the relation *WFConnect* can be used. The relations *WFConsume*, *WFUse* and *WFProduce* (Sect. 5.4) are used to interconnect the inputs and outputs of the work-flow tasks and the executing autonomous entities. The *WFConnect* relation is graphically represented as is shown in Fig. 5.40.

Figure 5.41 shows a sample organization diagram in which the work flow *holarchy formation* of a *pressing and glazing* organization is specified. In this figure we can see the different holons that participate in the work flow and the tasks they are responsible for. We can also see the task sequence that defines the work flow and the activation of interactions as a consequence of the execution of certain tasks.

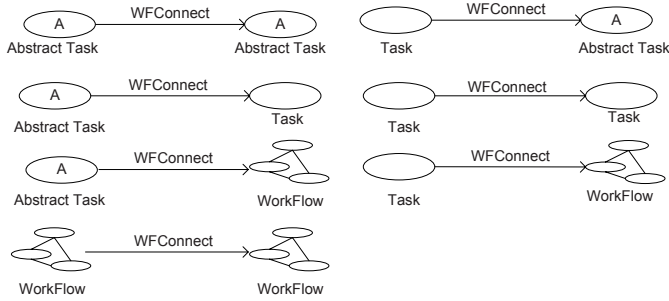


Fig. 5.40 WFCConnect relation

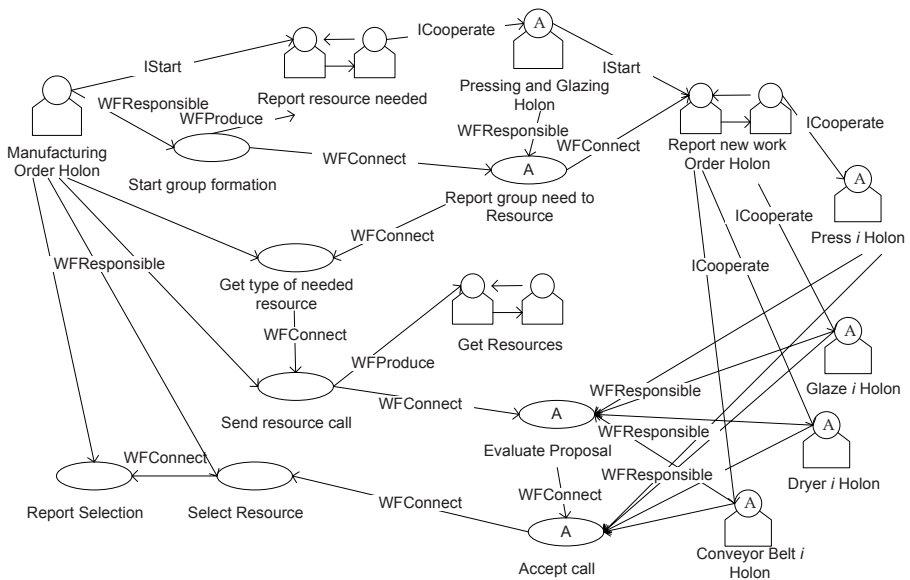


Fig. 5.41 A sample organization diagram for the work flow holarchy formation of a pressing and glazing organization

5.8 Conclusions

In this chapter we have presented the ANEMONA notation. Its central modeling entity is the abstract agent notion presented in Chap. 3. In ANEMONA the HMS is divided into different views of the system. These views make up the complete HMS analysis and design specification. These views are: The agent model which is concerned with the functionality of each abstract agent, responsibilities and capabilities; the organization model, which describes how system components (abstract

agents, roles, resources, and applications) are grouped together; the interaction model, which addresses the exchange of information or requests between abstract agents; the environment model that defines the non-autonomous entities with which the abstract agents interact; and the task/goal model, which describes the relationships among goals and tasks, goal structures, and task structures.

In Chap. 6 the ANEMONA development process is explained. The development process details how the different ANEMONA models are built and the step-by-step activities and tasks to develop the HMS.

Chapter 6

ANEMONA Development Process

The development process of a methodology includes a description of the subsequent development steps, the different activities the engineer has to do, the guidelines that guide the engineer in the process, and the products that have to be obtained from the process. In this chapter we describe ANEMONA in terms of these elements. In order to do so in a standardized fashion we will use the SPEM notation [134] for the ANEMONA development process and its components. A short overview of SPEM is presented in Sect. 6.1.

In the following sections we will use a simplified real-life case study from a supply chain scenario in order to illustrate the development process. This scenario is from an automotive, transport and service company that provides equipment parts to its dealers through a network of distribution centers. Section 6.2 presents the requirement specification of this case study.

The ANEMONA development process deals with the HMS modeling requirements by means of the abstract agent notion of Chap. 3, the notation presented in Chap. 5, and the development guidelines presented in this chapter.

6.1 SPEM











SPEM (*software process engineering metamodel*) is a notation used to define processes and their components [134]. This notation is based on an object-oriented approach to model a family of related software processes. SPEM provides a minimum set of process modeling elements in order to describe any type of software development process. The process elements of SPEM are described in terms of UML concepts [137]. A sub-set of UML is used to define SPEM. This sub-set is the metaobject facility (MOF) [139] that is the kernel of the definition.

SPEM defines a software development process as a collaborative process among active abstract entities called *process roles*. A *process role* executes *activities* on concrete and tangible entities called *work products*. Multiple *process roles* interact or cooperate interchanging *work products* and activating the execution of *activities*.

In other words, the *process roles* are the system engineer members of the development team. The *activities* are the development steps of the development process. And the *work products* are the different models that define the system architecture of the developed system.

Table 6.1 summarizes the main SPEM notation elements that are used in the following sections.

Table 6.1 SPEM notation elements

Element	Notation	Description
<i>Process</i>		Represents a complete process. A set of process descriptions that can be used to define other processes.
<i>Phase</i>		Represents a software development process phase. A specialization of <i>work definition</i> .
<i>Work definition</i>		Represents a set of tasks. A kind of operation (or complex task) that describes the work developed in a process.
<i>Work product</i>		Any produced, consumed or modified element of a process. It can be an information fragment, a <i>document</i> , a <i>model</i> , programming code, etc.
<i>Process role</i>		Defines the responsibilities in a given <i>work product</i> and the roles that execute and help in some <i>activities</i> .
<i>Activity</i>		The major sub-class of <i>work definition</i> . Describes the set of tasks, operations and actions that are executed by a <i>process role</i> .
<i>Document</i>		Represents a <i>process</i> -generated document.
<i>Guidance</i>		The guiding elements can be associated to any SPEM element. They are used to provide more detailed information on the associated element. Examples: guidelines, techniques, metrics, examples, patterns, etc.
<i>Model</i>		Represents the models used in the software development process. Examples: class model, agent model, conceptual model, dynamic model, organization model, interaction model, etc.
<i>Process package</i>		A container that contains and is used to import the process describing elements.

6.2 A Simplified Supply Chain Case Study

In this section we describe a simplified supply chain case study that is used to illustrate the ANEMONA development process. Modeling diagrams from this example

are included in the different development stages of the following sections. A complete case study is presented in Chap. 8.

An automotive parts supplier provides equipment parts to its dealers in Europe through a network of distribution centers. This company requires the automation of its supply chain process. A parts distribution system (PARDI) that will be operated by its delivery division is required. When a particular part is needed for service, the dealership will order it through PARDI and the part will be packed and shipped from a distribution center. Whenever possible, the shipment will be carried out by the company's dedicated delivery service. Otherwise, a third-part delivery service will be used. Requested parts not in stock at the closest distribution center are defined as referral parts. To better meet the specialized requirements of referral parts, PARDI has to search for ways to drive greater efficiencies and improve the delivery service of these part to its dealerships. PARDI will also automate the referral parts distribution process.

When a distribution center receives an order, it will send the referral part to an order-consolidation center. PARDI will sort the arriving parts by dealership. Multiple parts ordered throughout the day by a single dealership will be consolidated into one package. PARDI will process the continuous flow of orders, expand the daily window for ordering capability, and identify the most cost-effective method for delivery. Each order's tracking number will be scanned at key stages of the supply chain, and the data will be uploaded into the system's main database. Dealerships will log onto PARDI and have access to the status of their orders. The tracking data will also validate key performance metrics for the company, such as consolidation rates, order cycle times, and costs.

6.3 The Method

The ANEMONA development process is a mixed top-down and bottom-up process. It is an abstract-agent-oriented method for identifying and specifying holons from a manufacturing system specification. It offers clear and HMS-specific development guidelines and deals with the whole HMS life cycle.

Figure 6.1 illustrates the development phases of ANEMONA. The first phase *system requirement analysis* and the second phase *holon identification and specification* define the **analysis** stage of ANEMONA (Sect. 6.3.2). The goal of the **analysis** stage is to provide a HMS high-level specification from the manufacturing system *requirements* (Sect. 6.3.1). The **analysis** stage follows a top-down recursive approach. One advantage of such an analysis is that its work products, the *analysis models*, provide a set of elementary components and a set of composition rules from which a bottom-up system **design** can easily be defined (Sect. 6.3.3). The third phase, *holon design*, is a bottom-up process in order to produce the *system architecture*. The goal of the *holon implementation* phase (Sect. 6.3.4) is to produce the system *executable code*. This code is used in the *setup and configuration* phase

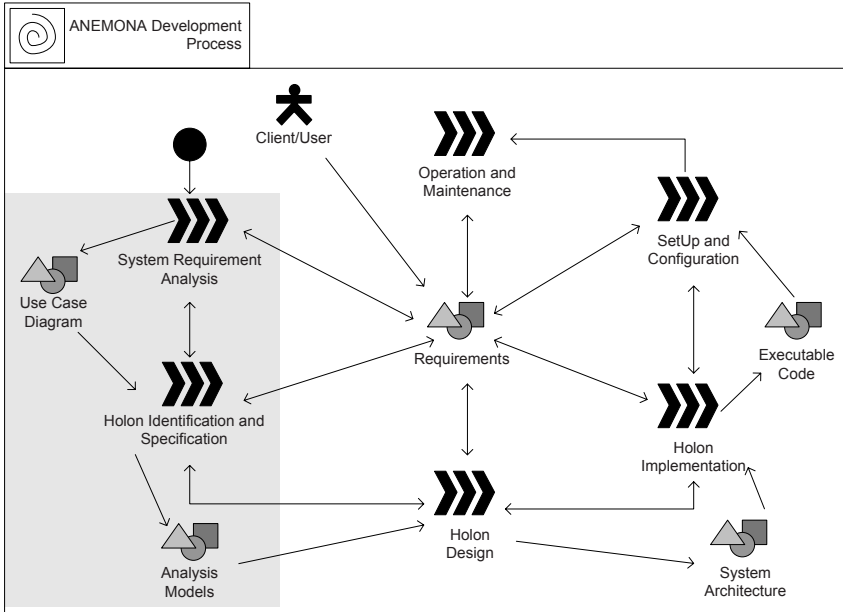


Fig. 6.1 The ANEMONA development process

(Sect. 6.3.5) to deploy the HMS. Finally, maintenance functions are carried out in the *operation and maintenance* phase (Sect. 6.3.6).

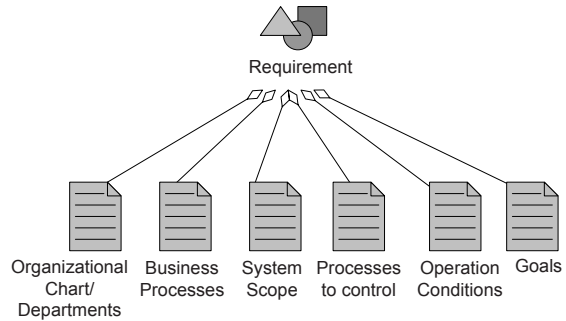
In the following sections we describe the definition of the ANEMONA development phases and illustrate their application via the simplified case study of Sect. 6.2. Chapter 8 shows the detailed development of a complete case study from a ceramic tile factory.

6.3.1 System Requirement

The *requirements* specification document describes the manufacturing system’s requirements and its associated control problems. This document is composed of the following parts (Fig. 6.2):

- *Organizational chart.* A description of the organizational chart of the company into which the system will be integrated. This description includes a functional description of every department in the organization, and the relations and interactions among them.
- *Business processes.* A description of the manufacturing company’s business processes [130].
- *System scope.* A description of the intended scope of the system in the company. This description includes the current processes which will be replaced, the new

Fig. 6.2 The requirement specification document



processes that will be added, and how these new processes should interact with the other manufacturing system components.

- *Processes to control*. A detailed description of the business processes that will be controlled by the system, including the available control interfaces [140, 141].
- *Operations conditions*. A specification of the operation conditions [140, 141].
- *Goals*. A specification of the system goals and the production requirements [140, 141].

These document parts could have any structure, but in order to get an organized *requirement* document we suggest using some semi-formal notations. The *organizational chart* and the *business processes* parts can be described using diagrams from IDEF0 [122], DFD [142] or UML Activity Diagrams [105]. The *system scope* requirement document is used to delimit the HMS in the manufacturing company under study. It is also used to specify the interactions among the new system and the other predefined components of the manufacturing company. These requirements may be specified using the diagrams of parts 1 and 2 (*organizational chart* and *business processes* parts) highlighting the processes that will be replaced and/or the new processes needed. The *processes to control* requirement document can be specified using natural language. When the controlling process is the production process, the description must include: the (physical) components of the production system and its placement on the factory floor [143]. Moreover, for every component its physical behavior and its controlling interface must be described. The physical behavior may be described using IDEF0 or Petri nets [144]. The controlling interface specifies what kind of information will be available and what kind of (physical) actions can be activated in the component. The *operations conditions* requirements describe the required inputs and outputs of the controlling processes, and the set of possible changes and failures that could happen during the HMS operation. The processes inputs are specified in terms of work orders or services that can be requested from the processes, and the required raw material to execute them. The output is specified by means of a list of the expected products/results. Finally, in the *goals* requirement specification, the system-performance requirements are described. Some sample production goals could be: high throughput, minimum waiting time, maximum capacity utilization, flexibility in the face of resources and work-order changes, re-

sponse capability in front of mechanical or controlling failures, scalability, quality assurance, maintainability, etc.

6.3.1.1 Example

Figure 6.3 shows a sample *business processes* diagram of the supply chain case study (the notation used is DFD). In this figure we can see the different processes executed by the company departments. Processes 4 and 5 are highlighted to indicate that they will be replaced by PARDI. Also apparent are the interactions that have to be maintained with the current company processes.

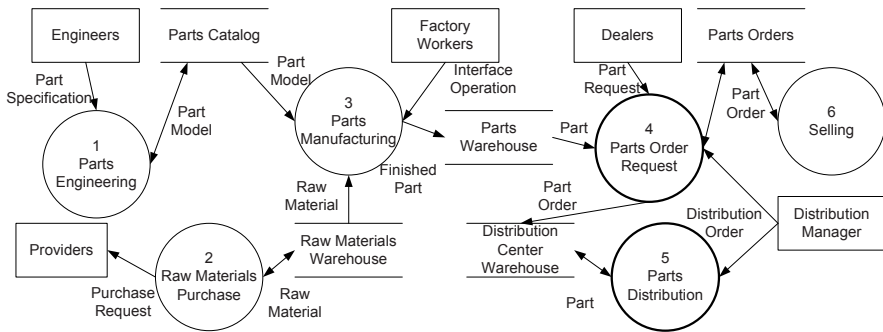


Fig. 6.3 A sample business processes diagram of the supply chain case study

From the case study description of Sect. 6.2 we can extract the system goals of Table 6.2 that configure a part of the *operations conditions* requirement document.

Table 6.2 PARDI goals

Goal
Manage the supply-chain process for distributing parts to dealers.
Integrate and automate the business processes 4 and 5 of Fig. 6.3.
Process the continuous flow of part orders.
Expand the daily window for ordering capability.
Identify the most cost-effective method for delivery.
Improve the delivery service of parts to the company dealerships.
Maximize delivery division efficiency.

6.3.2 Analysis

In the *analysis* stage the system engineer has to specify the HMS in terms of the ANEMONA models presented in Chap. 5 and *use case diagrams* from UML [115]. The *analysis* stage is a bottom-up, incremental and recursive process. The main goal of this stage is to identify the system component holons and to provide an initial specification of them. The system engineer has to produce the *analysis models* of the HMS from the system *requirement* document (Sect. 6.3.1).

Every iteration of the *Analysis* stage identifies and specifies holarchies of different abstraction levels (Chap. 3). This idea is illustrated in Fig. 6.4. At the highest level (level m) we can model the whole system as a single holon ($A-Agent_k$) that tries to achieve the system goals specified in the *requirements* document. In the first *analysis* iteration, the system engineer identifies a holarchy (level $m-1$) composed of a set of holons that implement the $A-Agent_k$. In this way, every holon in this holarchy represents an autonomous entity that is simpler than $A-Agent_k$, but one that, in cooperation with the other holarchy members, implements the functionality and behavior of the $A-Agent_k$. The *analysis models* of the first *Analysis* iteration specify the holarchy of level $m-1$. At the end of the first iteration, the system engineer must analyze each holon of level $m-1$ in order to figure out the advantages of decomposing it into a new holarchy at a lower abstraction level. For every holon that is decomposed an analysis process is started, in the next *analysis* iteration. In this process the specification of the holon produced in the previous iteration is taken as the sub-system requirements. In this way, each new iteration will have as many concurrent and collaborative processes as constituent holons of the previous iteration, which it was decided would be decomposed. In every iteration the *analysis models*

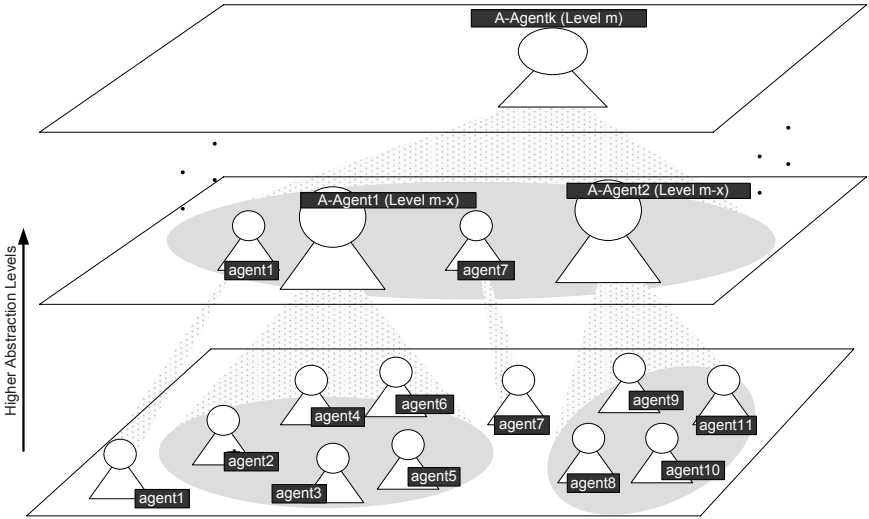


Fig. 6.4 Abstraction levels in the analysis stage

are completed in an incremental fashion. This process is repeated until every holon is completely defined and there is no need for further decompositions.

The analysis process guided by abstraction levels is based on the software engineering principles of abstraction and decomposition, and on the HMS recursive feature [4]. In this way, in every abstraction level the system engineer focuses only on the specification of the interactions and the relationships among the holons of the level without worrying about how these holons will be internally defined. The internal specification of these holons will be modeled in subsequent analysis iterations.

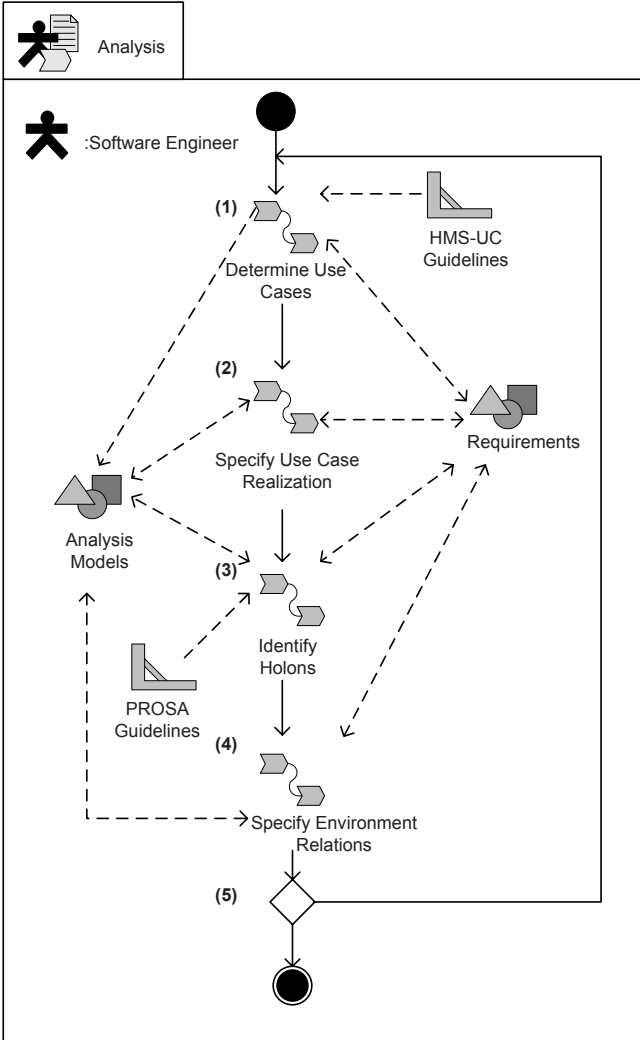


Fig. 6.5 The ANEMONA analysis stage.

In order to gain a better understanding of the *analysis* stage we describe a single analysis iteration. The activity diagram of Fig. 6.5 illustrates the set of complex tasks that are executed in one analysis iteration. It also shows the execution sequence and products of every complex task. The role that is responsible for the analysis stage is the *software engineer*. The goal of every iteration is to develop a set of models in which the holons that make up a given holarchy are identified and specified. This holarchy represents, in the first analysis iteration ($ni = 1$), the whole system, and in the subsequent iterations ($ni > 1$) the sub-systems that make up the whole system. The *analysis models* of the previous iteration are enhanced and refined in every new iteration. A single iteration definition is as follows.

1. *Determine use cases*. The goal of this complex task is to identify the system goals and the necessary cooperation domains [12] that satisfy these goals. In this step, the *software engineer* has to use the *HMS-UC guidelines* that will help them to identify the system use cases from the *HMS requirements* document. The product of this task is a *use cases model*.
2. *Specify use cases realization*. The goal of this complex task is to specify the interactions among the use cases and to define an initial set of abstract tasks (Sect. 5.4) that will implement them. In this step the *software engineer* makes use of *roles* to specify the autonomous entity that is in charge of the use case implementation. The products of this task are *organization model*, *interaction model* and *tasks and goals model*.
3. *Identify holons*. The goal of this complex task is to identify and specify the holons that make up the holarchy that is being analyzed in the current iteration. In this step the *software engineer* uses the *PROSA guidelines* that help to identify the system *abstract agents* and to assign the *roles*, of step 2, to these *abstract agents*. To this end, the *organization model*, *interaction model* and the *tasks and goals model* from step 2 are refined. Also, an *agent model* is generated for the identified holons.
4. *Specify environment relations*. The goal of this complex task is to identify and specify the non-autonomous environment entities that the holons have to work with. The product of this task is the *environment model*.
5. Finally, for every holon identified in the iteration, the *software engineer* has to analyze the advantages of decomposing it into a new holarchy. This decision is made based on the *requirements* document. If the decision is to decompose a new analysis process for every holon that will be decomposed is started in the next iteration. In the other case, the *analysis* stage is finished.

Figure 6.6 shows the different models that configure the product of the *analysis* stage. The specification details of these models are defined in Chap. 5.

In the following sections every complex task of the *analysis* stage is defined in terms of activities, execution sequences, and documents and models that are produced or modified by every task in the ANEMONA analysis process.

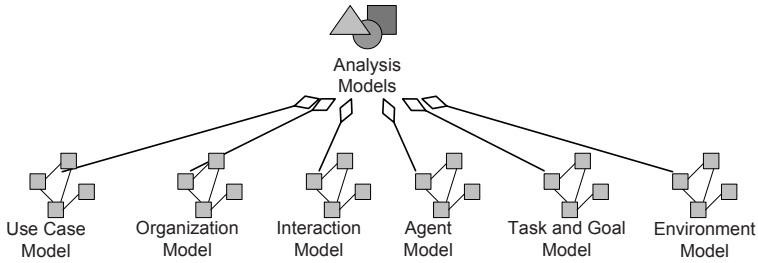


Fig. 6.6 The analysis models

6.3.2.1 Determine Use Cases

The first step in the *analysis* stage (see Fig. 6.5) is to *determine use cases* by building a *use case model* from the *system requirements*. This complex task is defined as a

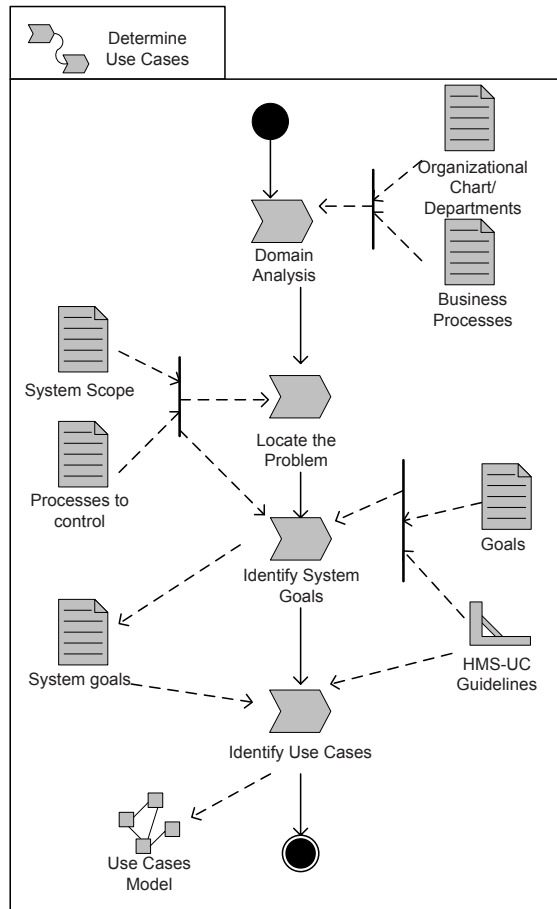



Fig. 6.7 Determine use cases task definition

sequence of four activities. This sequence is outlined in Fig. 6.7. The activities are defined as follows.

- *Activity A1: Domain analysis.* In this activity the *software engineer* has to comprehend the general features of the company in which the HMS will be deployed. To do this the engineer has to study the *organizational chart* and the *business processes* parts of the *requirement* document. This activity is executed in the first iteration, and has to be applied in subsequent iterations ($ni > 1$) only in the cases in which there are changes in the requirements (changes in the company, new business processes, etc.).
- *Activity A2: Locate the problem.* In this activity the *software engineer* has to place the HMS into the company. In other words, the engineer has to study the interfaces of the HMS under development with all the business processes of the company that are left outside the system. This is done by analyzing the *system scope* specification and the *processes to control* part of the *requirement* document. In a $ni > 1$ iteration the *software engineer* has to focus the analysis on the functionalities/business processes that are implemented by the abstract agent that is being analyzed in the iteration.
- *Activity A3: Identify system goals.* In this activity the *software engineer* has to identify the goals that have to be fulfilled by the HMS. In order to do this, the engineer has to study the *goals* specification, the *system scope* part and the *processes to control* of the *requirement* document. In this activity the *software engineer* makes use of the *HMS-UC guidelines* in order to identify the HMS goals. The engineer has to answer questions 1 to 6 of the *HMS-UC guidelines* (Table 6.3). In a $ni > 1$ iteration, the focus is on the models produced in the previous iteration for the abstract agent that is being decomposed.
A list of the goals identified for the PARDI case study is presented in the next section.
- *Activity A4: Identify use cases.* In this activity the *software engineer* has to identify the Use Cases (cooperation domains) that will fulfill the goals identified in the *Activity A3*. The engineer has to follow the guidelines 7 to 13 of the *ANEMONA HMS-UC guidelines* (Table 6.3) and has to build a *use case model* [121]. In a $ni > 1$ iteration, the use cases, which were assigned in the previous iteration to the abstract agent that is being decomposed, have to be analyzed using the *HMS-UC guidelines* in order to decompose them into simpler use cases, if necessary.
A sample *use case model* of the PARDI case study is presented in the next section. A complete example of the products obtained from this activity in various iterations can be found in Chap. 8.

Table 6.3 ANEMONA HMS-UC guidelines

 HMS-UC guidelines		
ID	Guideline	Comment
1	Is the system going to produce something? If yes, one goal of the system will be to produce it.	When the process is the production process, this guideline refers to the factory products. When the process is of another type, the product refers to those that have to be generated as a result of the process execution.
2	Is there a process to control? If so, a system goal will be to control it.	
3	Is there an external resource to work with? If so, a system goal will be to work with it.	
4	Can the system receive a work order? If so, a system goal will be to process the work order and to control its execution process.	A work order refers to a production request or an execution request of any HMS business process.
5	Do the system goals need to be controlled or managed? If so, a system goal will be to manage them.	This goal is related to the global vision or the staff holon management of the PROSA architecture.
6	Is it necessary to communicate the system products to an external business process? If so, a system goal will be to communicate the products and to manage the communication.	
7	Define a use case for every goal derived from question 1.	
8	Define a use case for each goal related to the execution control of a machine.	This use case will probably be translated into a controlling process of a holon over a machine. In the first iterations these types of use cases may not be identified.
9	Define a use case for every goal related to the control of a process.	
10	For each communication goal with external resources define a use case to manage the communication.	
11	For each goal related to a work order define a use case to implement the domain cooperation to process it.	
12	Define a use case for every goal derived from question 6.	
13	Define a use case to manage every communication process with external business processes.	

6.3.2.2 Example

In this section we present the products of *activities A1* to *A4* in the analysis of the PARDI case study (Sect. 6.2). Recall the *requirement* fragments of Sect. 6.3.1.1. From questions 1 to 6 of the *HMS-UC guidelines* (Table 6.3) we have derived the system goals of Table 6.4.

Table 6.4 PARDI system goals

ID	Goal
1	Manage the supply-chain process for distributing parts to dealers.
2	Manage the flow of orders.
3	Automate the distribution process.
4	Manage referral parts.
5	Automate and manage the package tracking.
6	Contract third-party delivery services that better meet the specialized requirements of referral parts and greater improve the delivery service.

Figure 6.8 shows the use cases produced from *activity A4*. These use cases have been derived by applying the guidelines 7 to 13 of Table 6.3 and taking into account the goals defined in Table 6.4. The use case *manage referral parts distribution* is defined in order to satisfy goals 1 and 4. Goals 1, 2 and 3 are considered in the use case *manage orders*. *Deliver packages* is defined in order to fulfill goal 3. Goal 6 is used in the use case *contract third-party delivery service*. Finally the *track packages* use case deals with goal 5.

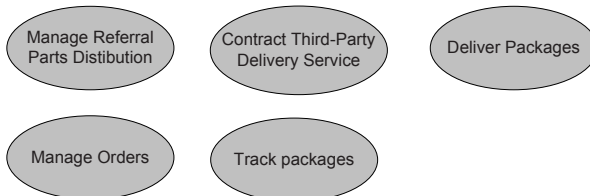


Fig. 6.8 Use cases diagram of PARDI case study

6.3.2.3 Specify Use Case Realization

The second step in the *analysis* stage (see Fig. 6.5) is the complex task *specify use case realization*. This task is decomposed into the four activities in Fig. 6.9. These activities are defined as follows.

- *Activity A5: Assign a manager to every use case.* In this activity the *software engineer* has to identify a provider/responsible *role* for every use case. An au-

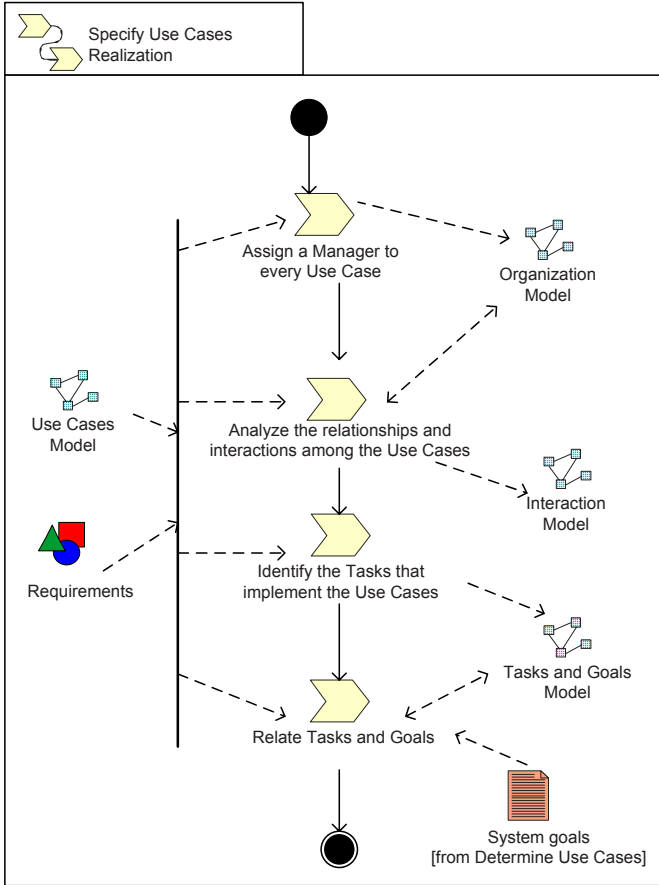


Fig. 6.9 Specify use case realization task definition

tonomous entity will be in charge of providing the service that the use case supplies to the system users. The autonomous entity is modeled using *roles*. In this way, at this abstraction level, it is not necessary to specify if the use case will be implemented by a set of autonomous entities or a single one, or the way in which the service will be implemented. These modeling decisions are postponed to future steps of the *analysis* stage when the internal system structures are clearer. With the identified use case providers an initial *organization model* is built. This model includes a set of *roles* that implement the use cases of the current iteration. This model will be extended in subsequent activities and analysis iterations. A sample *organization diagram* derived from this activity is presented in the next section.

- *Activity A6: Analyze the relationships and interactions among the use cases.* In this activity the *software engineer* has to specify the relationships and interac-

tions among the use cases. The interactions and relations among the use cases are very important because if the use cases are analyzed in an isolated way many important characteristics for controlling the global system are left out [81]. The services, and their associated actions, are not executed in isolation, but in a system in which their effects may affect other services in the system. In order to identify these type of dependencies, it is important to analyze and to model the interactions and relationships that may appear among them. They are identified by means of communications needs among the *roles* that are in charge of the use cases. These interactions could be a coordination need, a planning dependency, a negotiation or a cooperation. These requirements are specified in the *interaction model*. Also, the social relations (Chap. 5) among the *roles* have to be identified, such as client/server and subordination. These relations are modeled refining the *organization model* of activity A5.

- *Activity A7: Identify the tasks that implement the use cases.* In this activity the *software engineer* has to identify abstract tasks (Chap. 5). These abstract tasks will be executed by the *role* in order to implement the services of the use case the role is in charge of. In subsequent iterations these abstract tasks will be decomposed in order to transform them into tasks (atomic) or work flows. The abstract tasks are identified and associated to the *roles*. These relations are modeled in the *organization model*. The tasks are specified in the *tasks and goals model*. In a $ni > 1$ iteration, the abstract tasks, which are identified in this activity, represent the tasks decomposition of the abstract agent that is being analyzed in the iteration.
- *Activity A8: Relate tasks and goals.* In this activity the *software engineer* has to associate the abstract tasks with the goals that are affected by them. An initial list of goals is derived from *activity A3*. The engineer has to work with this list and the previously identified abstract tasks in order to complete the *tasks and goals model* with the relations identified among the tasks and goals. These relations may be *GTAffect*, *GTSatisfy*, *GTDestroy*, *GTCreat*e and *GTFail* (Chap. 5). In this activity the *software engineer* can identify new goals related to the interaction among the *roles*. Answering the following questions new goals can be identified: Is there any reason for initiating the interactions among the *roles*? Is there any condition in the interaction execution? Does the interaction need some management? These goals are specified in the *tasks and goals model* and are associated to the *roles* that have to fulfill them.

6.3.2.4 Example

Figure 6.10 shows an *organization diagram* for the PARDI case study. In this diagram the roles in charge of every use case are depicted. Also, the social relations among these roles are modeled. On the other hand, Fig. 6.11 shows the interactions among these roles. These diagrams are the results of *activities A5* and *A6*.

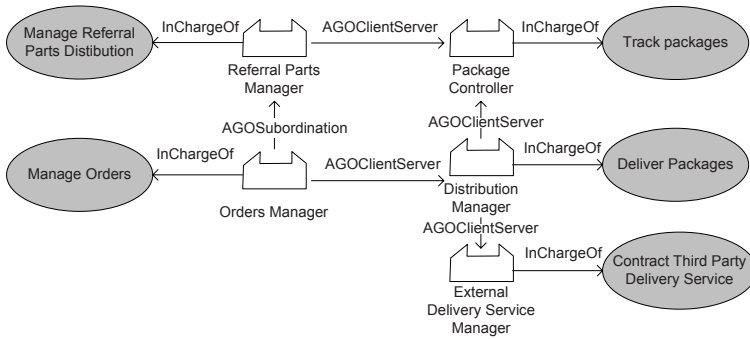


Fig. 6.10 An organization diagram of the PARDI case study

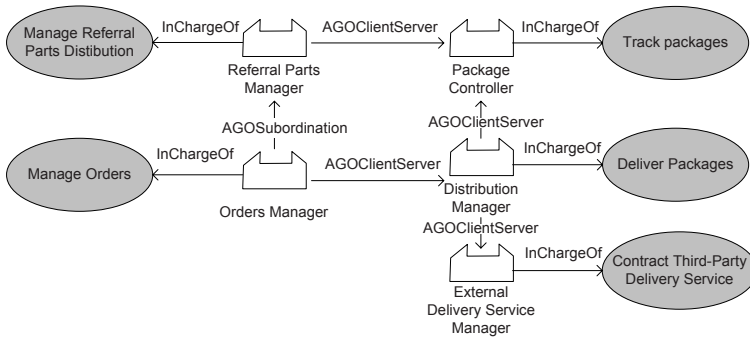


Fig. 6.11 An interaction diagram of the PARDI case study

Figure 6.12 shows some abstract tasks that implement some use cases of PARDI. These tasks were identified applying *Activity A7*. Analyzing these tasks and relating them to the system goals of Table 6.4 we have derived from *Activity A8* the relations depicted in Fig. 6.12.

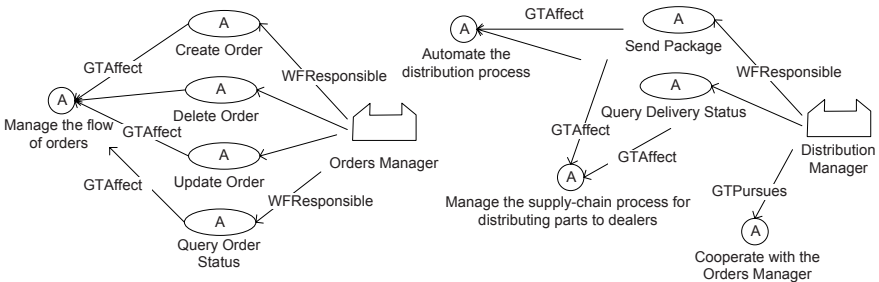


Fig. 6.12 A tasks and goals diagram of the PARDI case study

6.3.2.5 Identify Holons

The complex task *identify holons* is the task in which the *software engineer* has to spend more time in each iteration. For this task there are two approaches that can be used for identifying holons. The physical decomposition (the more obvious approach) and functional decomposition. In the first approach agents are used to represent the physical entities of the problem. These entities may be workers, machines, tools, products, parts, operations, etc. The physical approach defines different sets of state variables that can be managed in an efficient way by individual agents with a limited number of interactions. Nevertheless, this approach involves a large number of agents related to resources. A common example of this approach is the use of part agents and machine agents for manufacturing planning and scheduling [48, 82, 83]. In the functional decomposition approach, the agents are used to encapsulate functions such as work-order reception, planning, scheduling, material handling, transport management, product distribution, etc. In this approach the agents are not explicitly related to the physical entities. Examples of this approach are: the use of agents to encapsulate special functions (for example, facilitator agents [84], broker agents [85], and mediator agents [86]) and the use of agents to integrate pre-existing systems (for example, ARCHON [87], EXPORT [88] and CIIMPLEX [85]).

ANEMONA defines a set of guidelines in order to help the *software engineer* in the holon identification and specification process. These guidelines are the *PROSA guidelines* defined in Tables 6.5, 6.6 and 6.7 and are based on the PROSA type of holons [23]. PROSA architecture integrates the physical and functional decomposition approaches presented in the previous paragraph. PROSA architecture specifies that there are four types of holons in any manufacturing system: work-order holon, product holon, resource holon and staff holon. Based on these types of holons ANEMONA proposes 28 guidelines.

The activities and the flow that define the complex task *identify holons* are presented in Fig. 6.13. These activities are defined as follows.

- *Activity A9: Find out the types of holons.* In this activity the *software engineer* has to identify the abstract agents, that is, the *roles* identified in prior activities are assigned to holons. To do this the engineer has to analyze the *requirements* document and the *analysis models* of the previous steps, with the help of the *PROSA guidelines* (Tables 6.5, 6.6 and 6.7). The engineer has to refine the *organization model* assigning *Roles* to the abstract agents that are identified. In this activity the *software engineer* has to use the *PROSA guidelines* 1 to 13 and 22 to 25 in order to identify abstract agents.
- *Activity A10: Specify holons.* In this activity the *software engineer* has to build an *agent diagram* for every holon identified in the *activity A9*. The engineer has to complete the abstract agent definition from the goals and tasks of the *role* or *roles* associated with. New tasks or task decompositions may be identified. Moreover, intelligent and autonomous features that can help the abstract agent to fulfill its goals can be specified. In order to complete the identification of the abstract tasks

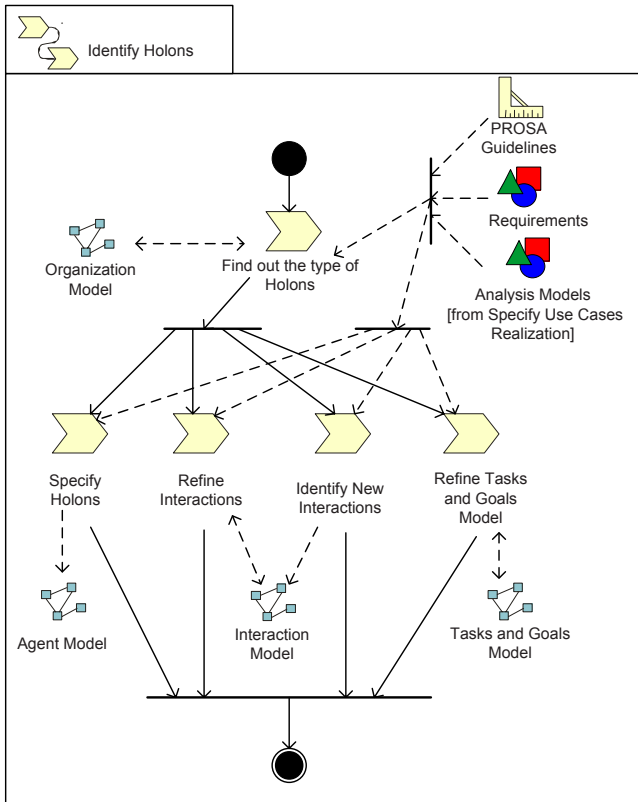


Fig. 6.13 Identify holons task definition

that the abstract agent is responsible for, the *software engineer* has to use *PROSA guidelines* 14 to 17 and 25. On the other hand, *PROSA guidelines* 18 to 21 and 25 have to be used in order to identify the *information mental entities* (Chap. 5). These guidelines define the set of basic and required abstract agent information data, functions and services. The *software engineer* has to add to this set the information entities and tasks that model the particular features of the system that is being modeled. Also, real-time constraints associated to mental entities and tasks may be analyzed and identified (if required). When the holon represents a physical entity (machine, tool machine, etc.), the engineer may specify (if required) the temporal or real-time constraints associated with the time in which a modification on the mental entity occurs (perceptions or events that modify the information state of the agent, properties *event type* and *period* of the *event* entity, for more details see Chap. 5) and the time in which the actions associated must be executed (properties *task type*, *max deadline* and *period* of the *A-Task* entity, for more details see Chap. 5).

- *Activity A11: Refine interactions.* In this activity the *software engineer* has to complete the definition of the *interaction model* of the holarchy that is being analyzed. That is, interaction types have to be stated, interaction goals have to be identified, interchanged messages have to be specified, as well as the sender and receivers of every message, and the temporal constraints (based on the *requirements* document) associated with the interactions.
- *Activity A12: Identify new interactions.* In this activity the *software engineer* has to identify the interactions (not identified yet) related to the particular characteristics of the PROSA architecture. To this end the engineer has to use *PROSA guidelines* 26 to 28. Also, interactions related to the guideline 25 have to be modeled, and the definition of these new interactions has to be completed. That is, the goals of the interactions, the interchanged messages, the senders and receivers, and, if required, the temporal constraints associated with them.
- *Activity A13: Refine tasks and goals models.* In this activity the *software engineer* has to refine the *tasks and goals model* from the previous step. To do this the engineer has to use *PROSA guidelines* 14 to 17 and 25. The tasks have to be associated to goals. The dependencies among the goals have to be stated. The tasks and goals may be decomposed. The tasks have to be associated with the interactions in which they are involved. The mental entities of the abstract agents have to be associated with the tasks that consume them. Time constraints associated with the task execution may be identified. Moreover, in every iteration, all of the abstract tasks and the abstract goals associated with an abstract agent of the higher level have to be assigned to at least one holon of the holarchy that is being modeled.

Table 6.5 ANEMONA PROSA guidelines - Part 1

 PROSA guidelines	
ID	Guideline
1	There are four types of abstract agents in a holarchy: resource, product, work order and staff. The following guidelines define their characteristics and how to identify them.
2	The product abstract agent represents “things” that can be produced by the holarchy. If the holarchy represents to the whole factory, the product abstract agents refer to the items of the company’s product catalog.
3	The work-order abstract agent represents the holarchy’s requested tasks. This abstract agent is in charge of activating the resource abstract agents in order to start production. If the holarchy represents to the whole factory, the work-order abstract agent models the items included in the factory order book.
4	The resource abstract agent represents a system element with processing capacity. If the holarchy represents the whole factory, the resource abstract agents represent the departments, shops, machines, employees, etc.
5	The staff abstract agent represents the entity in charge of managing the interactions among the different abstract agents.
6	Each production facility (factory, department, shop, machine, conveyor, pipeline, component, tool, tool holder, personnel, etc.) and the information processing that controls it, is modeled as an abstract agent. For every production facility of the holarchy that is being analyzed, there has to be an abstract agent that models it. The <i>roles</i> (that were identified in the previous step) for controlling and managing the production facility have to be assigned to this abstract agent. This abstract agent is a resource holon.
7	A resource abstract agent offers production capacity and services to the abstract agents with which it cooperates. It stores the methods with which to assign production resources, knowledge and the processes with which to organize them, plus it uses and controls the production resource. Every <i>role</i> in charge of these activities has to be assigned to a resource abstract agent.
8	Each product definition or recipe is modeled as an abstract agent. Every <i>role</i> in charge of managing updated information about a product, its user requirements, design, process plans, the material list to produce the product and its quality assurance procedures, has to be assigned to a product abstract agent. A product abstract agent stores the “product model” of one type of product, not the “production status” of a particular instance of the product.
9	A product abstract agent stores the knowledge and production process for the correct manufacturing of the product, with the required quality defined by the <i>requirement</i> document. It acts as an information server for the work-order abstract agents of the holarchy. The product abstract agent models the functionalities related to the product design, process planning, and quality assurance.

Table 6.6 ANEMONA PROSA guidelines - Part 2



 PROSA guidelines	
ID	Guideline
10	Every task in a manufacturing system (a client work order, a business process execution work order, a warehouse work order, a maintenance and resource repairing work order, etc.) is modeled as an abstract agent. Every <i>role</i> in charge of managing a production process or business process; managing the production status of a product requested in a work order; and storing all the logistics information related to the task, has to be assigned to a work-order abstract agent.
11	A work-order abstract agent may be considered a controlling piece, which is in charge of managing the work order in the manufacturing system. In other words, the work-order abstract agent negotiates with other entities and resources in order to be produced.
12	The goals and/or the <i>roles</i> identified in <i>activity A8</i> related to the management of interactions among the <i>roles</i> of the holarchy, have to be assigned to a staff abstract agent.
13	A staff abstract agent is in charge of assisting, with expert knowledge, the other types of agents, and of implementing, if required, a centralized control in the holarchy.
14	A resource abstract agent is able to initiate processing tasks over the products. That is, it is authorized to accept or reject, depending on its goals and those of the holarchy, a task assigned to it.
15	A resource abstract agent controls and monitors the execution of its processes (suspend, resume, abort), manages its sub-resources and plans its tasks.
16	A product abstract agent implements the design tasks of the product, process planning, and product quality verification.
17	A work-order abstract agent implements the scheduling, deadlock management, and order process monitoring, and triggers activation, pausing, resuming, aborting, and stopping process events into a resource.
18	A resource abstract agent stores beliefs about its abilities (the product list that it can manufacture), its execution tasks, its sub-resources, and a registry of its activities.
19	A product abstract agent stores information data about the process plan, product description and product quality requirements.
20	A work-order abstract agent stores information data about the work-order product status, and the task progress.
21	A staff abstract agent maintains information data on the interactions it is controlling. It could also store some kind of expert knowledge (plans, heuristics, etc.) in order to assist other abstract agents. A staff abstract agent can be used to encapsulate systems such as CAD, MRP, etc.

Table 6.7 ANEMONA PROSA guidelines - Part 3

 PROSA guidelines	
ID	Guideline
22	The following are types of work-order abstract agents: stock maintenance order; rapid orders, with due date shorter than normal; first-off order (this type of order requires more operation time, to possibly repeat failed processes, human supervision and intervention); client orders; maintenance orders.
23	Types of product abstract agents: variations on a product, a new version of one product, spare parts, mass production, high-quality product, etc.
24	Types of resource abstract agents. This group includes the different information resources, machines, tools, etc., that can be found in a manufacturing system. The personnel and the workers can also be modeled as resource abstract agents.
25	When a holarchy is being analyzed in a $n + 1$ iteration, the interface of the abstract agent that models it in the n level (previous level) can be implemented in two ways. (i) Assigning the management of this interface to a staff abstract agent. That is, every communication to/from outside the holarchy has to go through the staff agent. (ii) Distributing the interaction management to all the abstract agents in the holarchy. The last option could be very complex to implement and more difficult to maintain, nevertheless, it is more flexible than the first option.
26	In order to trigger a new work order in a holarchy the following steps are required: (i) the staff abstract agent, or the agent in charge of the management of the holarchy, determines if the holarchy is capable of processing the order; if so (ii) the staff abstract agent activates a work-order abstract agent responsible for processing it, and acknowledge it to the client; finally (iii) when the work order is completed, the staff abstract agent informs the client.
27	In order to execute the requested tasks of a work order the following steps are required: (i) the work-order abstract agent requests a process plan from the product abstract agents; (ii) the work-order abstract agent starts a negotiation process with the resource abstract agents in order to assign tasks (of the process plan) to resources; finally (iii) the abstract agent initiates the task execution asking the resource abstract agents to start the process.
28	When a new resource abstract agent is added to a holarchy, its presence is communicated to the other abstract agents of the holarchy. As a response to this event, the product abstract agents have to verify the utility of the new resource to their process plans.

6.3.2.6 Example

Applying the complex task *identify holons* to the PARDI case study we have identified the system holons of Fig. 6.14. This figure shows an *organization diagram* extended with the abstract agents identified in *activity A9* using the *PROSA guidelines*. The PARDI organization is made up of 8 abstract agents (in the first analysis iteration). These abstract agents will be decomposed if required in subsequent iterations.

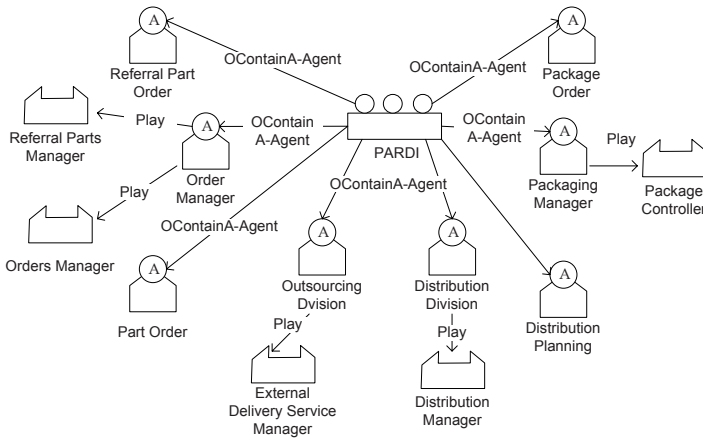


Fig. 6.14 An organization diagram of the PARDI case study

In order to specify an initial *agent model* of the system we have applied *activity A10* in order to assign the tasks and goals to the abstract agents. Figure 6.15 shows the *agent diagram* of the *order manager* abstract agent. To specify the ab-

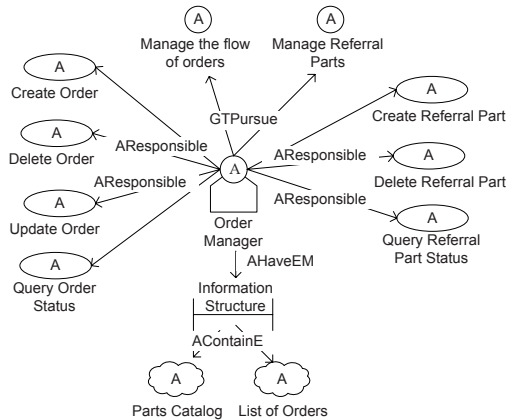


Fig. 6.15 An agent diagram of the order manager abstract agent

stract agent’s responsibilities we have applied *PROSA guidelines* 14 to 17 and 25, while for specifying its mental entities we have used the *PROSA guidelines* 18 to 21 and 25.

Figure 6.16 is a sample *interaction diagram* refined in *activity A11*. In this figure the *interaction units* interchanged among the *order manager* abstract agent and the *distribution division* abstract agent are depicted. The goals that motivate the execution of this interaction are also specified.

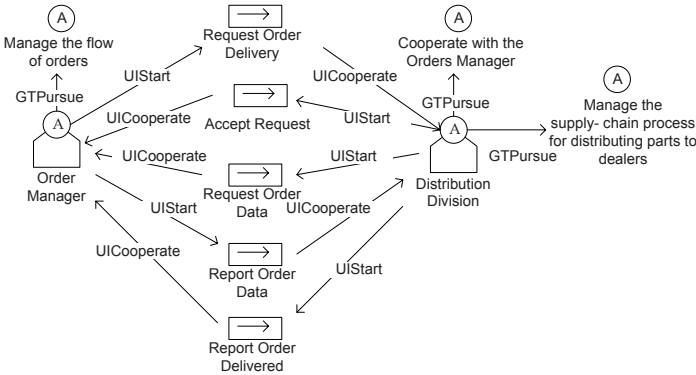


Fig. 6.16 An Interaction Diagram of the Request Delivery Interaction scenario

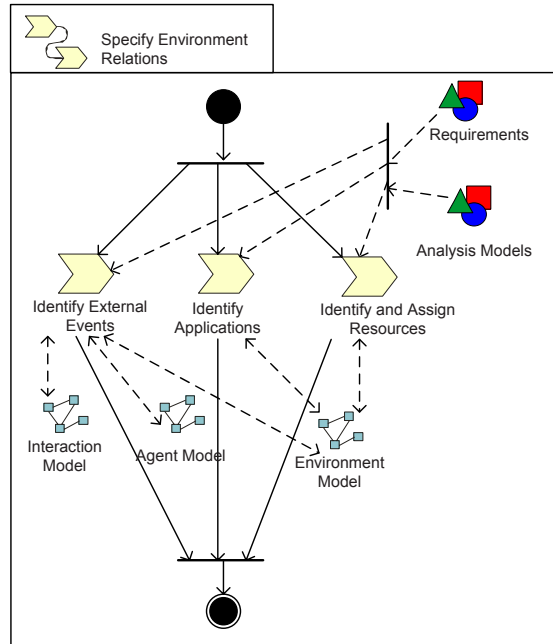
A complete example in which all of the products of the *identify holons* complex task are analyzed can be found in Chap. 8.

6.3.2.7 Specify Environment Relations

In the *specify environment relation* complex task the *software engineer* has to model the relations among the holons and their environment. These relations are derived from the *requirements* document and the study of the *analysis models* defined in the previous analysis activities. This complex task is made up of the 3 activities depicted in Fig. 6.17. These activities are defined as follows.

- **Activity A14: Identify external events.** In this activity the *software engineer* has to find out the list of external events that may in some way affect each holon. Also, the temporal constraints on the execution of these events have to be identified. In this activity the engineer has to build the *environment model* with events and their temporal features (if required) such as: occurrence pattern (periodic or aperiodic) and the minimum time in which there will be no event occurrence. In a $ni > 1$ iteration, the events list, associated with the abstract agent that is being decomposed, has to be distributed into events that affect its component holons. For every event identified, the holon’s reaction has to be specified. In order to do this, the engineer has to complete the *agent model* with the effect of the event

Fig. 6.17 Specify environment relations task definition



on the tasks and/or on the mental state of the abstract agent. Also, the *interaction model* has to be completed with the possible activation of an interaction in response to an event.

- *Activity A15: Identify applications.* In this activity the *software engineer* has to identify the list of non-autonomous applications with which the holons may work. A non-autonomous application is every type of software or hardware with which the system has to interact and cannot be modeled as a holon (the application does not fulfill the autonomous and cooperation features [4]). The applications are identified from the *requirements* document and the *analysis models* of the previous iterations. In this activity, the engineer has to complete the *environment model* with the identified applications. In a $ni > 1$, the applications list associated with the abstract agent that is being decomposed, has to be distributed into applications with which its component holons work. Every identified application has to be associated with the abstract agent that works with it by means of the *EPerceive* relation (Chap. 5).
- *Activity A16: Identify and assign resources.* In this activity the *software engineer* has to identify the list of resources with which the holons may work. A resource, in the *environment model*, is every resource of the manufacturing system that does not have an information processing part [4]. Resources are identified from the *requirements* document and the *analysis models* of the previous iterations. In this activity the engineer has to complete the *environment model* with the iden-

tified resources. In an $ni > 1$ iteration, the resources list, associated with the abstract agent that is being decomposed, has to be distributed into resources with which its component holons work. Every identified resource has to be associated with the abstract agent that makes use of it by means of the *EResourcePertain* relation (Chap. 5).

6.3.2.8 Example

Figure 6.18 shows an *environment diagram* of the *orders manager* abstract agent. This diagram was produced from *activities A14 to A16*. In this we can see a *dealers DB* application with which the *orders manager* works. It is a database application already, in use in the company, which will be reused by the PARDI system. The *new part request from client* is the event that perceives the *orders manager* and initiates the flow of a new order in the system.

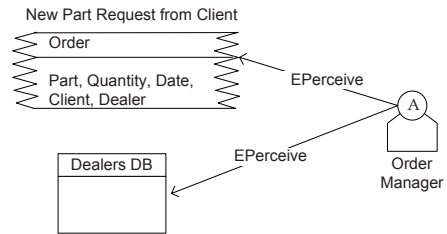


Fig. 6.18 An environment diagram of the orders manager abstract agent

When the five models are specified following the *analysis* steps described above, the *software engineer* has to decide if it is necessary to apply a new analysis iteration. To this end, the engineer has to analyze every abstract agent of the iteration in order to figure out whether it is convenient to decompose some of them. For every abstract agent that should be decomposed, a concurrent analysis process has to be applied in the next iteration. Each of these processes may be conducted in a collaborative fashion by different engineering teams.¹ The requirement specification for every concurrent process is defined by the different *analysis models* of the previous iteration that specify the given abstract agent. In this way, the integration rules are defined by the incremental analysis steps and models developed by every engineering team responsible for the different analysis processes. When there is no need for further decompositions the engineers can proceed to the following stage in the development process (*holon design*).

¹ One team may be decomposed into new teams in order to develop the collaborative analysis processes of the next iteration. The way in which a team is decomposed in sub-teams may be done following abstract agent complexity features, engineers' experience in related manufacturing processes, etc.

6.3.3 Design

In the *design* stage, the *software engineer* has to build the *system architecture* taking into account the details of the target implementation platform. The design stage basically involves the translation of the *analysis models* into a set of design models which define the system implementation details or requirements. The ANEMONA *design* stage is divided into two steps that are depicted in Fig. 6.19 and defined as follows.

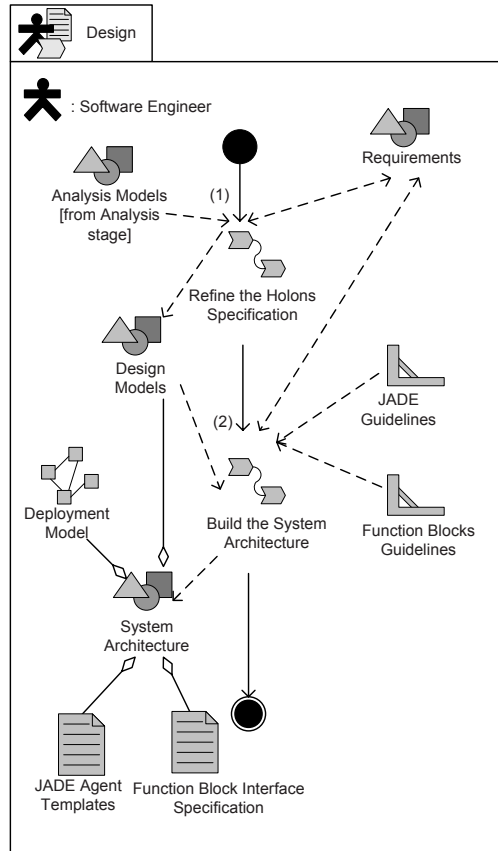
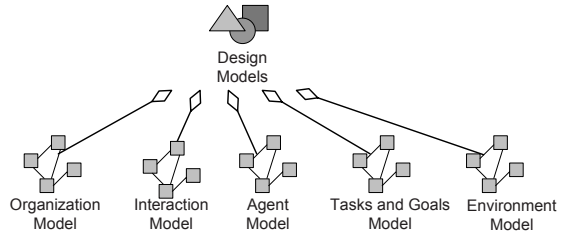


Fig. 6.19 The ANEMONA design stage

1. *Refine the holon specification.* The goal of this complex task is to complete the *analysis models* to guarantee that every system requirement is modeled. This phase is a bottom-up and platform-independent process. In this phase the *software engineer* focuses on every atomic holon in order to complete its definition. The product of this task is the *design model* presented in Fig. 6.20.

Fig. 6.20 Design models

2. *Build the system architecture.* The goal of this complex task is to build the *system architecture*. Our approach is based on the proposal of Christensen [9] for HMS implementation. For the high-level control (intraholon information processing and interholon cooperation) we use JADE (a FIPA-compliant agent platform) [145], while for resource holon low-level control (physical operations) we use functional blocks (IEC 61499 standard [14, 15]). The agent part (high-level control) negotiates and coordinates its tasks participating in interactions (cooperation domains). The physical part (optional) implements the physical processing by means of IEC 61499 applications. In this way, the *software engineer* can use the *JADE guidelines* (Tables 6.9 and 6.10) and the *functional blocks guidelines* (Table 6.11) in order to derive the *JADE agent templates* (Fig. 6.26) and the *functional block interface specification* (Fig. 6.27).

In this phase the *software engineer* has to associate the ANEMONA modeling entities with the implementation entities of the target implementation platform in order to define the system architecture. In the HMS field there are two types of processing: physical and informational [9]. There can also be temporal features, which may be soft or hard constraints. In a manufacturing system the real-time features and constraints are usually related to the resource physical functioning [9]. The IEC 61499 standard defines a platform for the real-time control of devices and real-time communication among the controllers [14, 15]. Taking these facts into account, the design of real-time requirements in ANEMONA is as follows. When the manufacturing system requirements impose hard real-time constraints they will be delegated to the functional blocks. On the other hand, when there are soft real-time constraints they can be implemented in the behavioral part of JADE agents.

In the following sections the complex tasks that make up the *design* stage are defined. These tasks are defined in terms of the activities, execution sequences, and the documents and models that are produced and/or modified in them.

6.3.3.1 Refine the Holon Specification

This complex task is a bottom-up iterative process that is based on the *analysis models* of the *analysis* stage. The key elements in this process are the holons, their recursion level (Chap. 3) and their composition rules. Figure 6.21 shows the concept. The first process iteration consists of designing the set of atomic agents (holons of

the recursion level $n = 0$) identified in the *analysis* stage. The *software engineer* has to specify the design details of these agents, building the corresponding design models. In the second iteration the engineer has to focus on designing the abstract agents of recursion level $n = 1$. To do this the engineer has to refine the *organization model* and the *interaction model* to design the organizational structure of the abstract agent and the interactions among its members (holons of the recursion level $n = 0$). This bottom-up process is repeated until there is no higher level in the (global) holarchy defined by the *analysis models*.

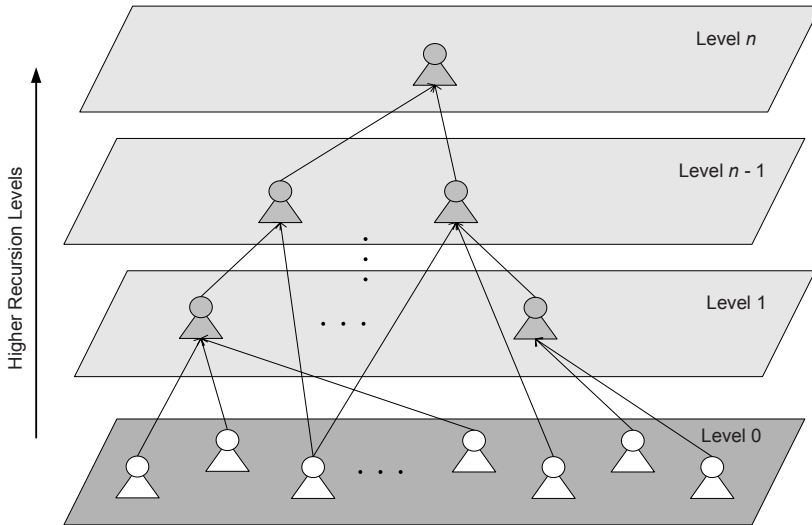


Fig. 6.21 Recursion levels in the design stage

A single iteration of this complex task is made up of 5 activities (Fig. 6.22) that are defined as follows.

- *Activity D1: Complete agent model.* In the first iteration, this activity requires the *software engineer* to focus on every atomic holon (identified in the *analysis* stage) in order to complete its definition based on the *Requirements* document and the *analysis models*. That is, the intermediate states that the agents go through have to be specified (*mental states*, Chap. 5); how an agent goes from one state to another should also be specified; the mental states have to be associated to the task executions. In the subsequent iterations, this activity requires the *software engineer* to verify that the responsibilities and capabilities associated to the holarchy abstract agents are included in the corresponding responsibilities and capabilities of the abstract agent that is being designed in the current iteration. That is, the composition of the abstract tasks (of the abstract agent) is based on the abstract tasks of its member autonomous entities (*WFDecompose* relation, Chap. 5); the composition of the goals is in terms of the goals of its member holons (*GTDecompose* relation, Chap. 5); the mental structure is defined

from the mental structure of its member holons (*AContainC* relation, Chap. 5). In other words, the *software engineer* has to complete the design of the abstract agent interface in terms of its constituent holons.

- *Activity D2: Complete tasks and goals model.* In this activity the *software engineer* has to ensure that every goal of an abstract agent has a corresponding task that satisfies it (*GTSatisfy* relation, Chap. 5) and also has to specify the dependencies among the goals (*GTDepend* relation). For all of the modeled tasks, pre- and postconditions have to be identified, as well as temporal constraints (if required). The tasks have to be associated with the interactions they initiate (*WFProduce* relation), with the mental entities they affect (*GTModify*, *GTDestroy* and *GTCreat* relations), and with the application operations (*WFUse* relation).
- *Activity D3: Complete environment model.* In the first iteration this activity requires the *software engineer* to focus on designing the environment relations of the atomic holons. To this end, the *EPerceive* relation (Chap. 5) has to be refined into *EPerceiveNotification* and *EPerceiveSample* relations in order to specify the

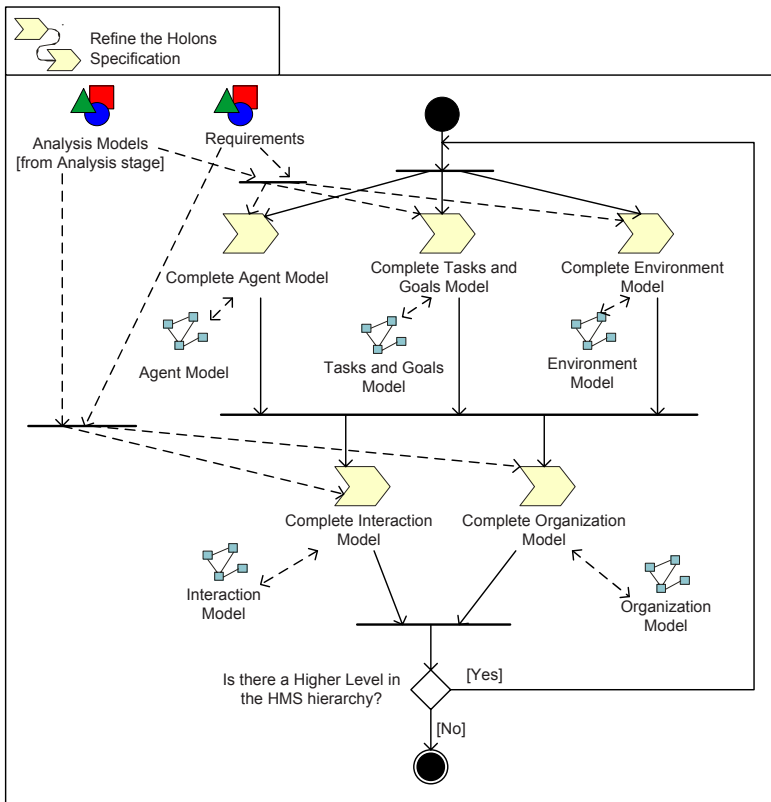


Fig. 6.22 Refine the holon specification task definition

type of perception. The environment resource attributes have to be specified as well. The resources, applications and tasks have to be related. The events definition has to be completed, paying special attention to refining the temporal features: pattern of event arrival and minimum time between arrivals. In the subsequent iterations the *software engineer* has to ensure that the abstract agent perception is specified in terms of its holon perceptions. That is, there is no resource, event or application of a given recursion level without a corresponding perception in the nearest lower level.

- *Activity D4: Complete interaction model.* In the first iteration this activity is not applied. In the subsequent iterations the goal of this activity is to design the set of interactions that are executed by the member holons of the abstract agent. These interactions are motivated by the goals of the abstract agent that models the holarchy in which they are executed. To this end, the *software engineer* has to focus on the abstract agent constituent holons and their interactions, in order to: specify the messages in terms of interaction units; specify an execution order among these interaction units; specify mental state conditions for the interaction unit's executions, and; specify the temporal constraints (if required) on the interaction unit's executions.
- *Activity D5: Complete organization model.* In the first iteration this activity is not applied. In subsequent iterations the goal of this activity is to design the social dependencies among the members of the abstract agent that is being designed. To this end, the *software engineer* has to refine the dependencies among the constituent members by mean of subordination and client–server relations.

Once every holon of a given recursion level is completely specified, the *software engineer* must move up to the nearest recursion level in the holarchy structure, i.e., to the cooperation domain in which the given holon interacts. The bottom-up process described above must be repeated until there is no higher cooperation domain in the *analysis models*.

6.3.3.2 Example

In this section let's work with the *track packages* use case (Fig. 6.10) and its corresponding holarchy. This holarchy has to package parts and track packages. Let's assume that the constituent holons of this holarchy are the ones listed in Table 6.8.

Figure 6.23 shows the *agent diagram* of the *package order* holon obtained from *activity D1*. In this diagram we can see the different *mental states* of the *package order* holon at different execution times. The entity *autonomous entity query* (Chap. 5) is used to represent an execution instance of the holon. The mental states in this diagram represent the information data and goals that motivate the *package order* holon to execute tasks.

Table 6.8 Holons of the PARDI track packages holarchy

Holon	Description
<i>Packaging manager</i>	Is the staff holon in charge of managing the track packages co-operation domain.
<i>Package order</i>	Is a work-order holon. It models a particular package order.
<i>Package</i>	Is a product holon that stores the knowledge about a particular type of package that can be packaged.
<i>Tracking sensor</i>	Is a resource holon. It models the different tracking sensor machines used in the tracking system.
<i>WareHouse</i>	Is a resource holon, representing the packaging division warehouse.
<i>Packager robot</i>	Is a resource holon. It models a packaging machine.
<i>Package transporter</i>	Is a resource holon. It models an AGV for transporting packages from one point of the shop floor to another.
<i>Conveyor belt</i>	Is a resource holon. It models the conveyor belt of the shop floor.

Figure 6.24 shows the *interaction diagram* for processing a package order in the *track packages* holarchy. This diagram is obtained from *activity D4*. A complete case study and all its associated design models are presented in Chap. 8.

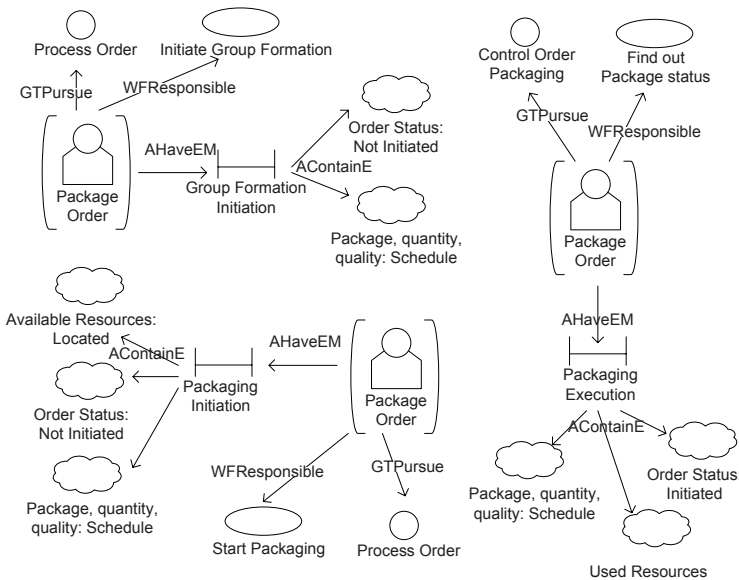


Fig. 6.23 Agent diagram of the package order holon

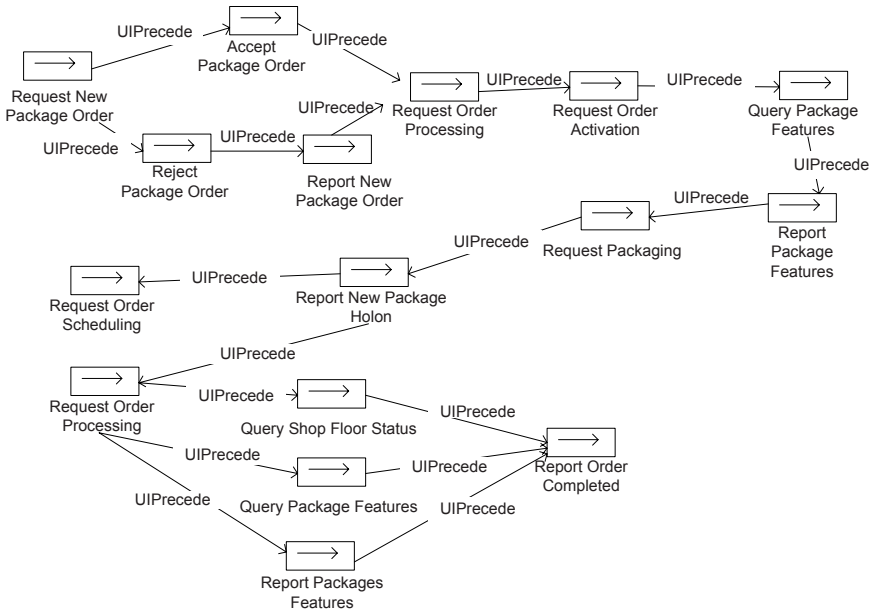


Fig. 6.24 Interaction diagram for processing a package order

6.3.3.3 Build the System Architecture

The last design step is *build system architecture*. The goal of this complex task is to complete the *system architecture*. In this phase the *software engineer* has to specify the target implementation platform details. ANEMONA provides two templates for specifying the platform design features. These templates are: *JADE agent template* and *function block interface specification*. In this phase the *software engineer* has to focus only on the atomic holons of the *design models* of the previous phase. This is so because the abstract agents are modeling notions (they are not implemented, see Chap. 3) and their “functionalities” are implemented by means of the agents of recursion level $n = 0$.

This task is made up of the four activities depicted in Fig. 6.25. These activities are defined as follows.

- *Activity D6: Find out distribution.* In this activity the *software engineer* has to find out the number of agent platforms required in order to distribute the system. Depending on the characteristics of the HMS, the engineer has to find out if the system will be deployed into a single-agent platform, or a set of them. In order to find out this number the engineer has to use *JADE guidelines* 1 to 4 (see Table 6.9). The product of this activity is the list of *identified platforms*, in which a name, number and the agent list that will be implemented, are specified.

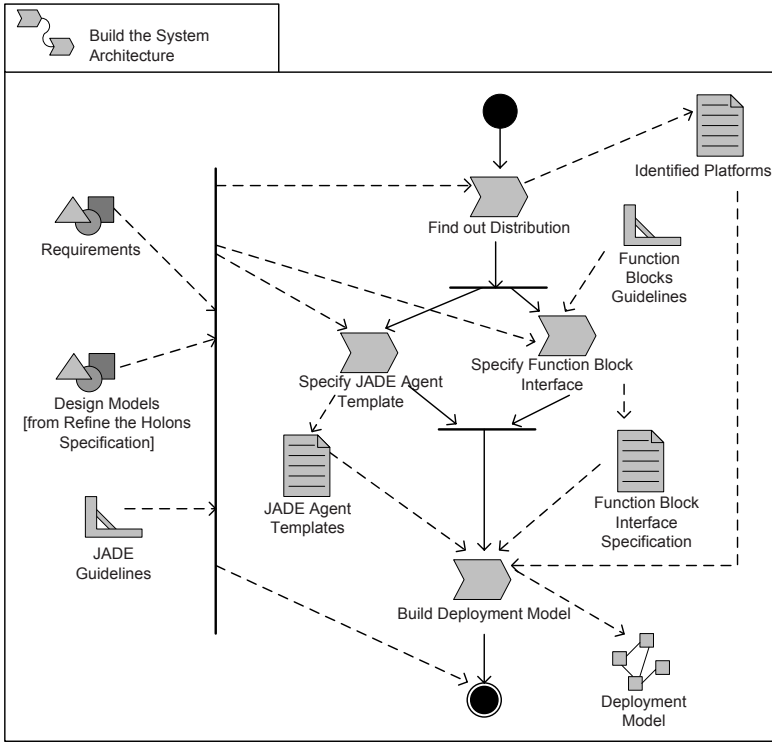


Fig. 6.25 Build the system architecture task definition

- *Activity D7: Specify JADE agent template.* In this activity the *software engineer* has to complete a *JADE agent template* (Fig. 6.26) for each agent in the *design models*. The *JADE agent template* includes specific characteristics of the JADE agent platform [145, 146] such as: agent identifier, behavior, service, communication, ontology, etc. In order to fill in this template the *software engineer* has to use the *JADE guidelines* 5 to 12 (Tables 6.9 and 6.10).

Table 6.9 ANEMONA JADE guidelines - Part 1



 JADE guidelines	
ID	Guideline
1	If the HMS represents a manufacturing system in which there is some kind of distribution (that is, departments, factory floors geographically distributed, or different branches), the control of each of these holons may be managed by a different agent platform. The interactions among the agents that populate these platforms will be managed by means of interplatform communication.
2	If the HMS represents a manufacturing system in which there is no physical distribution, the <i>software engineer</i> has to analyze the abstract agents of the <i>design model</i> . If it is a system in which there are highly related cooperation domains, the abstract agents participate in several cooperation domains, and; the number of agents is not too high, then the <i>software engineer</i> can consider implementing the whole HMS into a single-agent platform.
3	If the HMS has no physical distribution, the <i>software engineer</i> has to analyze the abstract agents identified in the <i>design models</i> . If there is a large number of agents, the engineer can consider the possibility of distributing the HMS in different agent platforms. In order to figure out the number of these platforms, they have to analyze the relationships and dependencies among the cooperation domains, security and data-encapsulation requirements, functionality, internal interactions, etc. The engineer has to group the cooperation domains by means of these criteria. The number of resulting groups has to be used as an indication to the number of agent platforms that will be identified.
4	In guidelines 1, 2 and 3 the <i>software engineer</i> also has to take into account the software engineering principle of low coupling and high cohesion, and apply it to the study of the identified platforms.
5	For the information processing part of every agent the <i>JADE agent template</i> of Fig. 6.26 has to be completed following guidelines 6 to 12.
6	Item 1 of the <i>JADE agent template</i> has to be filled in with a reference to the agent name in the <i>design models</i> .
7	Item 2 of the <i>JADE agent template</i> has to be filled in with the platform name to which the agent pertains.
8	In order to complete item 3 of the <i>JADE agent template</i> (Fig. 6.26) the <i>agent model</i> , the <i>tasks and goals model</i> and the <i>interaction model</i> have to be used. Items 3.1 and 3.2 must be completed with all the tasks a given agent offers as services. Item 3.2 must indicate whether the service is offered for internal agents only, or it is also available for agents of other platforms. To do this the engineer has to analyze the <i>interaction model</i> , particularly those interactions in which the agent participates.

Table 6.10 ANEMONA JADE guidelines - Part 2

 JADE guidelines	
ID	Guideline
9	In item 4 of the <i>JADE agent template</i> (Fig. 6.26), the <i>software engineer</i> has to identify the behaviors that will implement the services of item 3, and the internal tasks of the agent (those tasks that are not services). To do this, the engineer has to revise the <i>agent model</i> and the <i>tasks and goals model</i> . In those cases in which the identified behavior implements a service, item 4.3 must be filled in with the service name. For every behavior, item 4.2 must be filled with the service type (simple, cyclic, one-shot, complex, sequential, parallel, fsm).
10	In item 5 of the <i>JADE agent template</i> (Fig. 6.26), the <i>software engineer</i> has to specify the elements that the agent can use in the message content (ontology). To do this, the engineer has to analyze the <i>agent model</i> and the <i>interaction model</i> in order to register the informational mental entities and the content of the messages in the interactions. Item 5.1 must be filled in with the concept name, 5.2 with the name (or names) of one (or more) base ontology, and item 5.3 with the schema that defines the concept structure.
11	In item 6 of the <i>JADE agent template</i> (Fig. 6.26), the <i>software engineer</i> has to specify the set of messages that the agent can interchange. To do this the engineer has to analyze the <i>interaction model</i> , mainly the interaction units that the agent may interchange. Item 6.1 must be filled in with the message name, 6.2 with the interaction unit, 6.3 with a reference to the interaction name, and 6.4 with the participation type (initiator, collaborator).
12	In item 7 of the <i>JADE agent template</i> (Fig. 6.26), the <i>software engineer</i> has to specify if the agent has a physical processing part (that is, if it is a resource holon that controls a machine). If this is the case this item must be filled in with a reference to the <i>functional block interface specification</i> templates in which the physical processing tasks are specified.


 JADE Agent Template			
1. Agent ID		2. Platform	
3. Services			
3.1. Name		3.2. Type	
4. Behaviors			
4.1. Name		4.2. Type	4.3. Implemented Service
5. Ontology			
5.1. Name		5.2. Base Ontology	5.3. Schema
6. Communication			
6.1. Message	6.2. Type	6.3. Interaction	6.4. Participation Type
7. Does it have a physical processing part?			

Fig. 6.26 JADE agent template

- Activity D8: Specify function block interface.* In this activity the *software engineer* has to complete, for every agent with a physical processing part, the *function block interface specification* of the resource-controlling agent tasks. In this way, every “physical” task associated to an agent will be implemented as an IEC 61499 application (function blocks network). The *function block interface specification* template includes a table in which the *software engineer* has to specify the normal operation sequence, the abnormal sequences that may occur, the resource behavior in response to actuator commands perceived by sensors. Also, the engineer can include a state-transition diagram in order to represent the behavior of the device. These characteristics are derived from the *requirements* document, the device manufacturer specification and the characteristics identified in the models

generated in the previous phases. In order to fill this template in, the *software engineer* has to use the *function block guidelines* 3 to 8 in Table 6.11.

Table 6.11 ANEMONA function block guidelines


 Function block guidelines	
ID	Guideline
1	A function block can be seen as a processing unit. They are used as building blocks to define a device controlling or monitoring application. Some function blocks provide controlling behaviors and others provide input and output behaviors.
2	Some devices include preinstalled function blocks that can not be deleted and sometimes new function blocks cannot be added to these devices. On the other hand, there are some devices to which new function blocks can be added or deleted as required. It is very important to study the device specification to figure out the interface definition that will be controlled.
3	A <i>function block interface specification</i> template (Fig. 6.27) has to be completed for every task (physical processing part) of every resource agent in the <i>design models</i> . <i>Function block guidelines</i> 4 to 7 must be used.
4	Item 1 of the <i>function block interface specification</i> template (Fig. 6.27) has to be filled in with the agent name. Item 2 must be filled in with the platform name and item 4 with the name of the physical processing task that is being designed.
5	Item 3 has to be filled in with a unique function block template identifier.
6	The <i>software engineer</i> has to study the manufacturer device specification, the <i>agent model</i> and the <i>tasks and goals model</i> to find out the operation sequences to control. Items 5 and 6 have to be filled in with this information.
7	The <i>software engineer</i> has to study the manufacturer device specification and the <i>requirements</i> document to find out the device behavior and its operation conditions respectively. Item 7 of the <i>function block interface specification</i> template has to be filled in with a record for every combination of: input command, actuator address to which the command is sent, address of the sensor that gets the reply, desired replay, time to get a reply.
8	Item 8 can be filled in with a state-transition diagram that models the device behavior.

Fig. 6.27 Function block interface specification template

Function Block Interface Specification				
1. Agent ID		2. Platform		
3. Template Code		4. Agent task		
5. Normal operation sequence		6. Abnormal operation sequence		
7. Resource Behavior				
Command	Actuator	Sensor	Reply	time
8. State-Transition Diagram				

- *Activity D9: Build deployment model.* In this activity the *software engineer* has to build a UML *deployment model* [137] to depict the physical location of the network nodes (any execution resource such as computer, device or memory) that make up the system and the platform and container distribution of these nodes. To do this the engineer has to analyze the *requirements* document, the *design models* and the *identified platforms* document.

6.3.3.4 Example

Figure 6.28 shows a *function block interface specification* obtained from *activity D10*. The interface specification models the *divert package to belt* physical task of the *conveyor belt* resource holon of the *tracking package holarchy*.

A complete *system architecture* example can be found in Chap. 8.

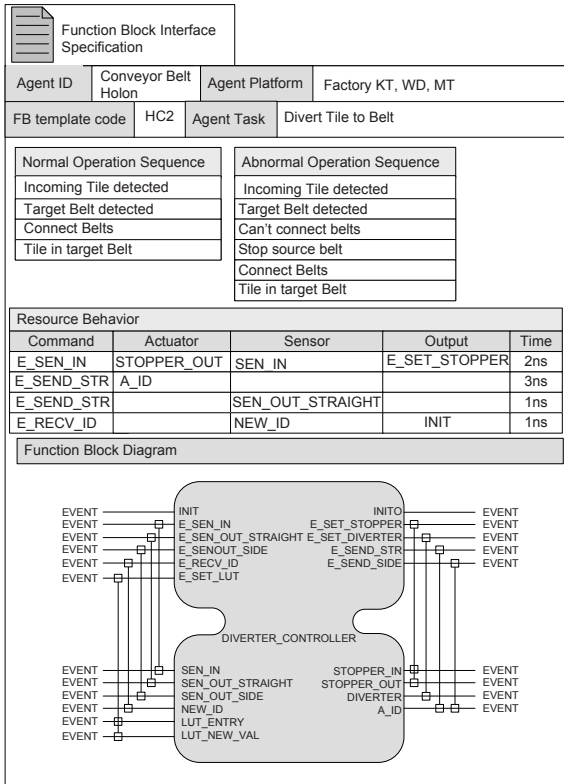


Fig. 6.28 Function block interface specification of the divert package to belt task

6.3.4 Holon Implementation

In the *holon implementation* stage (Fig. 6.29) the *programmer* has to implement the HMS using the *system architecture* defined in the prior development phases. To this end the *programmer* has to execute the following activities.

- **Activity I1: Implement the platform.** In this activity the *programmer* has to define and implement the different agent platforms identified in the *system architecture*. To do this the *programmer* has to follow the JADE programmers guide [146].
- **Activity I2: Implement the agents.** In this activity the *programmer* has to implement the different agents identified in the *system architecture*. To do this the *programmer* may use the JADE programmer and administrator guides [146]. Also, the agents platform-integration tests have to be executed.
- **Activity I3: Implement the function blocks.** In this activity the *programmer* has to implement new function blocks and/or re-use pre-defined function blocks. The

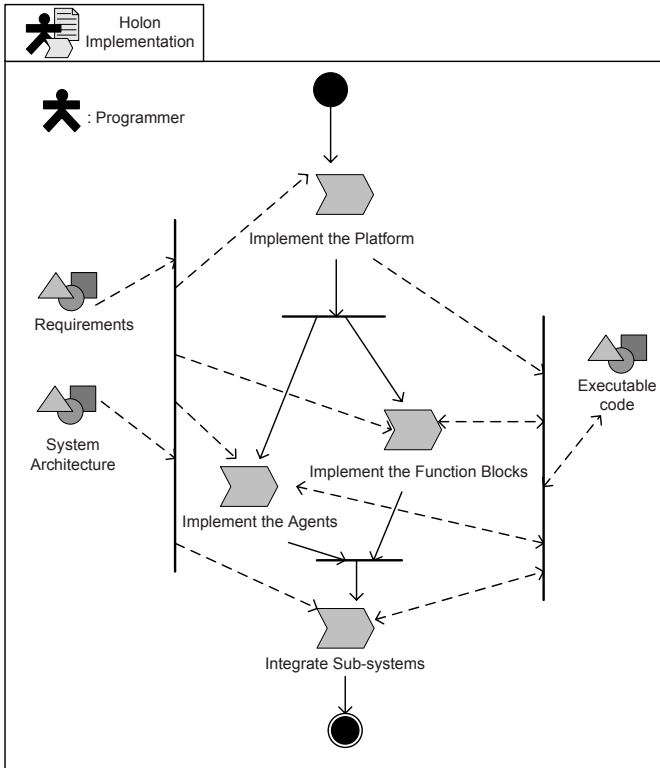


Fig. 6.29 Holon implementation task definition

IEC 61131-3 Structure Text [14] language specification has to be used. Also, the function block integration tests have to be executed on the devices, machines and tools. The *programmer* can follow the programming guidelines defined in the IEC 61499 standard [14, 15].

- *Activity 14: Integrate sub-system.* In this activity the *programmer* has to integrate, for every holon, the information processing part (the JADE agent part) with the physical processing part (the function blocks part). In this integration process two approaches can be used: a blackboard system for communicating the function blocks with the JADE agent [147], or; the implementation of a special function block to manage the interface service [13]. Also, local integration tests (intraholon) and global integration tests (interholon) have to be executed. In these activities the software engineering techniques for integration testing can be used.

6.3.5 Setup and Configuration

In the *setup and configuration* stage the *software engineer* has to control the HMS deployment process in the target execution environment (factory floor, machine, shop floor, work cell, etc.). In this stage the system configuration operations have to be made in order to fulfill the performance goals specified in the *requirements* document. In this activity conventional software engineering techniques for setup and configuration can be used, as well as the guidelines defined in the IEC 61499 [14, 15].

6.3.6 Operation and Maintenance

In this stage traditional software engineering techniques for maintenance can be used. Any maintenance activity that involves new functionalities, adaptation to environment changes and the fixing of design or implementation errors implies a new development process to adapt the analysis, design and implementation documents to the new requirements.

6.4 Conclusions

In this chapter, we have presented the development process of ANEMONA. This process is a mixed top-down and bottom-up approach. The aim of the analysis phase is to provide high-level HMS specifications from the problem *requirements*, which are specified by the *client/user* and that can be updated at any development stage. The analysis adopts a top-down recursive approach. One advantage of a recursive analysis is that its results, i.e., the *analysis models*, provide a set of elementary elements and assembling rules. The next step in the development process is the *holon design* stage that is a bottom-up process to produce the *system architecture* from the *analysis models* of the previous stage. The aim of the *holon implementation* stage is to produce an *executable code* for the *setup and configuration* stage. Finally, maintenance functions are executed in the *operation and maintenance* stage. Our approach provides HMS-specific guidelines to help the designer in every step of development.

A software engineer who follows the ANEMONA method is able to develop a holonic manufacturing system using multi-agent system technology from the beginning. This is so thanks to the abstract agent notion (Chap. 3). Moreover, the guidelines provided by ANEMONA help the software engineer to identify and specify all the possible flexible cooperation scenarios and agent features in order to fulfill the manufacturing system requirements. The ANEMONA methodology is appropriate for the domain of intelligent manufacturing systems and sufficiently prescriptive for a software engineer with minimal training in multi-agent technology.

In the next part of the book an evaluation discussion is presented in Chap. 7 and a complete case study is shown in Chap. 8.

Part III
Evaluation and Case Study

Chapter 7

Evaluation of the ANEMONA Methodology

The notation and the development process of the ANEMONA methodology for holonic manufacturing systems using multi-agent system technology were presented in Chapters 5 and 6. The ANEMONA methodology, is appropriate for the domain of intelligent manufacturing systems and is sufficiently prescriptive for a software engineer with minimal training in multi-agent technology. To substantiate this claim, it would be necessary to perform a large series of industrial case studies with a significant number of software engineers. However, this is beyond the scope of this book because of the quantity of documentation generated, the amount of human resources needed, the investment in time and the very high cost of a complete life cycle of a real industrial case study.

In order to lend support for this claim, we present in this chapter the results of two case studies (of differing complexity) that have been conducted using the ANEMONA methodology. With these case studies we demonstrate the applicability of the development method for holonic manufacturing problems by presenting two industrial case studies. These case studies were reviewed by: (i) software engineers from a manufacturing research and development institute with no prior experience in agent development; (ii) students with minimal agent-technology background, and; (iii) manufacturing engineers with minimal training in multi-agent technology. Moreover, we argue that ANEMONA is more appropriate for intelligent manufacturing problems than other existing (enterprise-based, agent-oriented, and holonic-based) methods by highlighting in what respect the methodology overcomes the limitations of the methodologies discussed in Chap. 4.

The chapter is organized as follows: Sect. 7.1 analyzes the applicability of the ANEMONA methodology and summarizes third-party reviews in order to evaluate the suitability of the methodology. Section 7.2 compares ANEMONA with the state-of-the-art methods presented in Chap. 4.

7.1 ANEMONA Applicability to Intelligent Manufacturing Problems

A software engineer who follows the ANEMONA method is able to develop a holonic manufacturing system using multi-agent system technology from the beginning. That is, ANEMONA is a method with a “uniformity of concepts” and there is no need to change modeling notions and implementation units in any development step. This is so thanks to the abstract agent notion (Chap. 3). Moreover, the guidelines provided by ANEMONA (6) help the software engineer to identify and specify all of the possible flexible cooperation scenarios and agent features in order to fulfill the manufacturing system requirements. To demonstrate these, we have designed two real-world manufacturing systems according to the rules of the methodology. These case studies are briefly discussed below.

The first case study has already been presented in Chap. 6. The parts distribution system (PARDI) is a sub-system of a supply chain management system. An automotive parts supplier company provides equipment parts to its dealers in Europe through a network of distribution centers. This company requires the automation of its supply-chain process. When a particular part is needed for service, the dealership orders it through PARDI and the part is automatically packed and shipped from a distribution center. PARDI also automates the referral parts distribution process. The PARDI system was first developed by a team of manufacturing engineers from a manufacturing institute without the help of ANEMONA. The ANEMONA methodology was therefore applied to the PARDI problem later in order to verify whether ANEMONA is able to support the development process fulfilling the original system requirements. Some ANEMONA development models from this case study can be analyzed in Chap. 6. The process itself and the development results were documented and then evaluated by a team of manufacturing students and engineers. The evaluation can be summarized as follows:

- The ANEMONA holon identification method created a group of agents with the same overall functionalities as the modules of the original system, each agent responsible for its (manufacturing) component.
- The ANEMONA guidelines proposed a set of cooperation scenarios that deal with all of the communication needs of the original system. Moreover the cooperation scenarios were evaluated in order to measure the message passing load and the results outperformed the original development.
- The methodology, in particular the models and the guidelines for identifying and specifying holons, was regarded as intuitively appropriate for designing intelligent manufacturing systems, and it was felt that the methodology would have facilitated the original development process if it had been available at the time.

ANEMONA was thus able to model and implement the PARDI system requirements and the original designers regarded the methodology as appropriate.

The second case study is about a ceramic tile factory. This case study is presented in detail in Chap. 8. In the ceramic tile factory under study, different products with

diverse sizes, designs and compositions are produced. Moreover, the factory has to deal with two kinds of clients: building firms and wholesalers. Several business processes can be distinguished in a ceramic tile factory. Firstly, a design department defines which ceramic products will be produced in the current season. Then, a sales and orders forecast is made by the commercial department, based on historical sales data, orders, etc. Later, medium-term production orders are defined, sequencing the different product lots to be produced. This sequence configures a tentative master plan, which is normally used as the major input data to generate the production programming, which includes activities such as determining start time and resource allocation for a specific lot production. The production department uses the master plan, information about raw-material availability, plant status, etc., to generate the production programming schedule. Finally, all tasks related to the production and final storage of the different product lots are carried out. Usually, the ceramic tile production process is represented as a three-stage hybrid flow shop with sequence dependency in which three stages can be identified: press and glass lines (first stage), kiln (second stage), and classification and packed lines (third stage). Each stage is a productive phase with different times, resources and objectives. Finally, the commercial department sells factory products and manages orders from clients. The HMS for the tile factory must: (i) integrate the different departments of the company, (ii) arrange factory resources for both on-demand and stock production orders, and (iii) automate resources and processes controls at different levels in the company.

This case study was completely modeled following the analysis and design phases of ANEMONA. The implementation, maintenance and configuration phases were applied to a sub-part of the whole system. This sub-system is the *tasks scheduling* cooperation scenario on the factory floor. The *system architecture* developed using ANEMONA was documented and then evaluated by a team of manufacturing students and engineers. The evaluation can be summarized as follows:

This case study showed that the methodology was able to support the analysis, design and implementation tasks successfully. In particular, the following observations can be made:

- The method for analyzing decision making captures the relevant aspects of the manufacturing system, that is, resource allocation, timing decisions and management decisions.
- The ANEMONA guidelines for identifying and specifying the holons produce a set of agents that reflects the manufacturing components, and the different control and management tasks are distributed among the different cooperation scenarios and holarchies.
- The work flows, interaction protocols, and cooperation domains are suitable. All of the possible cooperation opportunities and situations for implementing the system requirements were identified in a satisfactory and intuitive way.

In order to evaluate the implementation of the experimental prototype, different simulation tests have been executed in which the operating conditions have been analyzed. These tests give a measure of the system reliability, robustness, flexibility and efficiency, not only under normal conditions, but also under nonstandard

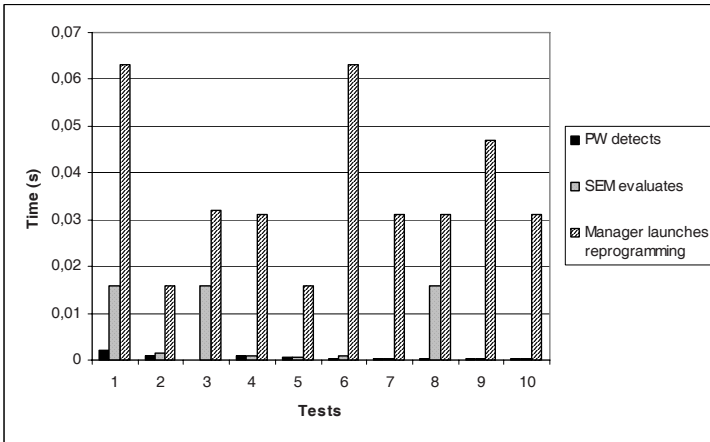


Fig. 7.1 Time needed for error recovery when error does not affect the calculated schedule

operating circumstances. More specifically, tests on functionality and efficiency of the system; concurrent programming capacity; system reaction to unexpected events such as machine failures; and also agent failures are carried out. A complete analysis of these tests can be studied in [148].

In Figures 7.1 and 7.2, there are some significant results pertaining to the evaluation experiments for the *tasks scheduling* cooperation scenario in the factory floor. These graphs represent the time elapsed from the moment when an error happens until the moment when the *plant wrapper* agent detects it (“PW detects”); then until the *scheduling execution monitor* agent checks whether the confirmed schedule must be re-programmed (“SEM evaluates”); and finally until the moment when the

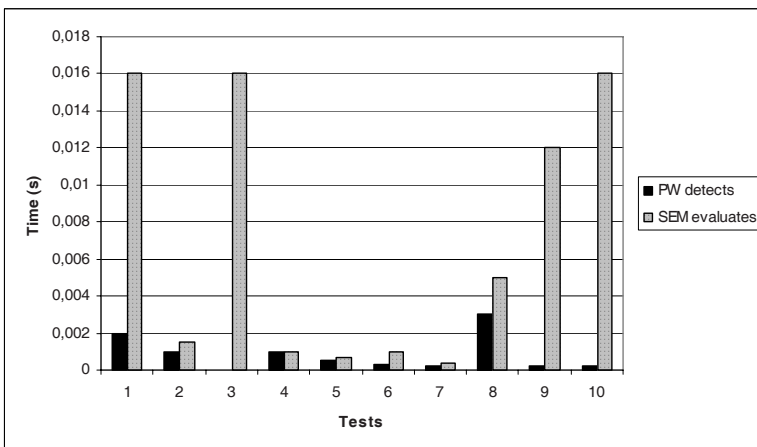


Fig. 7.2 Time needed for error recovery when error affects the calculated schedule

manager agent receives the notification and launches the reprogramming process (“Manager launches reprogramming”). When the error does not affect the confirmed schedule (Fig. 7.1), the process usually takes around 0.02 s from the failure event until the *scheduling execution monitor* decides that the error does not affect it. But, as shown in Fig. 7.1, when the *plant wrapper* or the *scheduling execution monitor* agents are busy, the time needed increases to up to 0.25 s. On the other hand, when the error affects the confirmed schedule (Fig. 7.2), the process usually takes around 0.32 s, from the failure event until the *manager* launches the reprogramming.

7.2 ANEMONA vs. State-of-the-Art Methods

In this section we compare ANEMONA to the state-of-the-art methodology presented in Chap. 4. The goal of this section is to demonstrate that ANEMONA is more suitable for the design of intelligent manufacturing systems than the other existing methodologies.

The state-of-the-art review in Chap. 4 has shown that existing methodologies are either not appropriate or not sufficiently prescriptive for developing holonic manufacturing systems. In particular, the review has identified eight HMS modeling requirements summarized in Sect. 4.1. Neither of the state-of-the-art methods from the HMS field, multi-agent system technology and enterprise modeling cope with all the HMS modeling requirements. A summary review can be analyzed in Table 4.1.

ANEMONA expands the state-of-the-art in that it provides a set of appropriate and sufficiently prescriptive methods, models, techniques and guidelines for each of the eight HMS modeling requirements. Thus, it improves methodological support for developing holonic manufacturing systems. The complete details of these methods, models, guidelines, etc. and their usage can be studied in Chaps. 5, 6 and 8.

ANEMONA is a Multi-agent System methodology for HMS analysis and design based on the abstract agent notion (Chap. 3) and on HMS requirements. The development process of ANEMONA aims to provide HMS designers with clear, HMS-specific modeling guidelines, and complete development phases for the HMS life cycle.

HMS modeling requirements 1, 2, 3 and 4 are supported by the Abstract Agent notion (Chap. 3), the development models (Chap. 5), and the ANEMONA development guidelines (Chap. 6).

The ANEMONA holon identification and specification guidelines deal with HMS modeling requirement 5. Requirement 6 is supported by the ANEMONA top-down recursive analysis phase (Chap. 6) and the abstract agent notion, while the ANEMONA mixed top-down and bottom-up development process supports the HMS modeling requirement 7. Requirement 8 is dealt with by the specific HMS guidelines to help the designer in identifying and implementing holons.

In summary, the methodology provides a greater degree of methodological support for designing holonic manufacturing systems than existing approaches because the models employed are more appropriate for capturing the HMS requirements, controlling and management decisions. The ANEMONA method and its guidelines are more prescriptive in that they provide more criteria for performing holon specification and design.

7.3 Conclusions

In this chapter we have presented an evaluation discussion on the applicability of ANEMONA to intelligent manufacturing problems. Also, we have argued that ANEMONA is sufficiently prescriptive for a software engineer with minimal training in multi-agent technology. These claims are supported by two real case studies evaluated by third parties. The case studies were reviewed by: (i) software engineers from a manufacturing research and development institute with no prior experiment in agent development; (ii) students with minimal agent-technology background, and; (iii) manufacturing engineers with minimal training in multi-agent technology. Moreover, we have argued that ANEMONA is more appropriate for intelligent manufacturing problems than other existing (enterprise-based, agent-oriented, and holonic-based) methods by highlighting in what respect the methodology overcomes the limitations of the methodologies discussed in Chap. 4.

In the next chapter a complete case study from a ceramic tile factory is presented.

Chapter 8

Case Study

In this chapter we present and develop a case study of a ceramic tile factory. The chapter is organized following the different phases of ANEMONA. Firstly, we present the *requirements* document of the case study (Sect. 8.1). In Sect. 8.2 the different products of the *analysis* stage are presented. The *system architecture* is designed in Sect. 8.3.

8.1 Requirements

KCG is a company that designs, manufactures and sells ceramic tiles. Its products are sold in more than 150 countries. The KCG holding is composed of:

- KT: design, manufacturing and marketing of ceramic products.
- MT: design, manufacturing and marketing of ceramic products.
- WD: design and manufacturing of ceramic glazes.
- KR stores: marketing of ceramic products, bathroom fittings, taps, and hydro-massage columns, bath and cabins.
- Branches: in different countries.

The ceramic products of KCG include: traditional, white and red body wall and floor tiles, sophisticated polishing and rectifying coverings, glazed and technical porcelains with high technical qualities. KCG's clients are: building companies, wholesalers and retailers. Sales to building companies are on demand. On the other hand, sales to wholesalers and retailers are based on warehouse stock. KCG has three types of suppliers, based on the material they supply: (i) clay supplier; (ii) ceramic frits and glaze supplier,¹ and; (iii) gas supplier.

In the following subsections the ceramic tile *requirements* document is detailed.

¹ Ceramic frits are vitreous compounds, non-soluble in water, obtained by melting and then rapidly cooling, carefully controlled blends of raw materials.

8.1.1 Organizational Chart/Departments

Figure 8.1 depicts the KCG *organizational chart*. In order to simplify the diagram it only depicts the department managers without detailing the personnel assigned to every department. The company structure is defined as follows.

- **KCG President:** is the head of KCG. The president is in charge of global management and maintains a hierarchical relation with: the Branch Manager, the Design Manager, the Marketing Manager, the Store Manager, the Ceramic Tile Factory Manager and the Transportation Manager.
- **Branch Manager:** is the interface of branches in the KCG holding. The Branch Manager is in charge of the different KCG branches in different countries and has a hierarchical relationship with the managers of the different branches. The Branch Manager also has cooperation relationships with: the Design Manager, the Marketing Manager, the Store Manager, the Transportation Manager and the Ceramic Tile Factory Manager.
- **Design Manager:** is in charge of the management and definition of KCG product design. The Design Manager cooperates with the Ceramic Tile Factory Manager in order to communicate the product designs and approve the designs defined by KT, WD and MT. The Design Manager gets historic sales data on a given product from the Marketing Manager, the Store Manager and the Branch Manager.
- **Marketing Manager:** is in charge of global sales management and KCG's advertising campaigns. The Marketing Manager has a hierarchical relationship with the Sales Manager and the Advertising Manager, and cooperates with the Branch Manager, Design Manager, Store Manager, Transportation Manager and Ceramic Tile Factory Manager.
- **Store Manager:** is in charge of the global management of the KCG stores. The Store Manager has a hierarchical relationship with the KR Store Managers. Also, the Store Manager cooperates with the Branch Manager, Design Manager, Marketing Manager, Transportation Manager and Ceramic Tile Factory Manager, in order to communicate manufacturing work orders, information data on the most-sold product design, transportation work orders, etc.
- **Ceramic Tile Factory Manager:** is in charge of managing the cooperation among the different ceramic companies of the KCG holding. The Ceramic Tile Factory Manager has a hierarchical relationship with the KT Manager, the WD Manager and the MT Manager. Also, the Ceramic Tile Factory Manager cooperates with the Branch Manager, Design Manager, Store Manager, Transportation Manager and Marketing Manager, in order to communicate warehouse sales data, product design information, incoming product transportation orders and manufacturing work orders.
- **Transportation Manager:** is in charge of product transportation management from one ceramic factory to another, from a ceramic factory to a store, from one store to another, etc. The Transportation Manager cooperates with the Store Manager and the Ceramic Tile Factory Manager.

- The *i* Branch Manager: is in charge of KCG branch management in a given country. *i* Branch Manager is subordinated to the Branch Manager on the KCG company policies.
- Sales Manager: is in charge of managing KCG's global sales. The Sales Manager is subordinated to the Marketing Manager in order to register the store sales, the warehouse sales from KT, WD and MT, and the different branch sales.
- Advertising Manager: is in charge of defining and managing KCG's advertising policies.
- The *i* KR Store: manages a KR store and communicates purchase orders and/or manufacturing orders to the Stores Manager, receives the product samples and catalogs.
- KT Manager: manages the KT ceramic factory based on KCG's policies. The KT Manager controls the different KT departments: KT Purchasing, KT Design, KT Manufacturing, KT Human Resources, KT Warehouse, KT Sales.
- KT Sales Manager: manages the KT raw material and supervises the KT factory Raw Materials Warehouse Managers.
- KT Design Manager: defines and maintains KT manufacturing product designs. Some of these designs may be defined by the KCG Design department.
- KT Manufacturing Manager: manages and controls the planning, scheduling, and production control processes of KT. The KT Manufacturing Manager coordinates the activities of the Planning Manager and the KT Factory Manager.
- KT Human Resources Manager: manages KT employees.
- KT Warehouse Manager: manages the KT finished products warehouse.
- KT Sales Manager: manages the KT finished products warehouse sales.
- KT Planning Manager: is in charge of defining the manufacturing plan and scheduling of KT.
- KT Factory Manager: controls and coordinates the Managers of KT Factory 1 and KT Factory 2.
- KT Factory *i* Manager: manages the production processes of the factory *i* of KT.
- KT Factory *i* Press and Glazing Manager: manages the different press and glazing lines of the factory *i* of KT.
- KT Factory *i* Kiln Manager: manages the different kilns of the factory *i* of KT.
- KT Factory *i* Classification Manager: manages the different classification lines of the factory *i* of KT.

The WD and MT department definitions are similar to the KT departments.

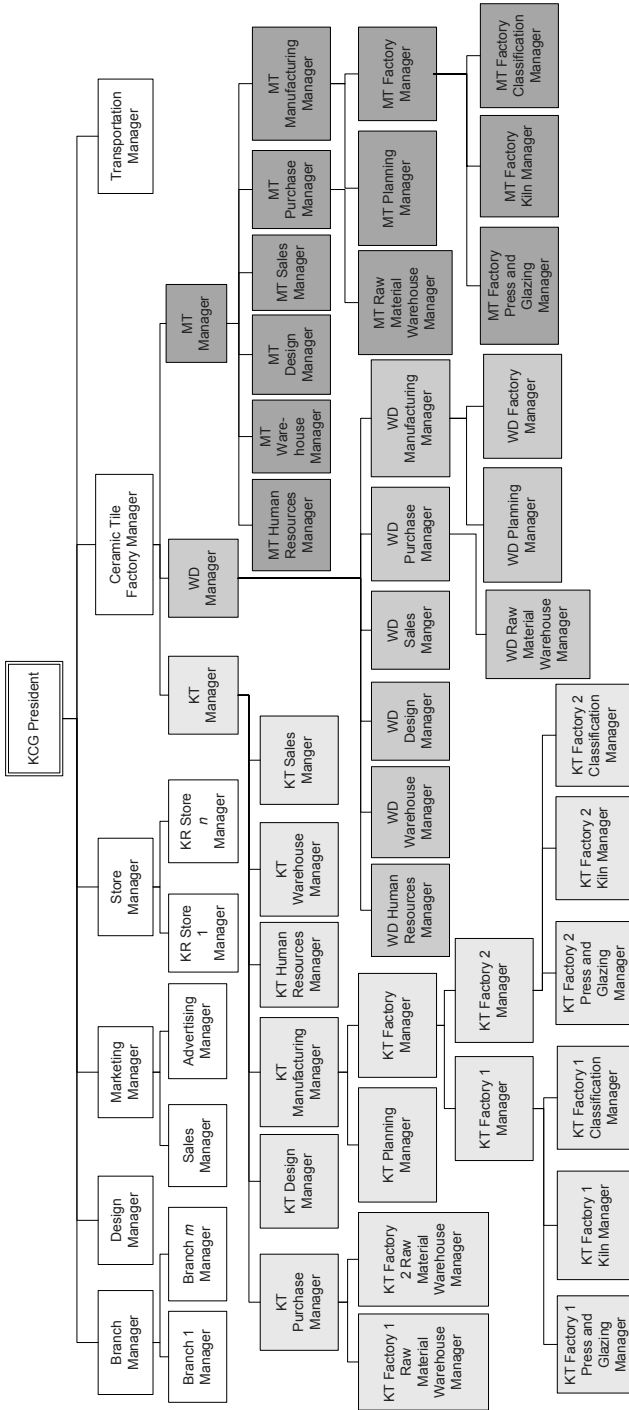


Fig. 8.1 KCG organizational chart

8.1.2 Business Processes

The *business processes* are depicted in Fig. 8.2 (the notation used is DFD). In this figure we can see a product definition process (process 1); a set of ceramic product manufacturing processes (processes 2 to 6); a process for building firm orders, or a process for maintaining finished product warehouse stock for retailers; a purchase process (process 8) for purchasing raw materials from suppliers, and finally, a sale process (process 7) that receives finished products and deals with company clients.

The DFD in Fig. 8.2 shows process 1 – ceramic product definition – which is activated by own product samples (defined by the KCG Design Department) and by product samples from frit and glaze providers. The result of process 1 is a list of the type of products (product catalog) that will be produced in a period of time. This list includes the specification of every type of product. A product specification includes the material to be used, the product design, the type of press pattern to be used, the glaze machine configurations, the kiln configuration to fire the ceramic product and the product demand forecast.

The global manufacturing process is the sequence of processes 2, 3, 4, 5 and 6. Process 2 – demand forecast analysis – uses as input data the product types and the sale history. Using these data the process calculates the demand forecast for a time period. The demand forecast is used in order to maintain a certain stock level in the finished product warehouse. This process is currently managed by the Manufacturing Manager using a software application to define an initial demand forecast for the Marketing Department that validates and tunes it if required. This is a very important process for the definition of the master plan. The demand forecast is based on the building companies' and retailers sale history. The Manufacturing Manager defines the master plan in process 3 – master plan definition (for a time period of three or four months). In this process the supplier purchase orders are defined (if required). The purchase orders are defined from the manufacturing orders, demand forecast, raw material availability, and the finished product stock. The master plan includes the manufacturing orders. In the manufacturing orders the models, manufacturing lots and manufacturing priorities are defined.

The Manufacturing Manager interacts with the different manufacturing section managers (glazing, pressing, kiln and classification) in order to define the list of the manufacturing orders that will be executed in the following five or six days. The list definition takes the different section manager's incident reports into account (process 4). In process 4 – schedule definition and monitoring – the loading, scheduling and timing tasks are executed. This is the most important process with regards to company manufacturing performance. The process works with master plan, the different section managers' reports, the raw material availability and the finished product stock. The schedules, which have already been defined, are revised and new manufacturing orders are included if required. The goals of this process are: to fulfill the priorities defined in process 3, to maximize resource utilization and to minimize the stock levels (maintaining a safe level of stock). These data are used to define a manufacturing schedule that is used in process 5 – manufacturing. Fi-

nally the Finished Products are stored in the finished product warehouse in process 6 – warehousing.

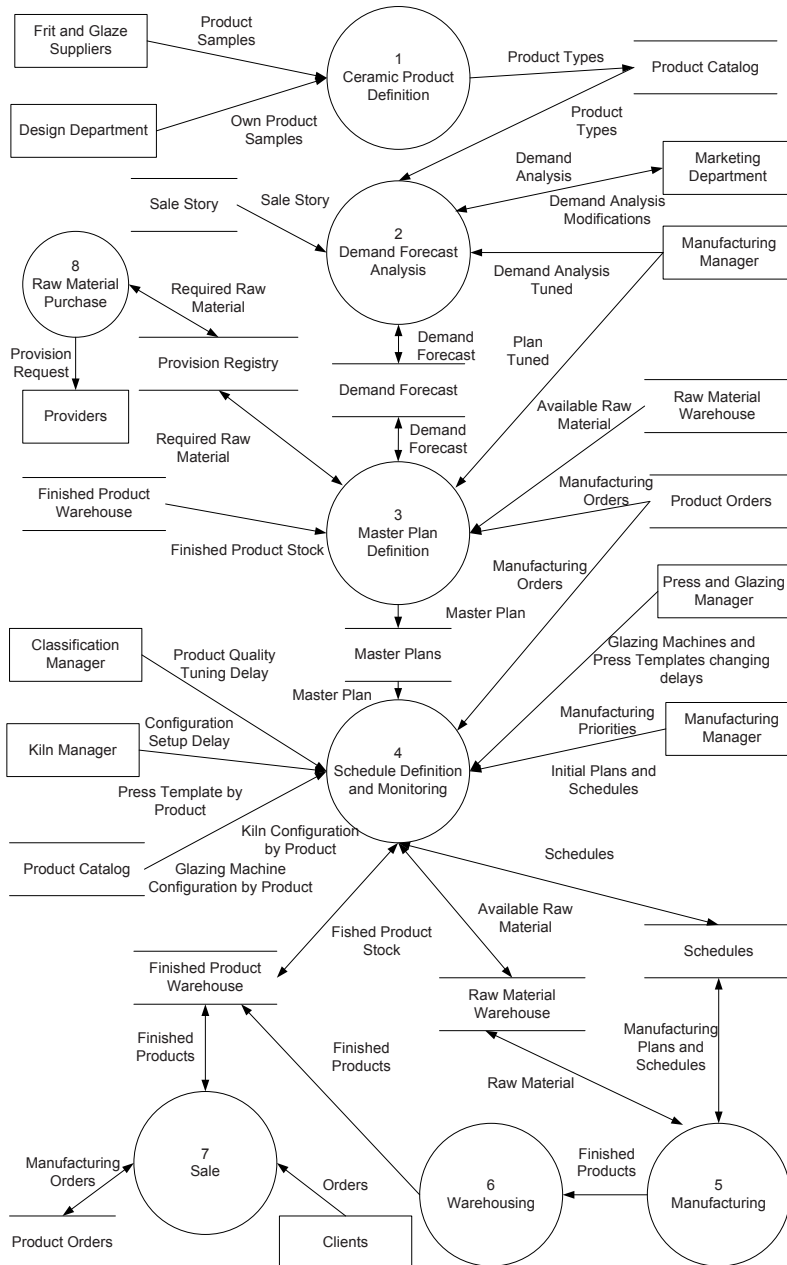


Fig. 8.2 KCG business processes

The business processes in Fig. 8.2 are neither integrated, automated, nor flexible. This means that an error in one of them is transferred to the other process and so can not be detected on time. Changes in client requirements are hard to adapt to the business processes. Company productivity is too dependent on the demand forecasts. The reply to a client order is too dependent on finished product stocks. Coordination among the different ceramic factories is inefficient.

8.1.3 System Scope

The holonic manufacturing system must:

- Automate the management of the raw material warehouse, and supply a management system for the purchase department. These functionalities imply the modification of process 8 in Fig. 8.2.
- Provide a master plan definition module (process 3 in Fig. 8.2).
- Automate the schedule definition and monitoring process (process 4 in Fig. 8.2).
- Automate the controlling of the manufacturing process (process 5 in Fig. 8.2).
- Integrate processes 4 and 5 in Fig. 8.2.
- Minimize manufacturing based on demand forecasts.
- Automate the finished product warehouse management process (process 6 in Fig. 8.2).
- Provide a system for store sales and warehouse management (process 7 in Fig. 8.2), coordinating processes 4 and 5 in Fig. 8.2.
- Provide a system for the coordination of the different ceramic factories.

8.1.4 Processes to Control

In this section the *processes to control* are defined.

The process *raw material management* – (P1) is in charge of managing a registry of available raw materials and their localization in the warehouses, a date forecast for material out of stock, arrival date of raw material, and date, quantity and destination of the raw material that left the warehouse. The *purchase management aid* sub-system – (P2) must interact with the raw material management process and the production sub-system (processes 4 and 5 of Fig. 8.2) in order to find out and recommend the following to the Purchase Department: raw material purchase orders, estimated raw materials out of stock dates, supplier delivery times, and transportation costs.

The process *finished products warehouse management* – (P3) registers the finished product's warehouse entry and exit, the localization of products in different warehouses, the product stock and the available warehouse free spaces. This process has to interact with the production sub-system in order to input finished products from the factory, and to share warehouse free-space availability. Moreover, it

has to interact with the purchase management aid sub-system in order to supply information on product availability and accept product requests.

The *purchase management aid* sub-system – (P4) manages store sales and the direct sales from the finished products warehouse. This sub-system interacts with the finished products warehouse management process in order to find out a given product's stock level. In cases where there is no stock in the warehouse, the sub-system initiates a negotiation interaction with the production sub-system. The goal of this interaction is to estimate the production cost and finished production date of the client order. In this way the shop assistant can inform the client of the client order availability in a short time (this should not be longer than 10 minutes).

The *master plan definition aid* sub-system – (P5) identifies and recommends the foreseen manufacturing plans to the Manufacturing Manager. These plans include product configurations' quantity to produce (manufacturing lots) and planned manufacturing dates. In order to fulfill its goals, the master plan definition aid sub-system bases its processes on the demand forecasts (defined by the marketing department), the manufacturing orders (managed by the sales department), information from the finished product warehouse management and the raw material warehouse management processes, and the interaction with the production sub-system (processes 4 and 5 of Fig. 8.2).

The *production* sub-system – (P6) integrates processes 4 and 5 in Fig. 8.2. This sub-system defines and controls manufacturing schedules and controls the manufacturing process. In the schedule definition (P6.1) the client orders get the highest priority, and the production sub-system uses the master plan defined by the master plan definition aid sub-system, only in cases where there is time available.

The *manufacturing process* – (P6.2) (process 5 in Fig. 8.2), is a sub-process of the production sub-system. It is the most complex process to control. There are two types of processes in the ceramic tile manufacturing: the twice-fired and the once-fired process. In each type of production process there are different tile formats (size). At the same time, every format has a different pressing mold. The number of tile formats changes dynamically, and usually increases year by year.

The ceramic tile (Fig. 8.3) is made from clays that are passed through a humid grinding process and then a spray dryer. The dried clay is pressed, making pieces that, in the twice-fired process, are fired before glazing (biscuit), then are glazed and fired for a second time. In the once-fired process the pieces go straight to the glazing. Finally, the glazed product goes to the kiln. Between the glazing lines and the kilns there are intermediate warehouses in order to collect the products from the glazed lines. When the products are fired they are transported to a warehouse waiting to be classified into different product qualities. At the same time a worker analyzes possible surface defects. The products are packaged into cardboard boxes and several AGVs² transport them to the finished product warehouse. The product is ready to be delivered.

² Automated guided vehicles

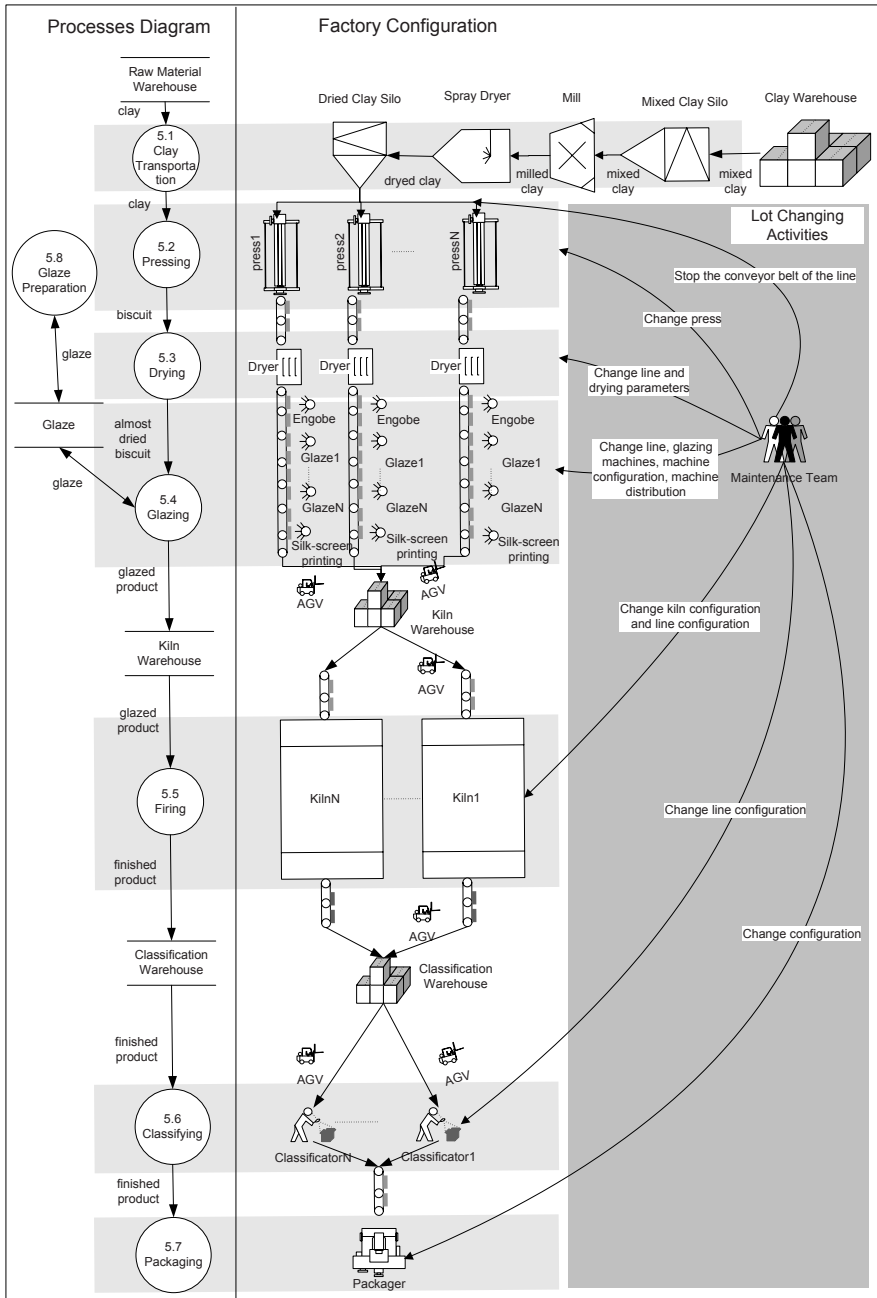


Fig. 8.3 KCG factory process

The clay transportation – process 5.1 in Fig. 8.3 – represents the sequence: mixed clay silo, mill, spray dryer and dried clay silo, the transportation among them and finally the transportation from the dried clay silo and the pressing lines. The dried clay mixture feeds the presses where the tiles are formatted (process 5.2). The next step is the drying process (process 5.3 in Fig. 8.3). This process makes the biscuit more adhesive for the glaze application. The biscuits are transported on conveyor belts to the driers. During transportation the biscuits are polished in order to remove the dust adhered to the pieces after the pressing process. The pieces remain in the dryers for a period and the temperature used in the drying process depends on the thickness of the tile. Temperature and humidity levels are checked periodically. The pieces leave the dryer at the optimal temperature for the glazing process (process 5.4 in Fig. 8.3). In the first part of the line every piece receives two glazing applications of engobe and base, respectively. The base application is done just as the engobe starts to dry (this is achieved by regulating the line speed). These layers are applied in order to remove the red color of the paste, as well as to remove or correct possible surface defects and improve the adherence of the paste. Following the two applications there is a long line for decorating the pieces by means of different glaze applications. The glazing lines can be configured by changing the glazing machine sequence based on the tile design. The glazed product goes through the kilns where the glazed biscuit is fired (process 5.5) in order to get the finished product. When the biscuit is in the kiln it goes through different sections called prekiln, preheating, firing, natural cooling and forced cooling. The kiln temperature is regulated by a microprocessor. The product leaves the kiln at a high temperature, but with the required resistance and rigidity. The fired products are carried to the warehouses waiting to be classified (process 5.6). In the classification process the tiles are tested in order to classify them by quality and calibre. There are three qualities of ceramic tiles (first, second and third). The tiles with serious defects are thrown away in order to mill them later and to carry them back to the clay warehouse. When the finished products are classified they are packaged into cardboard boxes with a given tile number depending on the tile format. The glazing preparation process (process 5.8) is done before the tile production is started. The glazes are manufactured from frits, colors, additives and water. These products are put into special mills and milled until the required granularity and the viscosity are achieved. The glazed tiles are stored into dipping tanks.

The process diagram in Fig. 8.3 represents the manufacture of a given product lot. Due to the great competitiveness in the ceramics field, companies have to deal with a wide range of customized product orders. Because of this the catalogs are very large, which makes manufacturing planning for the production of several different models difficult for ceramic tile companies. Manufacturing lines must be shared among different product designs, with the associated problems related to changing-lot activities. The changing-lot activities (as shown in Fig. 8.3) imply a series of line configuration activities in order to produce a given product. Apart from the time that it can take to configure the line, there is another delay that is associated with the tuning of the machines to achieve the quality required for a given product.

When a new order is initiated some tuning activities are required to calibrate the machines in order to manufacture the desired product. In addition to this tuning time, more time is required to change the machines and their localization in the line. This time is called preparation time. The tuning time plus the preparation time define the changing lot time, which can be very long.³

Nowadays, the controlling processes and machine configuration of KCG's manufacturing line are done by workers. As a consequence, personnel costs are high due to the fact that a full-time work team is required for this activity. Line productivity depends on the maintenance workers' ability and expertise. Several configuration activities are mechanical and not automated. The propagation through the tuning activities of the line are not very efficient as machines are configured manually. Schedule definition and monitoring process feedback is not done in real time and it is inefficient due to the managers' communication policy.

In order to make the company agile and flexible, the automation of several processes is required. KCG wants to change the controlling and configuration system: lines, glazing, engobe and printing machines, kiln control, dryer, classification and packaging machines. The manual process for the changing and configuration of the presses will be maintained due to the physical characteristics of the presses. Also, the expert classification by workers will be maintained. Moreover, tuning operations carried out by workers on the line should be permitted.

Among the physical components to control there are: conveyor belts, glazing and engobe machines, drying machines, AGVs, kilns, classification machines, etc. Due to space limitations in this chapter we will only discuss the kiln specification. The kiln is made up of a set of rollers for transporting the tile inside the kiln and the kiln itself. The kiln is divided into three zones: preheating, firing and cooling. The kiln is designed so that the air heated in the cooling zone is reused at the firing zone. Also, the combustion air from the firing zone is piped to the preheating zone. The temperature is regulated independently in each of the three zones. The inner kiln temperature reaches 1500 degrees Celsius. The total length of the kiln is 80 meters and the means of combustion is natural gas. Every zone has a thermostat for measuring the temperature, a security thermostat and several burners. It also has chimneys and extractors for the expulsion of the combustion fumes. The kiln roller system is mechanical. By regulating the roller speed it is possible to tune the length of time the product remains in the kiln. The kiln control process takes into account the different operation states that the kiln may have: entering products to the kiln, the kiln is full of products, drawing of kiln, changing zone temperature by product type, a kiln component error, roller speed, load and unload sub-systems synchronization.

³ In a conventional ceramic tile factory the time required to change the press and configure the glazing line can be considerable – around 7 hours. Changing the configuration and classification of the kiln can take 1 hour or more.

8.1.5 Operation Conditions

In this section we describe the required inputs and outputs, and the possible runtime operation changes and failures. of every process that needs to be controlled. This information is presented in Tables 8.1 and 8.2.

Table 8.1 Operation conditions

Process	Input	Output	Change/Failure
P1	Raw material input data (type, amount, date, supplier). Raw material output data (type, amount, date, destination). Request on raw-material availability data (type, amount, date).	Raw-material availability data.	Warehouse delay on input/output of raw material. New type of raw material, with new preservation requirements.
P2	Warehouse capacity. Raw-material availability (type, date, out of stock forecasted date). Raw-material output flow (type, amount, time period).	Purchasing recommendation (type, amount, date, provider)	New supplier and/or raw material.
P3	Finished product entry data (type, amount, date, factory floor from which it comes). Finished product output data (type, quantity, date, destination). Information request on finished product availability (type, quantity, date). Information request on warehouse free-space availability (date, product type).	Information on finished product availability. Information on warehouse free-space availability.	Warehouse delay on input/output finished products. New type of finished product, with new preservation requirements and formats.
P4	Client order (product type, quantity, date).	Order availability (date, cost).	
P5	Forecasted demand information, manufacturing orders, raw material availability, finished product stock. Master plan definition request. Master plan modification request.	Tentative master plan (product configuration, quantity to produce, estimated production date).	New product configuration.
P6	Client order scheduling order. Lot manufacturing order.	Client order scheduling simulation. Manufactured lot	The order cannot be manufactured. The lot cannot be manufactured. New factory floor configuration.

Table 8.2 Operation conditions – (cont.)

Process	Input	Output	Change/Failure
P6.1	Scheduling definition request. Scheduling modification request.	New schedule. Modified schedule	The schedule can not be defined. The schedule cannot be modified.
P6.2	Lot manufacturing order	Manufactured lot.	Factory floor failure (machine, resource, product). New machine, resource or product.
Kiln	Change the temperature kiln zone request. Roller speed change request.	Temperature modified. Roller speed modified.	Kiln component failure. Input/output kiln sub-systems synchronization problem.

8.1.6 Goals

The HMS goals are:

- To coordinated the KCG ceramic tile company's activities.
- To integrate business processes 3, 4, 5, 6, 7 and 8 in Fig. 8.2.
- To minimize master-plan-based scheduling and to maximize on-demand manufacturing.
- To make KCG's manufacturing process more flexible with regards to resource and work-order changes.
- To automate manufacturing control.
- Ability to response to mechanical or factory control failures.
- To allow human intervention in the controlling and aiding processes.
- To optimize KCG's client response speeds.

8.2 Analysis

In this section we present the different *analysis models* obtained from applying the analysis activities to the case study. The goal of the analysis stage is to identify the holons that make up the system and to provide an initial holon specification. This is a top-down, iterative and incremental stage.

8.2.1 Iteration 1

In the first iteration the manufacturing system is decomposed into a HMS, in which the constituent entities cooperate in cooperation domains in order to fulfill the system goals. In order to identify these cooperation domains/scenarios, activities *A1*, *A2*, *A3* and *A4* are applied. The *HMS-UC guidelines* (Table 6.3) of the complex activity *determine use cases* (Chap. 6) are also used. The products of these activities are shown in Table 8.3 and Fig. 8.4.

Table 8.3 Iteration 1, system goals

ID	Goal
1	To coordinate the manufacturing activities of ceramics companies.
2	To automate the raw material warehouse management process.
3	To automate the finished product warehouse management process.
4	To supply an online recommendation to the shop/warehouse assistant about the delivery date of a client order.
5	To recommend raw-material purchase orders to the purchase department.
6	To optimize KCG's response speed to clients.
7	To recommend a master plan to the Production Manager.
8	To minimize master-plan-based scheduling and maximize on-demand manufacturing.
9	To automate scheduling and rescheduling of the manufacturing processes.
10	To automate manufacturing control.
11	To integrate scheduling and rescheduling with the factory control.
12	To allow human intervention in the controlling and aiding processes.

Table 8.3 summarizes the goals identified using *HMS-UC guidelines* 1 to 6 from Chap. 6. From these goals and *HMS-UC guidelines* 7 to 13, we have specified the cooperation domains to fulfill them (Fig. 8.4). We have identified 10 use cases. In the following activities of the analysis stage the system goals are associated with the use case managers (Fig. 8.5). The use case will be decomposed in the next iterations.

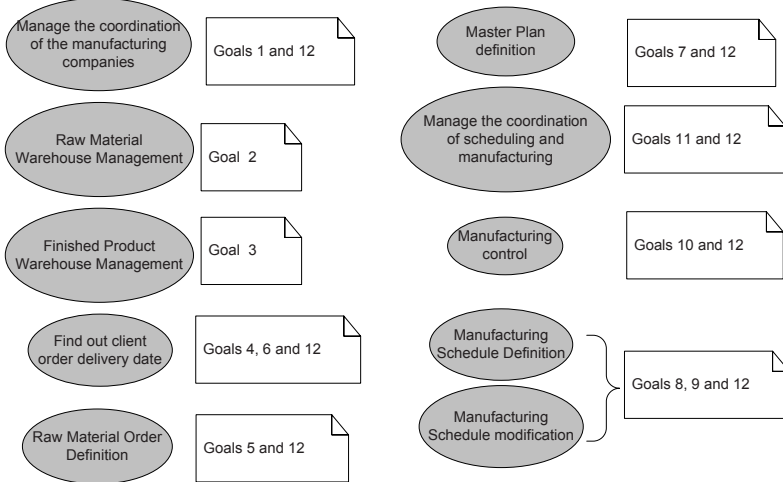


Fig. 8.4 Iteration 1, use case diagram

The next step in the analysis stage is the *specify use cases* (Chap. 6). The roles assignment to the use case managers (activity A5) can be seen in Fig. 8.5. This

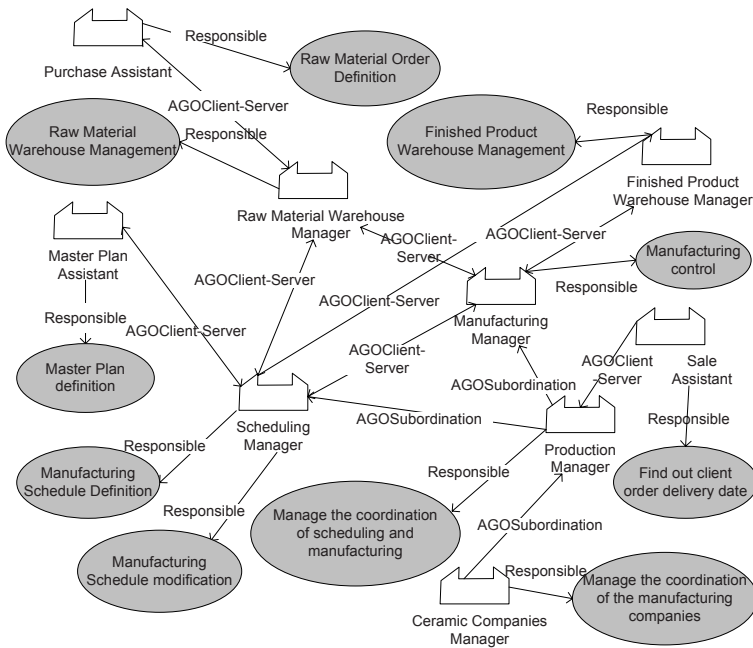


Fig. 8.5 Iteration 1, organization diagram

figure also models the social relations among the roles of the organization (activity A6). These relations (derived from the *requirements* document) are of *AGOCClient-Server* type. This is so because an information or service request between two roles is required (for example, *purchase assistant* and *production manager*). The relation is of *AGOSubordination* whenever there is one manager role and one subordinated role (for example, *production manager* and *scheduling manager*).

Apart from the *organization diagrams*, activity A6 produces *interaction diagrams* (Fig. 8.6) in which the communication needs among the HMS roles are specified.

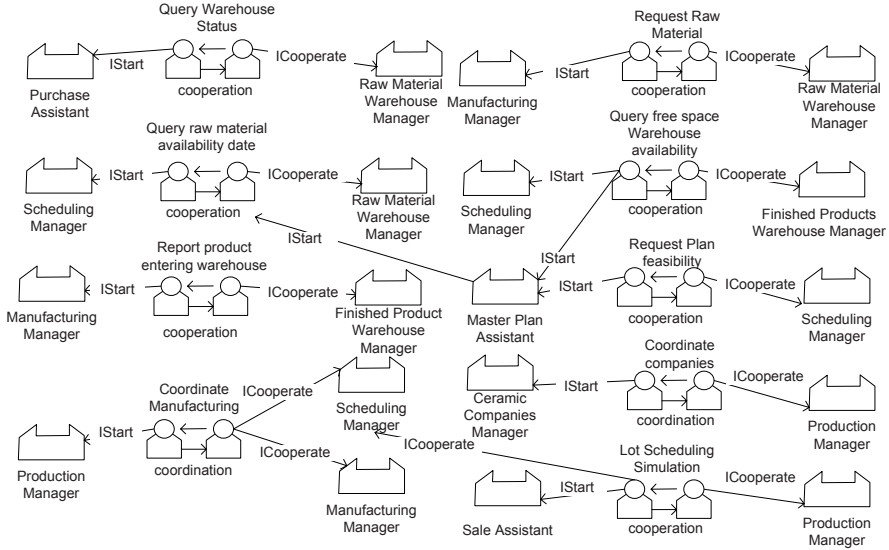


Fig. 8.6 Iteration 1, interaction diagram

We have identified the following interactions for the KCG HMS:

- *Query Warehouse Status*: The *purchase assistant* needs information on the raw material stock, and to this end it starts a communication process with the *raw material warehouse manager*. There is one manager for every warehouse.
- *Request Raw Material*: The *manufacturing manager* requests raw material for the manufacturing process from the *raw material warehouse manager*.
- *Query Raw Material Availability Date*: In the use cases *manufacturing schedule definition* and *manufacturing schedule modification*, the *scheduling manager* needs information on the raw material required in order to manufacture a given product. To this end, it queries the availability to the *raw material warehouse manager*. The *master plan assistant* can also initiate this interaction in order to get information for the definition of the master plan.
- *Report Products entering Warehouse*: In this interaction the *manufacturing manager* informs the *finished product warehouse manager* that there is a given product quantity entering the warehouse.

- *Coordinate Manufacturing*: The *production manager* initiates an interaction with the *scheduling manager* and the *manufacturing manager* in order to coordinate their activities in the production process.
- *Query Free Space Warehouse Availability*: In order to define or modify a manufacturing schedule, the *scheduling manager* needs to know the free space availability in the warehouses. This information is supplied by the *finished product warehouse manager*. The *master plan assistant* can also initiate this interaction in order to get information for the definition of the master plan.
- *Request Plan Feasibility*: The *master plan assistant* before communicating the master plan to the *production manager*, requests the feasibility plan from the *scheduling manager*. The *scheduling manager* has the up-to-date knowledge on factory capacity.
- *Coordinate Companies*: This interaction is executed in order to fulfill the goal to coordinate the manufacturing activities of ceramics companies. The *ceramic company manager* coordinates this interaction. This interaction starts a coordina-

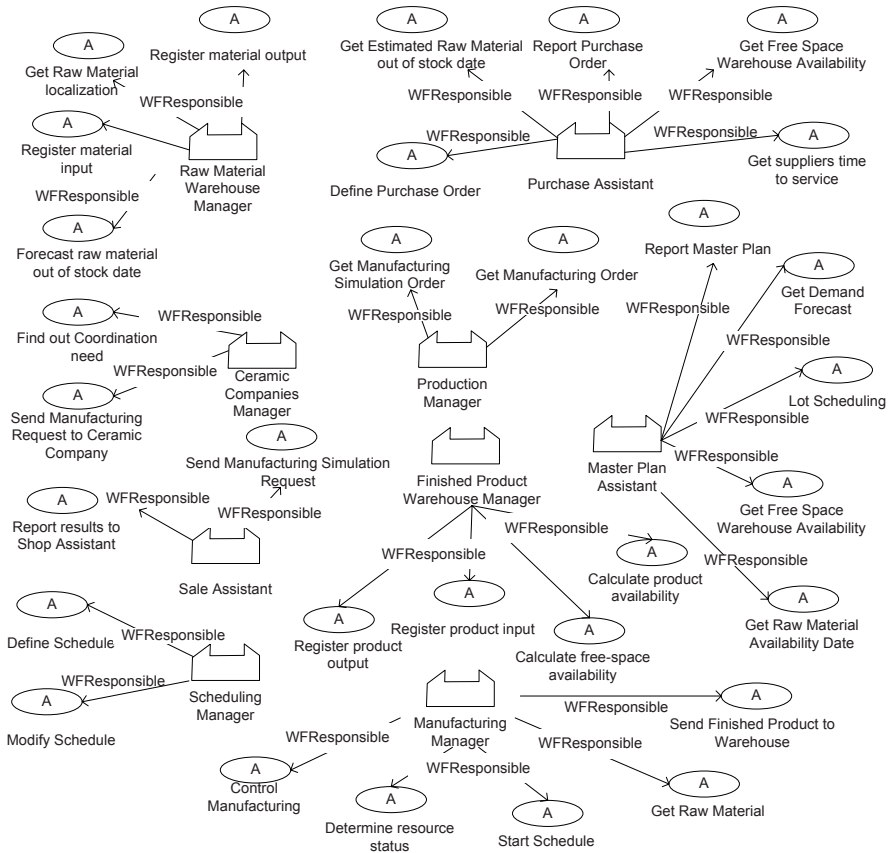


Fig. 8.7 Iteration 1, abstract tasks in the organization diagram

tion process among the different *production managers* of the ceramic companies of KCG.

- *Simulate Lot Scheduling*: The *sales assistant* cooperates with the *production manager* in order to get information on the feasible date for a client order delivery.

Activity A7 produces the *organization diagram* of Fig. 8.7. In this figure the role's abstract tasks for implementing the use cases are modeled. In Fig. 8.8 we can see the *tasks and goals diagram* produced by activity A8. In this diagram the abstract tasks associated with the system goals are depicted. The dependencies among the goals are also shown.

The next step in the analysis stage is the complex *identify holons* task (Chap. 6). In this step the system holons are identified, the roles are associated to these holons and a first holon specification is produced. Figure 8.9 shows an *organization*

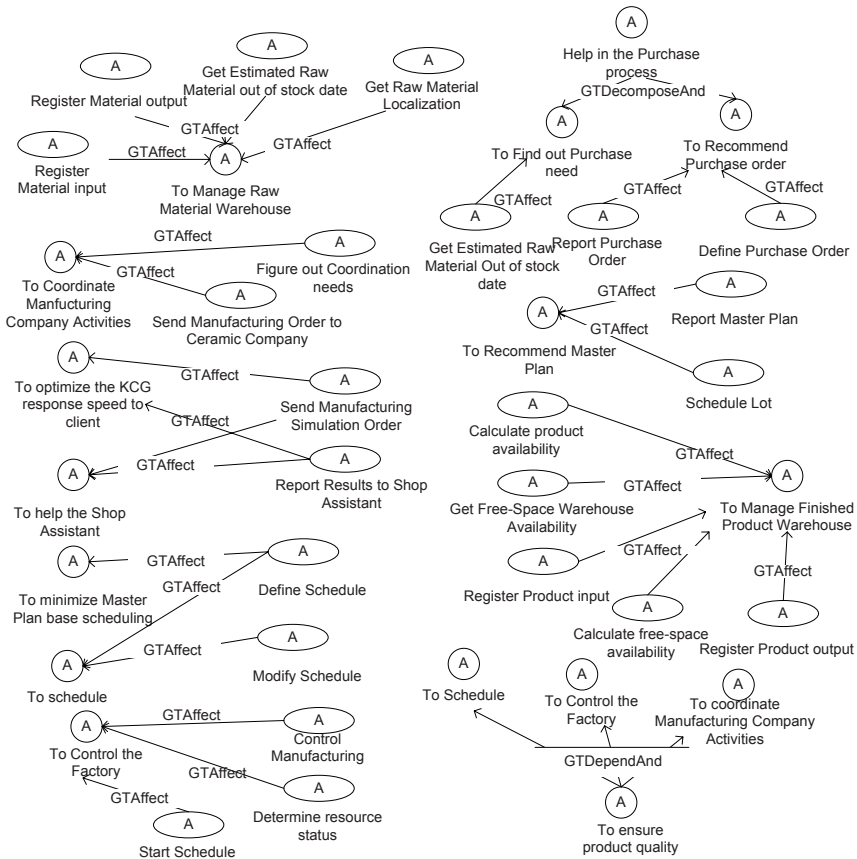


Fig. 8.8 Iteration 1, tasks and goals diagram

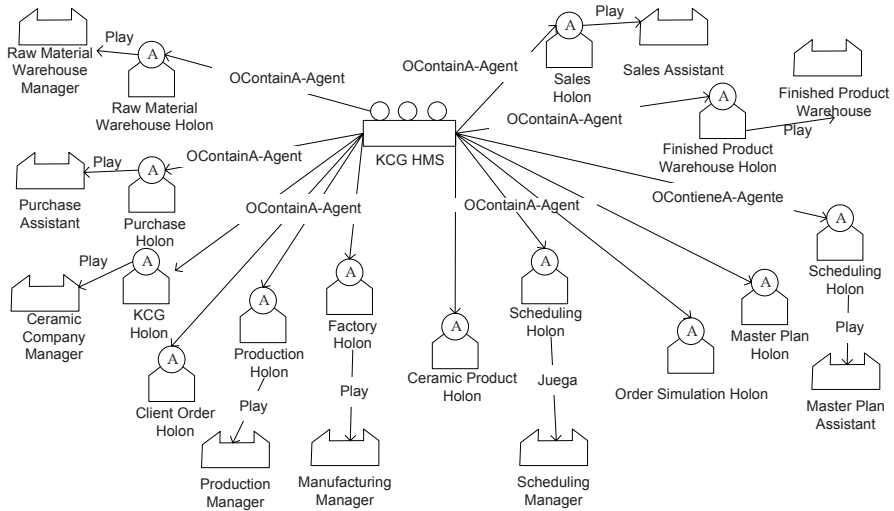


Fig. 8.9 Iteration 1, abstract agents and the organization diagram

diagram with the holons identified in activity *A9* using the *PROSA guidelines* (Chap. 6).

The holons we have identified for the KCG HMS are:

- *Purchase Holon*: A resource abstract agent with the processing capacity to define a raw material purchase order. In the holarchy of Fig. 8.7 it plays the *purchase assistant* role.
- *Raw Material Warehouse Holon*: A resource abstract agent with the master processing capacity to manage the raw material warehouse. In the holarchy of Fig. 8.7 it plays the *raw material warehouse manager* role.
- *Finished Product Warehouse Holon*: A resource abstract agent with the processing capability to manage the finished product warehouse. It plays the *finished product warehouse manager* role.
- *Factory Holon*: A resource abstract agent that provides manufacturing capacity. It plays the *manufacturing manager* role.
- *Scheduling Holon*: A resource abstract agent with computing capabilities to schedule manufacturing lots. It plays the *scheduling manager* role.
- *Sales Holon*: A resource abstract agent with the processing capability to manage the shop sales and direct sales from warehouses. It is also in charge of getting the date for a client order delivery. It plays the *sales assistant* role.
- *Production Holon*: A staff abstract agent in charge of managing the interaction between the *scheduling holon* and the *factory holon*. It plays the *production manager* role. There is a *production holon* for every ceramics company.
- *KCG Holon*: A staff abstract agent in charge of managing the KCG holarchy. It manages the interaction among all the holarchy holons and plays the *ceramics company manager* role.

- *Client Order Holon*: A work-order abstract agent that is created in the sales process. It is processed by the *production manager* using KCG’s factory resources.
- *Simulation Order Holon*: A work-order abstract agent that is created in the sales process when there is no product stock. This work order is processed by the *scheduling manager*.
- *Ceramic Product Holon*: A product abstract agent that maintains the design and specification data of a given ceramic product.
- *Master Plan Holon*: A product abstract agent that stores the master plan definition of the ceramic company to which it belongs.

When the holarchy holons are identified we have to complete an initial specification of these holons. This is done in activity *A10*. Figure 8.10 shows the *agent diagram* for the *factory holon*. We have assigned to it the goals and tasks associated with the *manufacturing manager* role. Moreover, following the *PROSA guidelines*, we have identified the new goals and tasks. Its information structure is also specified in terms of abstract beliefs.

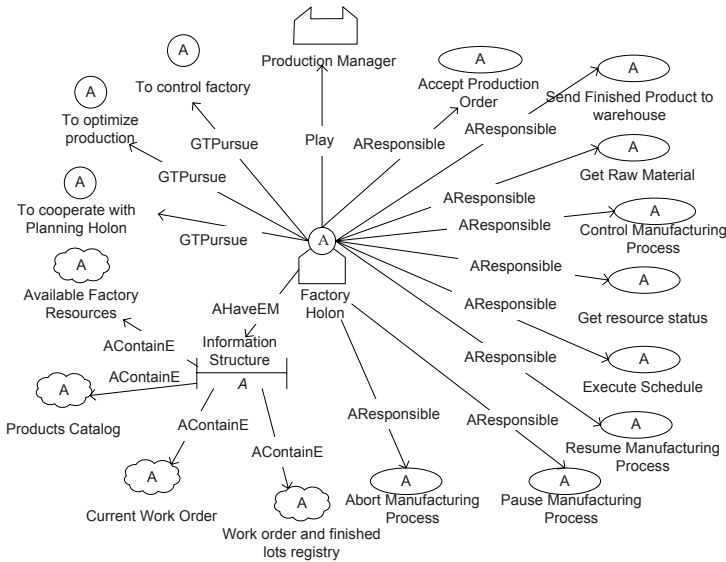


Fig. 8.10 Iteration 1, agent diagram, factory holon

Figure 8.11 shows the *agent diagram* of the *scheduling holon*, while Fig. 8.12 shows the *agent diagram* for the *order simulation holon*.

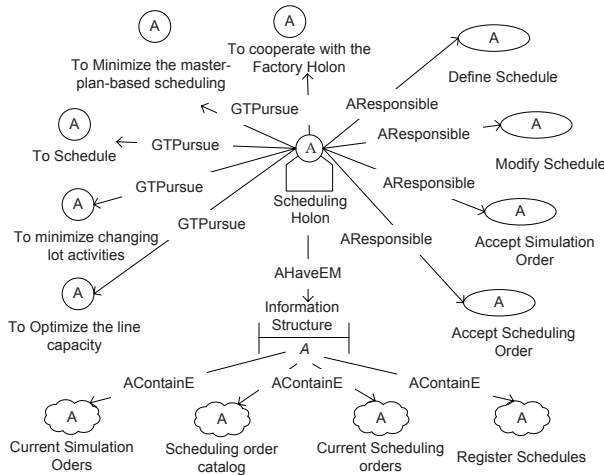


Fig. 8.11 Iteration 1, agent diagram, scheduling holon

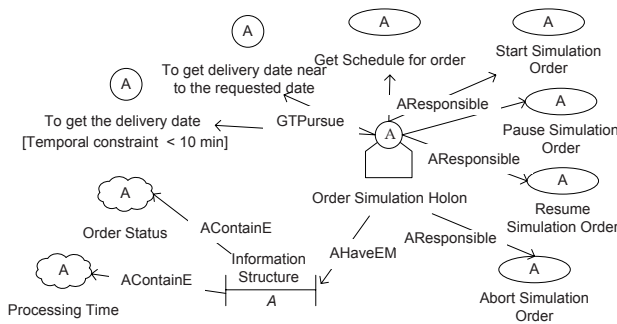


Fig. 8.12 Iteration 1, agent diagram, order simulation holon

The goal of activity *A11* is to specify the interactions identified in activity *A6*. Figure 8.13 shows the specification of the interaction *request raw material*. In this figure we can see the goals that the different interacting roles fulfill, the interchanged messages, and the message sequences.

In Fig. 8.14 we can see the *simulate lot scheduling* interaction that has a temporal constraint on the maximum duration of the interaction. The maximum time the *sales assistant* has to get a response from the *production manager* is 10 minutes. This constraint is translated into temporal conditions associated with the interaction messages. Moreover, the *simulation order holon* can be seen as the manager to obtain a feasible date for the client order delivery. The structure of this interaction is derived from *PROSA guideline* number 26 (Chap. 6).

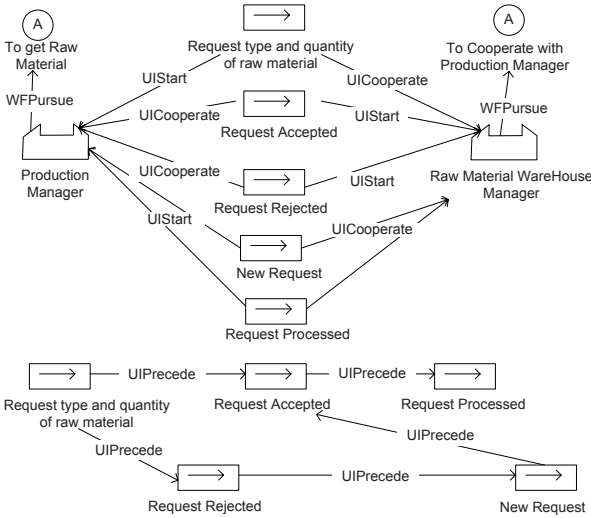


Fig. 8.13 Iteration 1, interaction diagram, request raw material

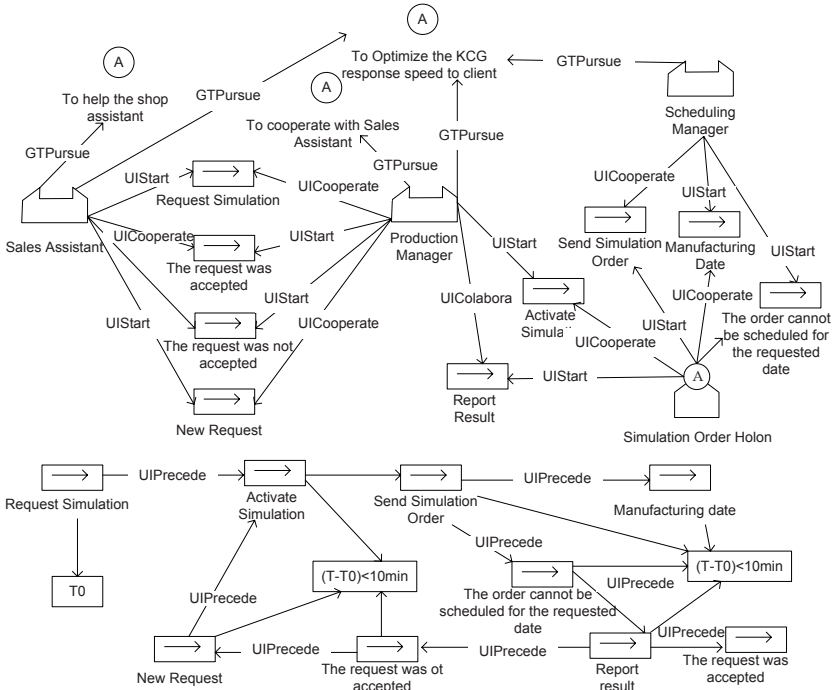


Fig. 8.14 Iteration 1, interaction diagram, simulate lot scheduling interaction

The new interactions identified in activity *A12* deal with *PROSA guidelines* 26 and 28. These guidelines help the engineer to identify interactions related to the *PROSA reference architecture*. In Fig. 8.15 we can see the interaction to process a manufacturing order. The client manufacturing order is received by the *sales assistant* and communicates the order to the *ceramics company manager*. The *ceramics company manager* determines the company in which the order will be processed. The *ceramics company manager* sends a message to the *production manager* in order to report the entrance of the new manufacturing order. The *production manager* activates a *client order holon* in order to manage the order manufacturing process. When the *production manager* receives the message *manufacturing request* from the *client order holon*, it informs the *scheduling manager* and the *factory manager* that the *client order holon* will start to interact with them.

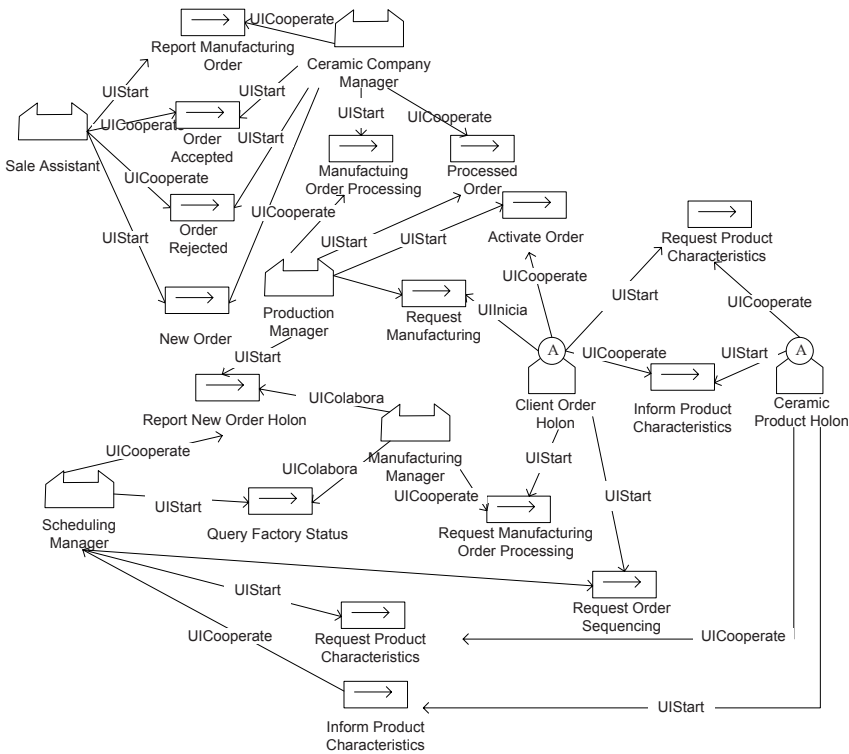


Fig. 8.15 Iteration 1, interaction diagram, manufacturing order processing interaction

The next activity in the holons specification is activity *A13*. In activity *A13* the engineer has to refine the *tasks and goals model* (Chap. 6). To do this the engineer may use the *PROSA guidelines* 14 to 17.

Figure 8.16 shows the decomposition of the goals and the identified relations among the goals and tasks of the *factory holon*. Figure 8.17 shows the task and

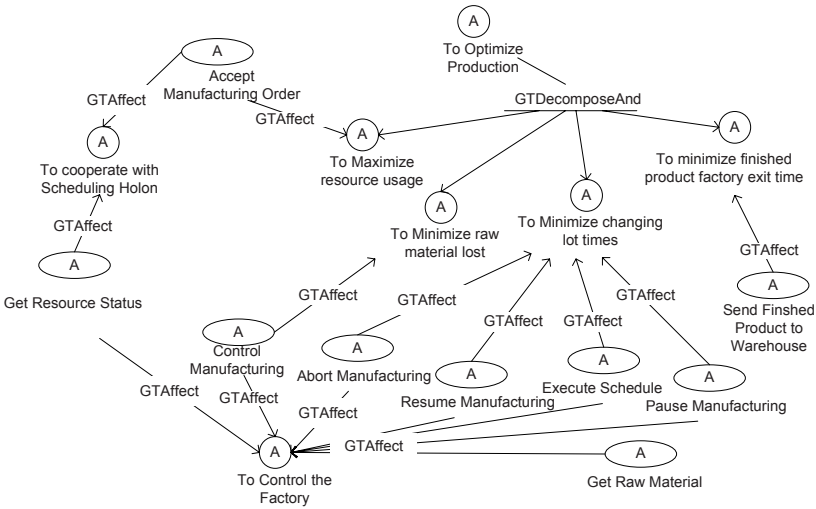


Fig. 8.16 Iteration 1, tasks and goals diagram, tasks and goals of the factory holon

belief relations of the *factory holon*. In this initial analysis stage the identified relations among goals, tasks and beliefs are of the GTAffect type (Chap. 5). This is so because in the initial phases there is not sufficient knowledge about the specialized types of relations (GTCreate, GTDestroy, GTModify, GTFail, GTSatisfy). On the other hand, Fig. 8.18 shows the decomposition of the *sales holon* task, *get client order delivery date*. In this figure it can be seen that the task is decomposed into *get warehouse availability* and *get date by simulation*. The first task is used to calculate the delivery date whenever the requested product is in stock. The second task is executed when there is no product availability in warehouses and the *lot scheduling simulation* interaction (Fig. 8.6) has to be initiated in order to forecast a feasible delivery date. The *get client order delivery date* task has a temporal constraint on its

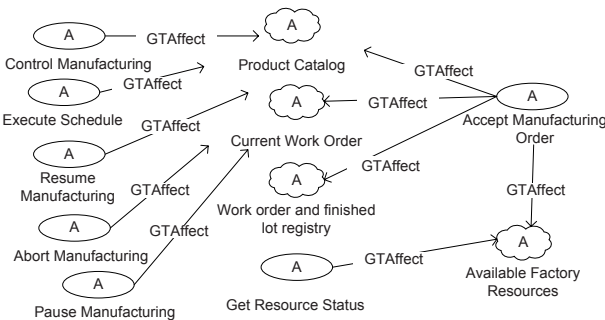


Fig. 8.17 Iteration 1, tasks and goals diagram, tasks and beliefs of the factory holon

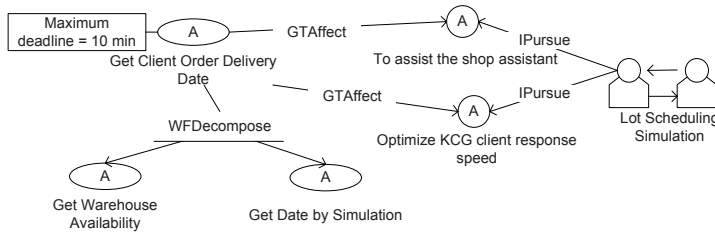


Fig. 8.18 Iteration 1, tasks and goals diagram, task temporal features

maximum deadline. This constraint is passed on to the sub-tasks and the interaction (Fig. 8.14) which is activated from the *get date by simulation* task.

In order to finish the first iteration of the analysis stage we have to *specify the environment relations* (Chap. 6). This complex task is decomposed into activities *A14, A15* and *A16*. These activities determine the external events that affect the system execution, applications and resources with that the system has to work. Figure 8.19 shows the external events⁴ that may affect the *sale holon* and the *raw material warehouse holon*. The applications are also depicted. *BD sale* and *BD warehouse* are databases with which the *sales holon* and the *raw material warehouse holon* have to interact. These databases are applications that are being used in the KCG company.

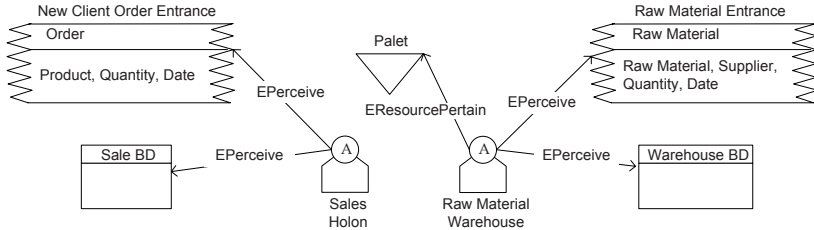


Fig. 8.19 Iteration 1, environment diagram, applications and events associated with the sales holon and the raw material warehouse holon

At this point iteration 1 is completed, so we have to decide whether to execute another analysis iteration. In order to do this we have to analyze the identified holons and decide, based on the *requirement* document, what the advantages are of decomposing one or more holons. Table 8.4 summarizes this analysis. In the new iteration we have to apply the different analysis activities to every non-atomic holon of iteration 1.

⁴ At this abstraction level of the KCG HMS, no external resource identified has temporal features. In lower abstraction levels the event temporal constraints will probably be identified. In this case, the bottom-up process of the design stage will propagate them to all of the system abstraction levels.

Table 8.4 Holons of iteration 1

Agent	Atomic	Comments
Raw-material warehouse holon	No	There are several warehouses of raw material and there are several processes and resources to control in a warehouse.
Purchase holon	No	In order to assist the purchasing process purchase distributed information is required.
KCG holon	Yes	The coordinating functions that implement this holon are not complex and can be controlled by a single agent.
Client order holon	No	The client order can be decomposed into parts that can be processed in different KCG factories.
Production holon	Yes	The coordinating functions that implement this holon are not complex and can be controlled by a single agent.
Factory holon	No	The factory has a set of resources and processes to control. Moreover, it can be organized in different ways.
Ceramic product holon	No	There is a ceramic product catalog and every product has its own different features.
Scheduling holon	No	In the scheduling process there are several processes and resources to control.
Simulation order holon	Yes	The simulation order is not decomposed into parts or sub-orders.
Master plan holon	Yes	The master plan is not decomposed into sub-plans.
Planning holon	Yes	The planning algorithm that implements this holon is centralized.
Finished product warehouse holon	No	There are several finished product warehouses and in a warehouse there are several products and resources.
Sales holon	No	Sales are executed in different stores and this process manages a set of processes that are executed in a distributed fashion.

8.2.2 Iteration 2

Every non-atomic holon of the previous iteration is transformed into an organization (or MAS). In This way all the t abstract entities that were associated to the holon are transformed into group entities (Chap. 5). The requirements of every organization are defined in the *requirements* document and the *analysis models* of the previous iteration. In the following paragraphs we illustrate the products of every iteration 2 analysis activity for the *scheduling* and *factory* holarchies.

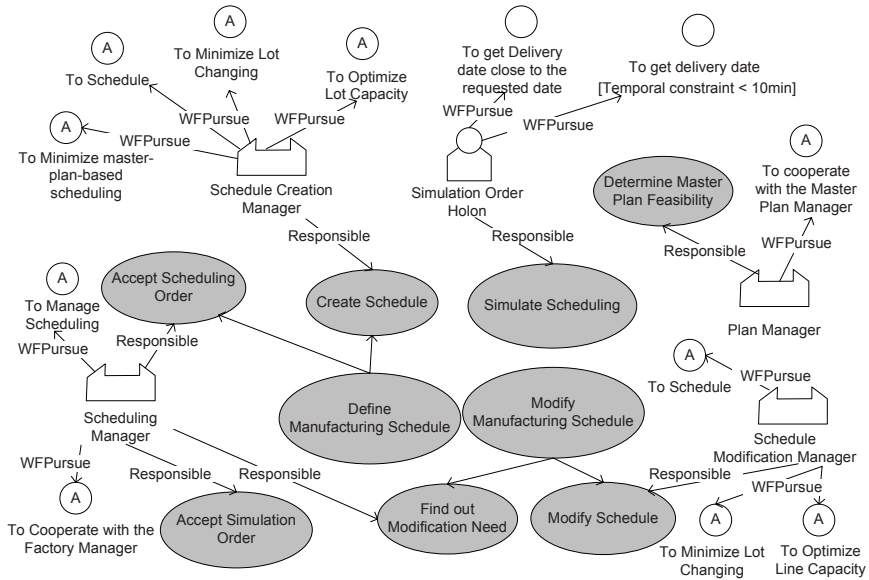


Fig. 8.20 Iteration 2, use cases, holons and goals of the scheduling holarchy

Figure 8.20 shows the use cases identified in the *scheduling* holarchy. The responsible role of every use case and the goals associated with the role are also depicted. On the other hand, Fig. 8.21 shows the use case diagram for the *factory* holarchy. In Fig. 8.20 we can see that the use cases *define manufacturing schedule* and *modify manufacturing schedule* identified in the previous iteration have been refined into simpler use cases. Moreover, we have identified new use cases in order to specify the task sequences that have to be executed in the *scheduling* holarchy. These use cases are *accept simulation order*, *simulate scheduling*, and *determine master plan feasibility*. In this figure we can also see the roles of the holarchy that are responsible for executing the use cases. The *scheduling manager* is responsible for coordinating the holarchy members, and it tries *to manage scheduling* and *to cooperate with the factory manager*. The *scheduling manager* has to *accept scheduling order*, *accept simulation order* and *find out modification needs*. In the scheduling processing scenarios we have identified the *schedule creation manager* and *schedule modification manager* roles. These two roles are associated with most of the holarchy goals. The *simulation order holon* (atomic holon) takes part in this holarchy and is responsible for the *simulate scheduling* use case. This holon is responsible for obtaining the available resources of the *scheduling* holarchy in order to process the simulation.

Figure 8.21 depicts the different use cases that have to be executed in the *factory* holarchy. We have decomposed the *control manufacturing process* use case of iteration 1 in the *get resource status* and *determine product status* use cases. Moreover, we have identified new use cases in the holarchy. The *manufacture* use case and its decomposition are derived from the *requirements* document. The use cases that

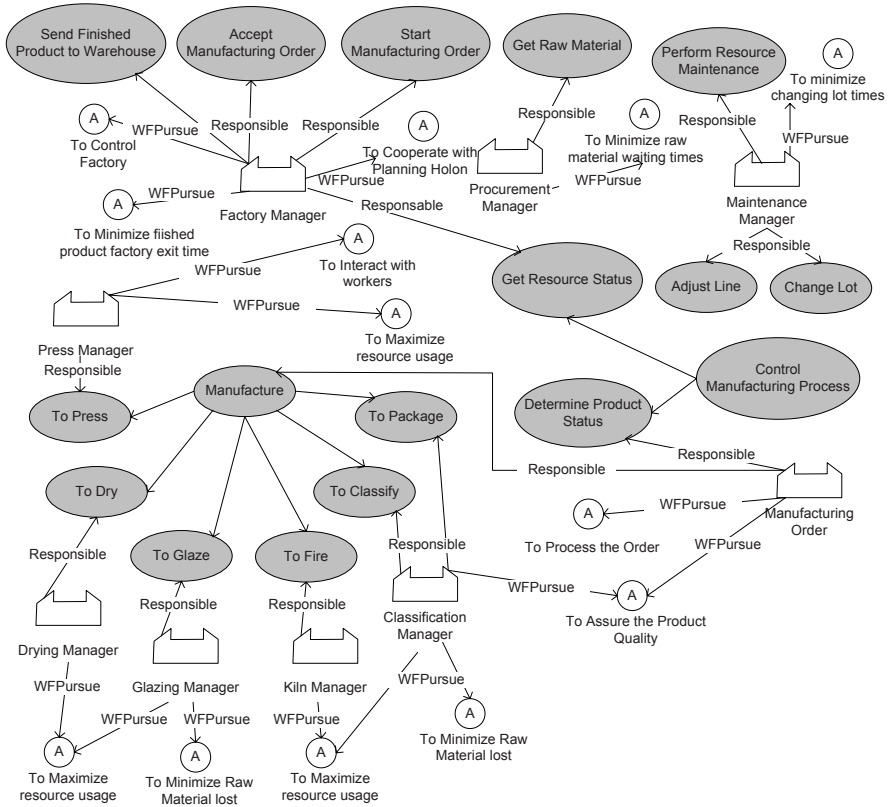


Fig. 8.21 Iteration 2, use cases, holons and goals of the factory holarchy

make up the *manufacture* use case represent all the steps for manufacture of ceramic tiles. The goals of the *press* use case are *to maximize the resource usage* and *to interact with the workers*. The second goal is derived from the *requirement* document, in which it is specified that, due to the physical and mechanical characteristics of the tile-pressing process, the press machine must be operated by workers. To this end the *press manager* has to be modeled as a controlling and a decision-making help interface with the press workers. The *factory manager* manages the holarchy and is the interface entity with other holarchies. The *manufacturing order* role manages a given manufacturing order process. This role has the following goals: *to process the order* and *to assure product quality*. The *maintenance manager* is in charge of the factory lot-changing activities: *adjust line*, *change lot*, and *perform resource maintenance* tasks. The *procurement manager* controls the raw-material procurement process.

Figure 8.22 shows the product of the *specify use case realization* task for the *factory* holarchy. In this figure we can see the tasks and goals associated to the *factory* holarchy roles. The *factory manager* has the goal *to cooperate with the scheduling*

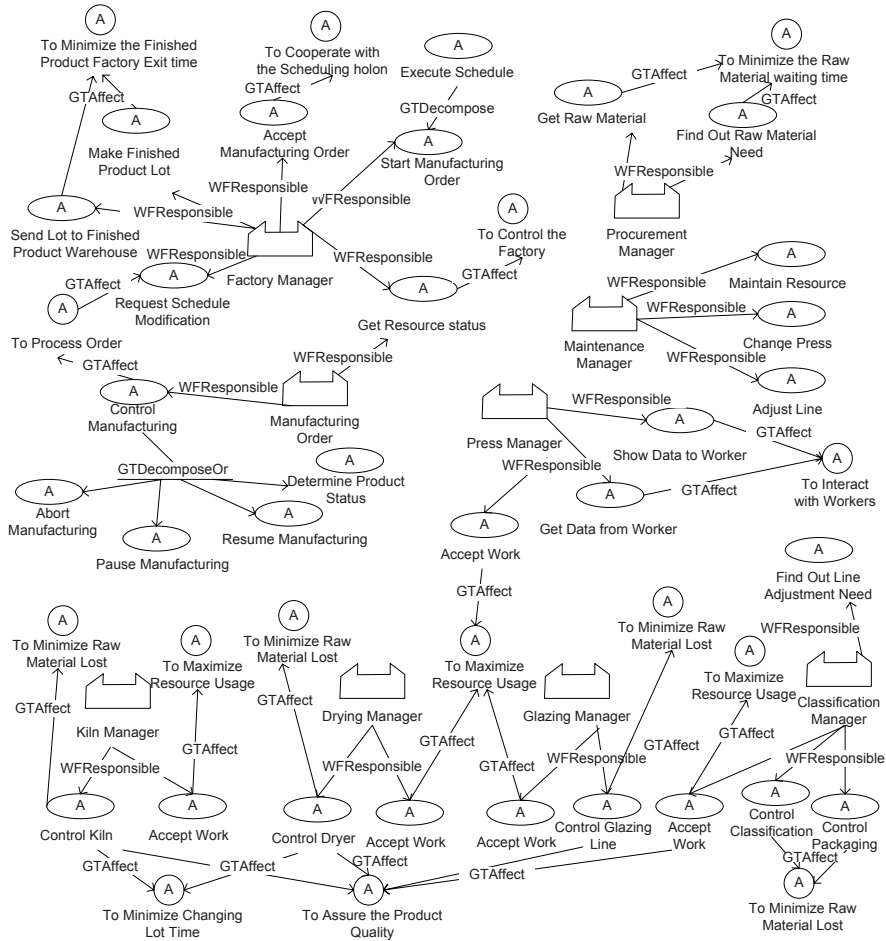


Fig. 8.22 Iteration 2, tasks and goals of the factory holarchy roles

holon, and the tasks *accept manufacturing order*, *execute schedule*, *send finished product to warehouse* and *get resource status*. The *execute schedule* task is decomposed into a set of instances of the *start manufacturing order* task due to the fact that a schedule is a set of manufacturing lots, and there is an order for each lot. The *control manufacturing* task is decomposed into *abort manufacturing*, *pause manufacturing*, *resume manufacturing* and *determine product status*. The role responsible for this task is the *manufacturing order*. The *procurement manager* fulfills the goal *to minimize raw-material waiting time*, to do so it is responsible of the tasks *find out raw-material needs* and *get raw material*. The different managers of the factory floor sections (kiln, drying, glazing, pressing and classification) implement the tasks for controlling the resources and the tasks to *accept work* in order to fulfill the goal *to maximize the resource usage*. In the controlling tasks, apart from trying to *min-*

imize the raw-material loss, the goal to minimize the changing lot time also has to be fulfilled. The section manager is in charge of the lot-changing resource tuning, except the press machines that are the responsibility of the *maintenance manager*. The *classification manager* is in charge of the *find out the line adjustment needs* task. All of the section controlling tasks fulfill the goal to assure product quality.

Figure 8.23 depicts the tasks and goals associated to the roles of the *scheduling* holarchy.

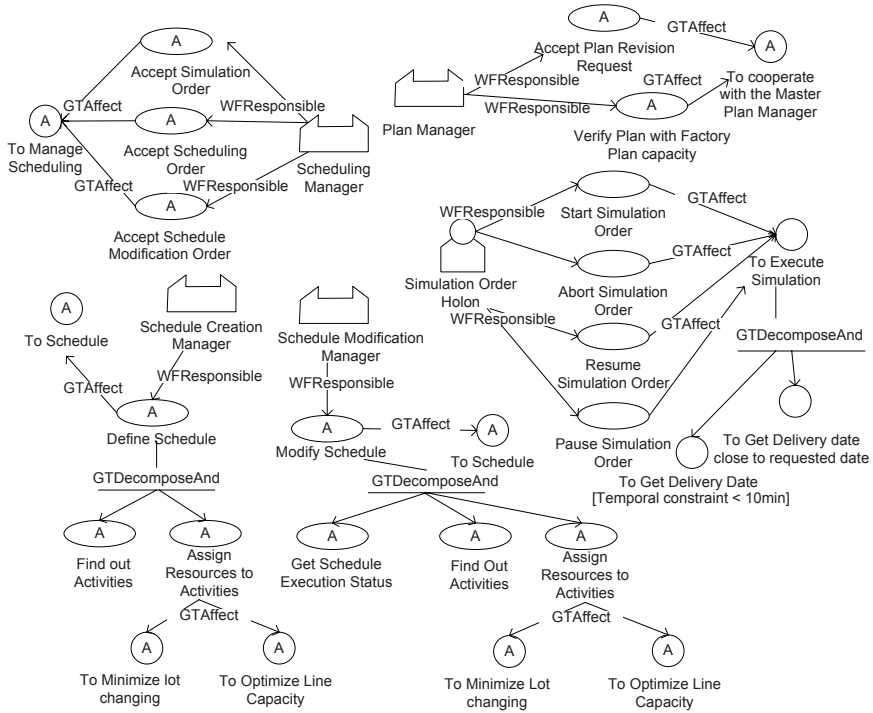


Fig. 8.23 Iteration 2, tasks and goals of the scheduling holarchy roles

In the next task of the analysis stage we have to identify the holons of the holarchy that is being analyzed. Some of these holons, the *simulation order holon* for example, had already been identified in the previous iteration. In this task we make use of the *PROSA guidelines*. In the *factory* holarchy we have applied physical and functional decomposition to discover which holon it is made up of. Thus, we have a set of holons for the different factory resources and the information processing that controls them, and another set of holons for management activities, specialized knowledge and process information management. For the *scheduling* holarchy, on the other hand, we have applied the functional decomposition. Figure 8.24 shows the roles assigned to the abstract agents of the *factory* holarchy, while Fig. 8.25 shows an *organization diagram* for the *scheduling* holarchy.

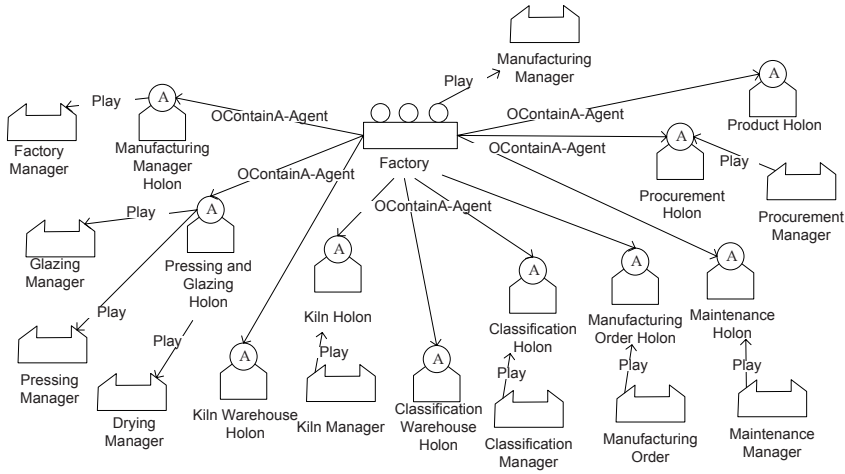


Fig. 8.24 Iteration 2, organizational structure of the factory holarchy

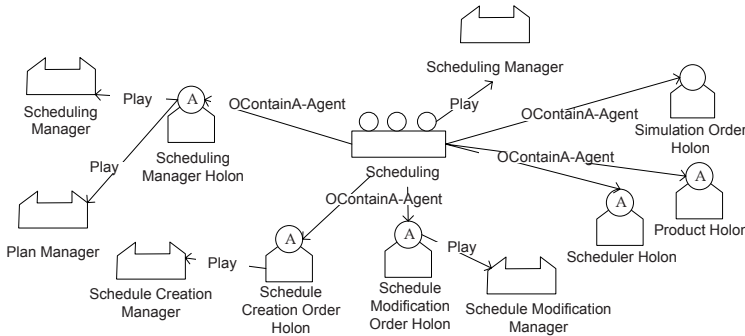


Fig. 8.25 Iteration 2, organizational structure of the scheduling holarchy

In Fig. 8.24 we can see the *factory* holarchy and its holons. These holons are:

- *Manufacturing Manager Holon*: This holon plays the *factory manager* role, manages the whole factory and interacts with the *scheduling manager* of the *scheduling* holarchy and the *production manager*.
- *Pressing and Glazing Holon*: Plays the *pressing manager*, *drying manager* and *glazing manager* roles. It is in charge of the management of the factory floor glazing line, and the different press machines in the factory. Every press machine will be controlled by one worker to which this holon will provide the press-related data. The details of the press holon will be studied in the next analysis iteration.
- *Kiln Holon*: Plays the *kiln manager* role and manages the tile firing process.
- *Classification Holon*: This holon plays the classification role and manages the tile classification process. It is also responsible for finding out the factory floor

resource adjustment needs when a changing lot is started. The management of the packaging process of the finished and classified product is also the responsibility of the *classification holon*.

- *Maintenance Holon*: It plays the *maintenance manager* role and manages the resource maintenance process whenever a resource failure occurs. It also makes requests to the maintenance workers for lot-changing activities in the press machines.
- *Procurement Holon*: This holon plays the *procurement manager* role and is in charge of the raw material procurement process when there is a factory floor raw-material need.
- *Classification Warehouse Holon*: Between the pressing/glazing lines and the kilns there is a temporary warehouse for the kiln input. This holon manages the available free space, the input and output of the product to the warehouse, and the means of transporting the product to the warehouse.
- *Manufacturing Order Holon*: Plays the *manufacturing order* role and is in charge of getting a manufacturing lot produced.
- *Product Holon*: This holon stores the information related to the manufacturing activities required for a product, product quality, product features, etc. In the *factory* holarchy it is used to find out the product quality requirements in the lot-changing activities, and as a sample in the classification process.
- *Resource Maintenance Order Holon*: This holon represents the resource maintenance activity requests and is in charge of getting these activities executed.

When all of the holarchy holons are identified we have to complete their specification. To this end we build an *agent diagram* for every holon. In this activity we use the *requirements* document and the *analysis models* produced in the previous iteration. Figure 8.26 shows the *manufacturing order holon*, its tasks, goals and associated mental entities.

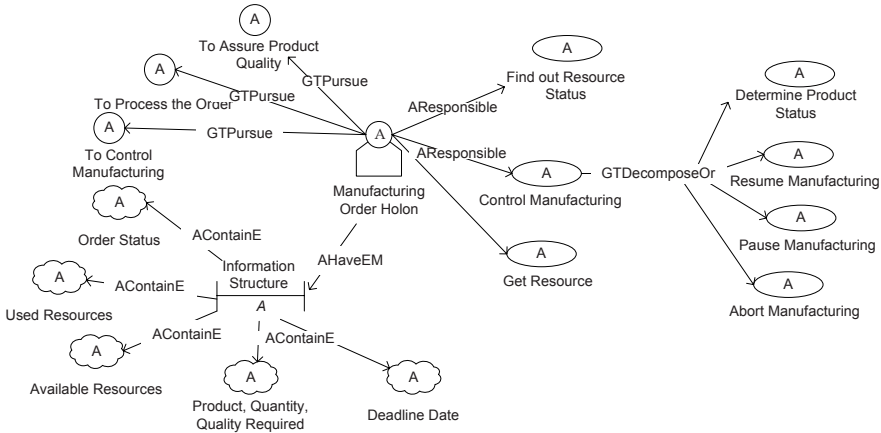


Fig. 8.26 Iteration 2, agent diagram of the manufacturing order holon

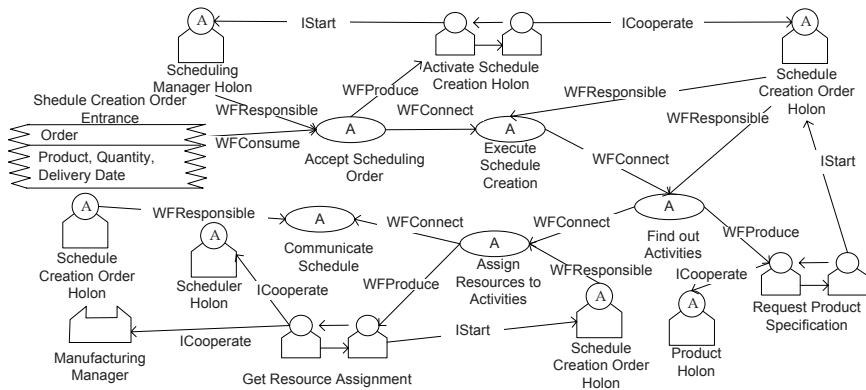


Fig. 8.27 Iteration 2, work flow for processing a scheduling order

In order to create a schedule, in the *scheduling* holarchy, a set of tasks must be executed. These tasks define the work flow from the schedule creation request entrance event until the *production manager* is informed of the defined schedule. When the *production manager* receives the new schedule it communicates with the *manufacturing manager* in order to request it to start the manufacturing process. Figure 8.27 shows this work flow in the *scheduling* holarchy, while Fig. 8.28 shows the associated work flow for processing a manufacturing order in the *factory* holarchy.

In the lot changing the line adjustment processes are executed. This process consists of the complete manufacturing of a sample lot. The produced product is analyzed in order to determine the adjustment activities required for quality assurance. When the quality obtained does not satisfy the quality required the adjustment process is initiated in order to tune the resources to get the desired product quality. Figure 8.29 shows the *adjust lines* interaction. Every resource holon executes

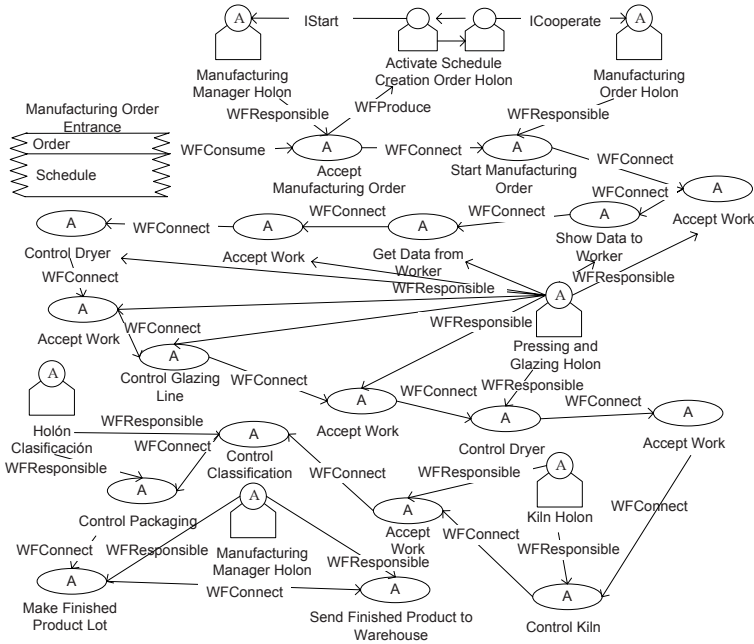


Fig. 8.28 Iteration 2, work flow for processing a manufacturing order

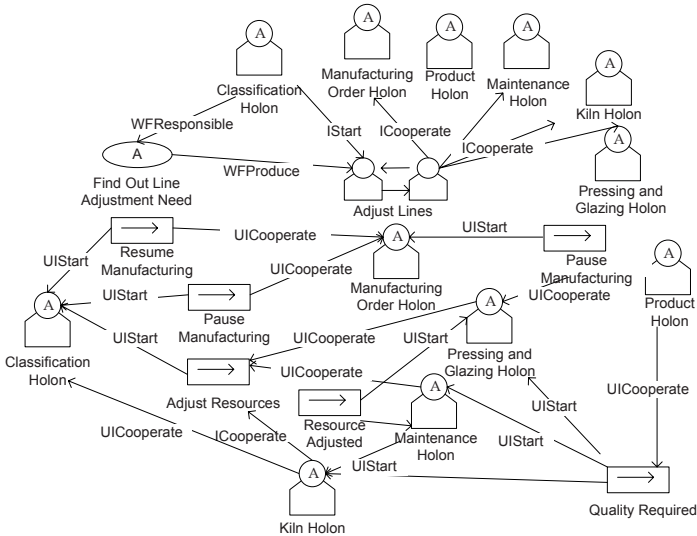


Fig. 8.29 Iteration 2, adjust lines interaction

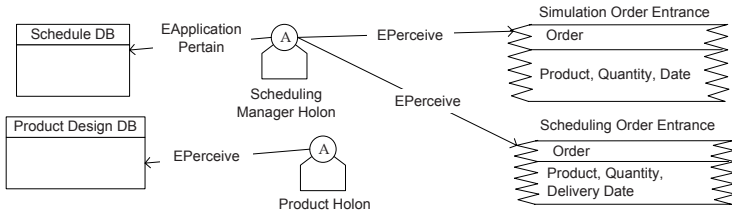


Fig. 8.30 Iteration 2, environment diagram of the scheduling holarchy

the required adjustment activities. The *pressing and glazing holon* is in charge of communicating the adjustment activity order to the press workers and to receiving confirmation from the workers when they are finished.

The *factory holarchy* works with the applications and external (non-autonomous) resources that are depicted in Fig. 8.30. Figure 8.31 shows the *environment diagram* of the *scheduling holarchy*.

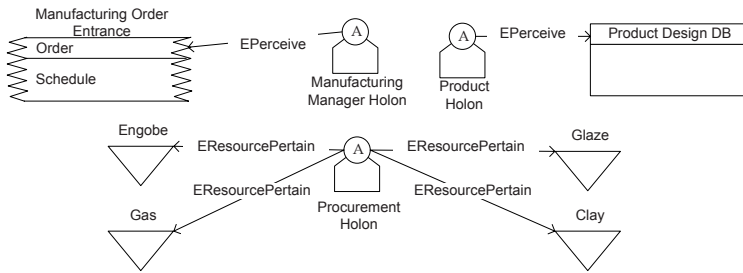


Fig. 8.31 Iteration 2, Environment Diagram of the Factory holarchy

At this point, iteration 2 is finished for the *scheduling* and *factory* holarchies, and we have to decide whether to apply another analysis stage iteration. To this end we study every identified holon to find out the convenience of decomposing some of these holons. Table 8.5 summarizes this analysis.

Table 8.5 Holons of iteration 2 from the scheduling and factory holarchies

Agent	Atomic	Comments
Scheduling manager holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Schedule creation order holon	Yes	The scheduling order is not decomposed into parts.
Schedule modification order holon	Yes	The scheduling modification order is not decomposed into parts.
Scheduler holon	Yes	Offers processing capability to assign tasks to factory resources (scheduling algorithms).
Product <i>i</i> holon	Yes	The ceramic product is not decomposed into sub-products.
Simulation order holon	Yes	In the previous iteration it was decided not to decompose it further.
Manufacturing manager holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Pressing and glazing holon	No	Controls and manages several resources and processes.
Kiln warehouse holon	No	Controls and manages several resources and processes.
Kiln holon	No	Controls and manages several resources and processes.
Classification warehouse holon	No	Controls and manages several resources and processes.
Classification holon	No	Controls and manages several resources and processes.
Manufacturing order holon	Yes	The manufacturing order is not decomposed into parts.
Maintenance holon	No	Controls and manages several resources and processes.
Procurement holon	Yes	Its functions can be implemented by a single agent.

8.2.3 Iteration 3

In the previous section we illustrated iteration 2 of the analysis stage for the *scheduling* and *factory* holarchies. In this section we continue with the analysis of the non-atomic holons in Table 8.5. These holons are from the *factory* holarchy.

Figure 8.32 shows the use cases, the responsible roles and the goals of the *pressing and glazing* holarchy. The use cases are derived from the *requirements* document and the study of the previous iteration. In this figure we can see that the use cases that deal with the management of the holarchy are the responsibility of the *pressing and glazing manager*. The use cases that deal with the manufacturing of particular instances of a product are the responsibility of the *manufacturing order* role. The same structure is repeated in the *kiln*, *classification*, *maintenance*, *kiln warehouse*, and *classification warehouse* holarchies. Nevertheless, these holarchy functions are adapted to the particular resource types of each factory floor section.

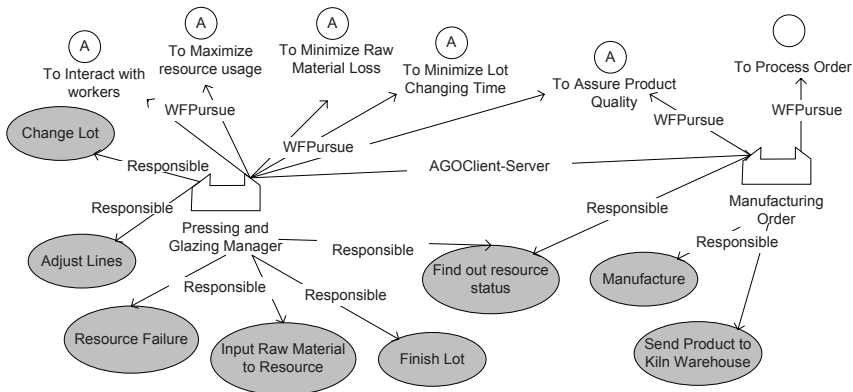


Fig. 8.32 Iteration 3, use cases, goals and roles of the pressing and glazing holarchy

In order to implement the uses cases of the holarchy, the responsible holons have to get the manufacturing resources and try to execute work flows with these resource holons. Figure 8.33 shows the *organization diagram* of the *Pressing and Glazing* holarchy. In this diagram we can see the holarchy holons, and the work flows that are executed to fulfill the holarchy goals. The *holarchy formation* work flow implements the task sequence to form a resource holons group that will process a manufacturing order. The *locate resource* work flow is used to locate the glazing and engobe resources on the line. This work flow is also used by the *manufacturing order holon* to send the manufacturing schedule to every resource holon. The *adjust lines* work flow implements the task sequence that are executed for the process of lot changing. The *stop*, *pause*, and *resume* work flows represent the management tasks of the pressing and glazing process. The *change resource* work flow models the tasks that are executed when a resource fails and it has to be changed by another resource in

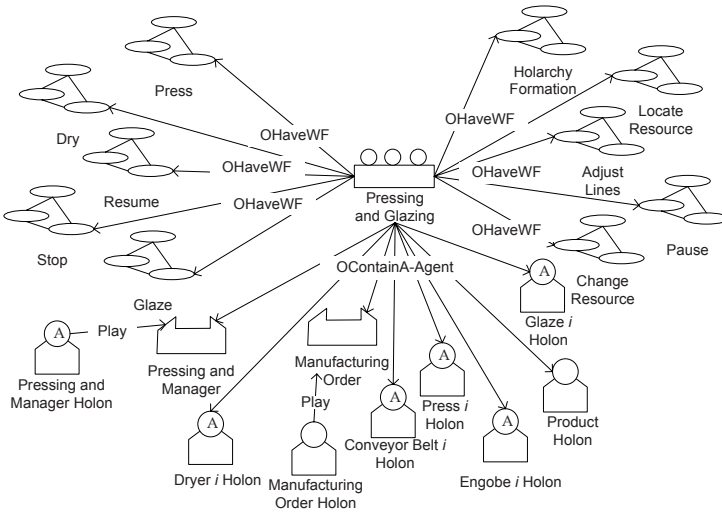


Fig. 8.33 Iteration 3, holons and work flows of the pressing and glazing holarchy

order to continue processing the order. On the other hand, the *press*, *glaze* and *dry* work flows represent the task sequences for the physical processing itself.

Figure 8.34 shows the specification of the work flow *holarchy formation* for the *pressing and glazing* holarchy. Similar work flows are repeated for the *kiln*, *classification*, and *maintenance* holarchies, when the *manufacturing order holon* searches

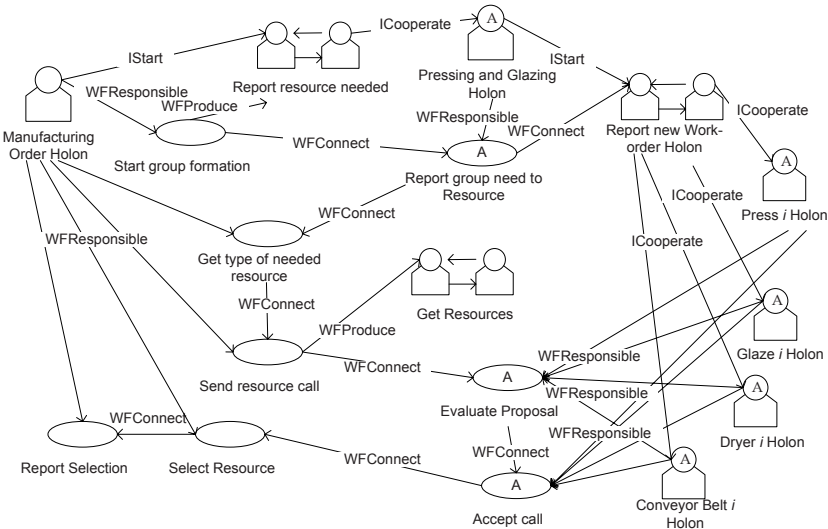


Fig. 8.34 Iteration 3, holarchy formation work flow of the pressing and glazing holarchy

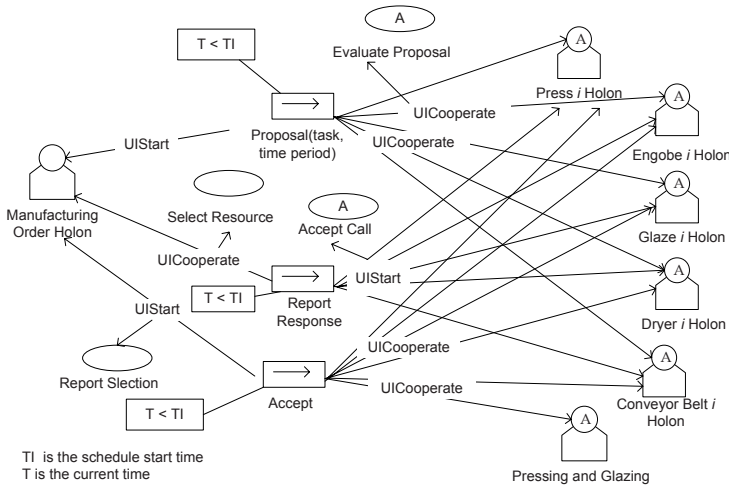


Fig. 8.35 Iteration 3, get resources interaction of the pressing and glazing holarchy

for resources to process the order. The *holarchy formation* work flow starts with the *start group formation* task of the *manufacturing order holon*. This task produces the *report resource needed* interaction via which the *manufacturing order holon* communicates the resource interaction required to process the order to the *pressing and glazing manager holon*. The next task in the work flow is the *pressing and glazing manager holon* responsibility. The task is *report group need to resource*. This task produces the *report new work-order holon* interaction. In this interaction the *pressing and glazing manager holon* notifies the resource holons that the *manufacturing order holon* is in charge of *finding out the types of resources needed*, in order to *send resource call* by means of the *get resources* interaction (see Fig. 8.35). The *evaluate proposal* and *accept call* tasks are executed by the resource holons in order to cooperate with the *manufacturing order holon*. The *select resource* and *report selection* tasks complete the *holarchy formation* work flow.

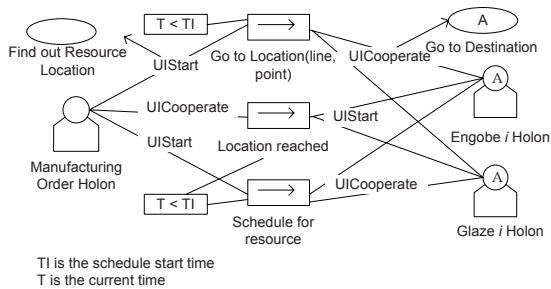


Fig. 8.36 Iteration 3, interaction for the locate resource work flow specification

Figure 8.36 shows the specification of the *locate resource* work flow, while Fig. 8.37 shows the *glaze* work flow.

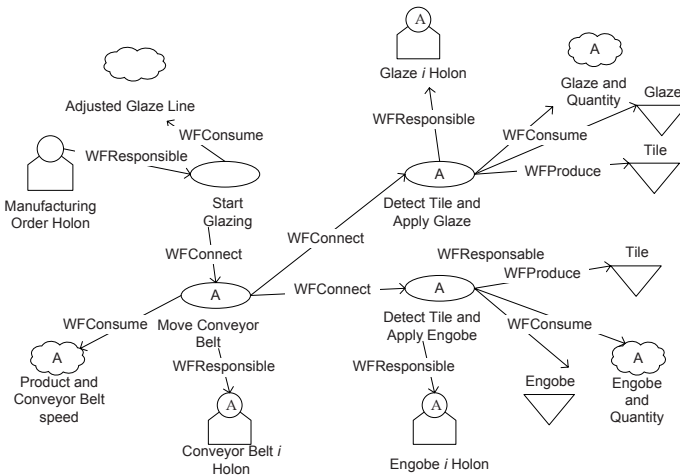


Fig. 8.37 Iteration 3, glaze work flow of the pressing and glazing holarchy

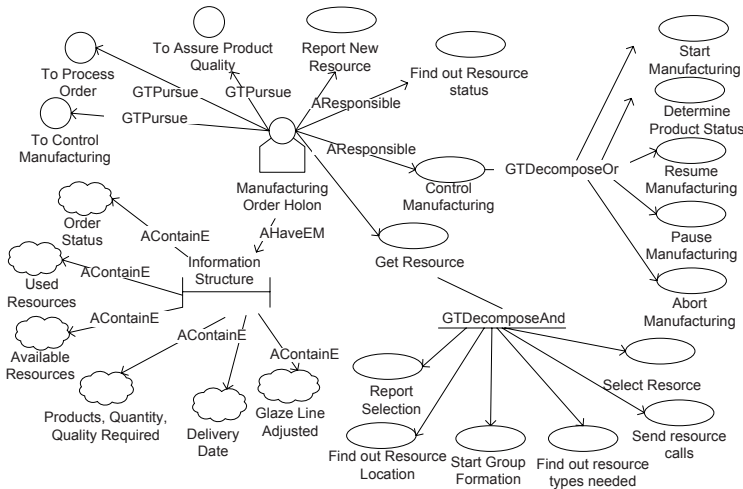


Fig. 8.38 Iteration 3, agent diagram of the manufacturing order holon

Figure 8.38 shows the *agent diagram* of the *manufacturing order holon* that has been extended with new tasks in the current interaction. Figure 8.39 shows the task decomposition of the *start manufacturing* and *resume manufacturing* tasks.

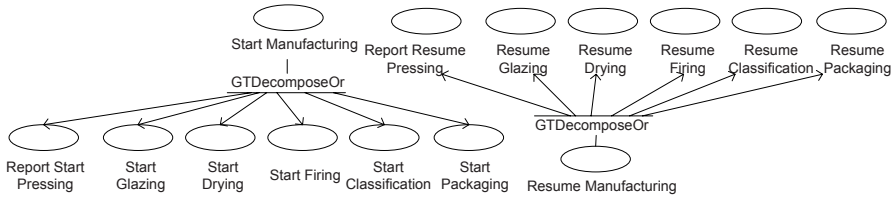


Fig. 8.39 Iteration 3, task decomposition of the manufacturing order holon

In Fig. 8.40 we can see the agent model of the *kiln i holon*. On a factory floor there could be kilns made by different manufacturers and so with different hardware controllers. Here, we illustrate the agent model of a continuous roller kiln. The tasks assigned to the *kiln i holon* have to deal with its structure. Therefore, we have a *zone temperature regulation* task, a *read thermostat* task. The *regulate the Roller speed* task to control the tile exposure time in each kiln zone, and the *synchronize roller speed with the input and output system*. The figure also shows the temporal constraints of the tasks and goals. Moreover, we can see the information mental entities of the *kiln i holon* for executing its tasks and fulfilling its goals.

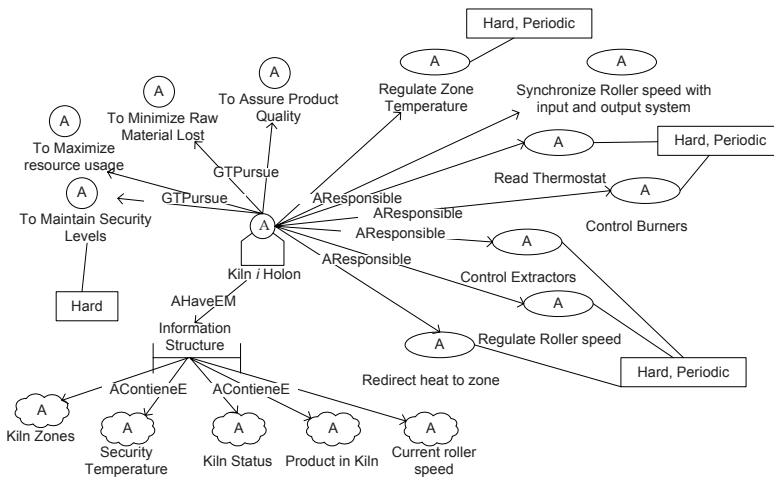


Fig. 8.40 Iteration 3, agent diagram of the *kiln i holon*

At this point iteration 3 is completed for the *pressing and glazing, kiln, kiln warehouse, classification warehouse, classification, and maintenance* holarchies. We have to decide whether to apply another analysis stage iteration. To this end we study every identified holon and find out the advisability of decomposing some of these holons. Tables 8.6 and 8.7 summarize this analysis.

Table 8.6 Holons of iteration 3 from the pressing and glazing, kiln, kiln warehouse, classification warehouse, classification and maintenance holarchies – Part 1

Agent	Atomic	Comments
Scheduling manager holon	Yes	The coordination functions that implements this holon are not complex and can be controlled by a single agent.
Press <i>i</i> holon	Yes	Implements worker interface functions. These functions are not complex and can be controlled by a single agent.
Dryer <i>i</i> holon	Yes	The dryer control is managed by a non-complex central hardware module and can be implemented by a single agent.
Engobe <i>i</i> holon	Yes	The engobe machine control is managed by a non-complex central hardware module and can be implemented by a single agent.
Glaze <i>i</i> holon	Yes	The glaze machine control is managed by a non-complex central hardware module and can be implemented by a single agent.
AGV <i>i</i> holon	Yes	The AGV control functions are not decomposed into sub-functions.
Conveyor belt <i>i</i> holon	Yes	The conveyor-belt controlling mechanism is implemented into a single hardware controller.
Kiln <i>i</i> holon	No	The kiln is managed from a single central controlling unit.
Kiln warehouse <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Classification <i>i</i> holon	Yes	The controller of the artificial vision classification machine is managed by a unique process.
Classification warehouse <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Packaging <i>i</i> holon	Yes	A single processor moves and controls the packaging robot arms.
Pressing and glazing manager <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Kiln manager <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.

Table 8.7 Holons of iteration 3 from the pressing and glazing, kiln, kiln warehouse, classification warehouse, classification and maintenance holarchies – Part 2

Agent	Atomic	Comments
Classification manager <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Resource maintenance order <i>i</i> holon	Yes	A maintenance order is not decomposed into parts.
Maintenance manager <i>i</i> holon	Yes	The coordination function that implements this holon is not complex and can be controlled by a single agent.
Product <i>i</i> holon	Yes	In previous iterations it was decided not to decompose it.
Manufacturing order holon	Yes	In previous iterations it was decided not to decompose it.

The analysis stage is completed because there is no new iteration required. In the following section the design stage of the Ceramic Tile Factory is presented.

8.3 Design

In this section we present the different *design models* produced by the ANEMONA design activities applied to the KCG case study. The goal of the holon design stage is to define the system architecture. The ANEMONA design stage has two major phases. In the first phase the *analysis models* of the analysis stage are completed with design details (Sect. 8.3.1). In the second phase the implementation platform features are specified (Sect. 8.3.2).

8.3.1 Holons Specification

In this section we show the diagrams produced in the design activities of the *refine holons specification* task. This phase starts with the atomic holons identified in the analysis stage, and gradually completes the specification of the higher level holons (bottom-up process).

The set of recursion level-0 holons, identified in the *analysis models*, are shown in Tables 8.8 and 8.9. In the first design iteration we complete the specification of the *agent model*, the *tasks and goals model* and the *environment model*. Figure 8.41 shows *agent diagrams* of the *manufacturing order holon*. In these diagrams we can see the different mental states of the holon at different execution times. The mental

Table 8.8 KCG atomic holons – Part 1

Holon	Holarchy
Raw material DB manager holon	Raw material warehouse management
Material output order holon	Raw material warehouse management
Material input order holon	Raw material warehouse management
Raw material warehouse holon	Raw material warehouse management
Clay silo holon	Raw material warehouse management
Clay warehouse holon	Raw material warehouse management
Spray dryer holon	Raw material warehouse management
Purchase data manager holon	Purchase assistance
Raw material order holon	Purchase assistance
KCG holon	Ceramic company coordination management
Client order part holon	Manufacturing order processing, and master planning
Production holon	Production coordination management, ceramic company coordination management, production order simulation, and manufacturing order processing
Manufacturing manager holon	Production coordination management, factory, production order simulation
Press <i>i</i> holon	Factory
Dryer <i>i</i> holon	Factory
Engobe <i>i</i> holon	Factory
Glaze <i>i</i> holon	Factory
AGV <i>i</i> holon	Factory
Conveyor belt <i>i</i> holon	Factory
Kiln <i>i</i> holon	Factory
Kiln warehouse <i>i</i> holon	Factory
Classification <i>i</i> holon	Factory
Classification warehouse <i>i</i> holon	Factory
Packager <i>i</i> holon	Factory
Pressing and glazing manager <i>i</i> holon	Factory
Kiln manager <i>i</i> holon	Factory
Classification manager <i>i</i> holon	Factory
Resource maintenance order holon	Factory
Maintenance manager <i>i</i> holon	Factory

Table 8.9 KCG atomic holons – Part 2

Holon	Holarchy
Procurement manager holon	Factory
Manufacturing order holon	Factory
Product <i>i</i> holon	Scheduling, and factory
Schedule creation order holon	Scheduling
Scheduling manager holon	Scheduling
Schedule modification order holon	Scheduling
Scheduler <i>i</i> holon	Scheduling
Simulation order holon	Order simulation, and sale assistance
Master plan holon	Master planning
Planning holon	Master planning
Finished product DB manager holon	Finished product warehouse
Finished product output order holon	Finished product warehouse
Finished product input order holon	Finished product warehouse
Finished product warehouse manager holon	Finished product warehouse
Transportation unit <i>i</i> holon	Finished product warehouse
Sale DB manager holon	Sales assistance
Sale order holon	Sales assistance

states of these diagrams represent the information knowledge and goals that make the *manufacturing order holon* execute tasks.

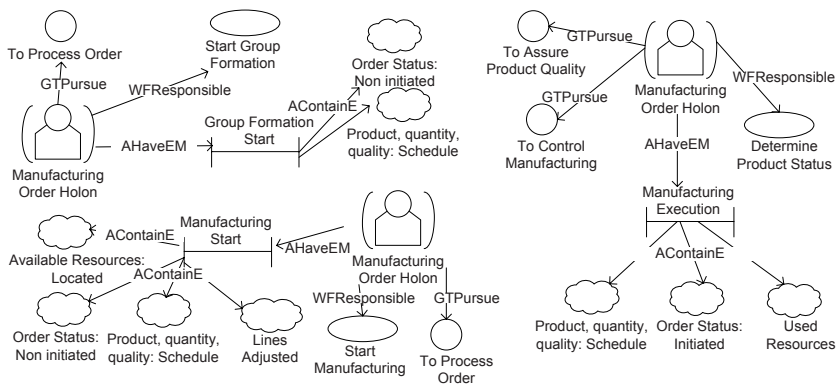


Fig. 8.41 Design, agent diagram of the manufacturing order holon

Figure 8.42 shows an *agent diagram* for the *kiln i holon*. In this diagram we can see the mental state *normal functioning*, in which the *kiln i holon* is in charge of executing the temperature control tasks for the different kiln zones. These tasks are: *read thermostat*, *regulate zone temperature*, *control extractor* and *control burners*. We can also see the task's temporal constraints, which are hard, real-time and periodic.

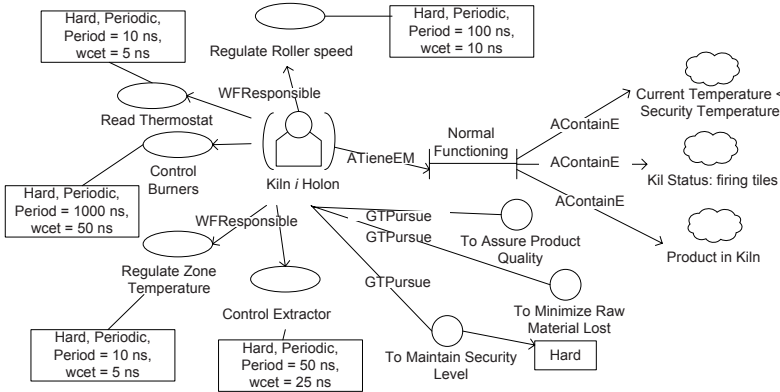


Fig. 8.42 Design, agent diagram of the kiln i holon

Figure 8.43 shows a *tasks and goals diagrams* of the *kiln i holon*. In this diagram we can see the relationship of the tasks and goals with the mental entities.

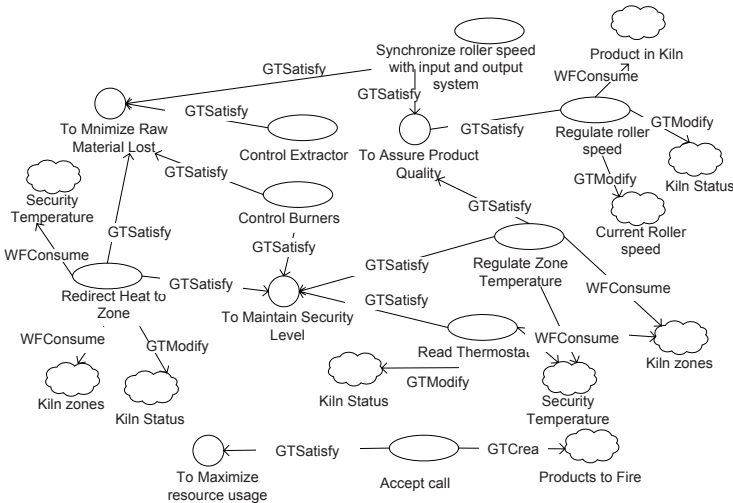


Fig. 8.43 Design, tasks and goals diagram of the kiln i holon

In Fig. 8.44 we can see the *environment diagram* of the *kiln i holon*. The external events that the holon perceives are *stop order entrance* and *product entrance*. The first event is the worker manual stop order. The second event has real-time characteristics.

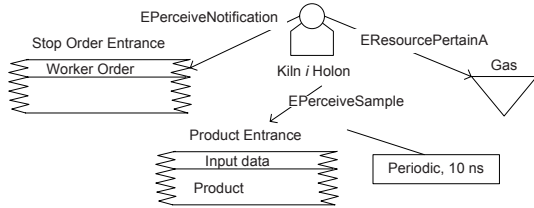


Fig. 8.44 Design, environment diagram of the kiln i holon

When the *design models* for the atomic holons are completed, the same steps have to be repeated for the holons of recursion level 1. By studying the *analysis models* we know that the holons of recursion level 1 are: *client order holon*, *ceramic product holon*, *pressing and glazing holon*, *kiln warehouse holon*, *kiln holon*, *classification warehouse holon*, and *maintenance holon*.

The *client order holon* represents a client order that can be divided into several parts or *manufacturing order holons*. These manufacturing orders can be executed in different KCG ceramic company factory floors (where the execution of an order is decided by the *KCG holon*). The *manufacturing order holon* represents the order part in the *scheduling* and the *factory* holarchies. Every *manufacturing order holon* is responsible for its own processing and one of them will be responsible for regis-

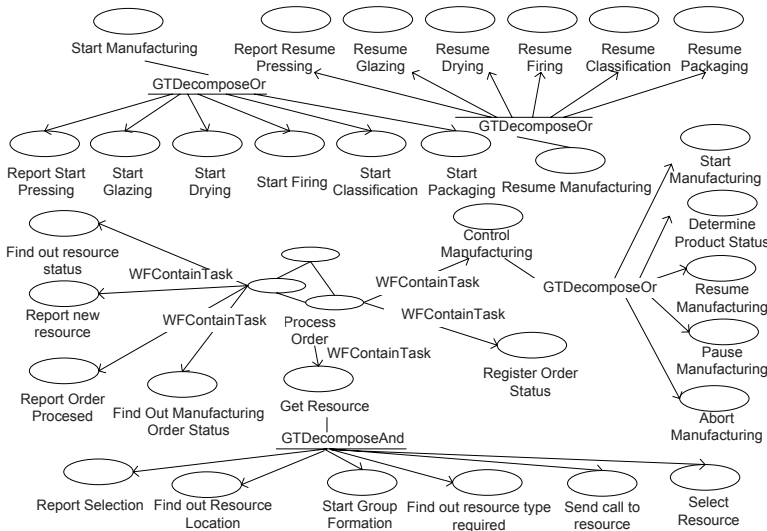


Fig. 8.45 Design, tasks and goals diagram of the client order holon

tering the time in which the client order is completed (that is, all the manufacturing orders that have been processed). Figure 8.45 shows the *tasks and goals diagram* of the *client order holon*. On the other hand, Fig. 8.46 shows the interaction among the *manufacturing order holons* of a client order for registering its processing status. In this diagram we can see the *order manager* and the *order part* roles.

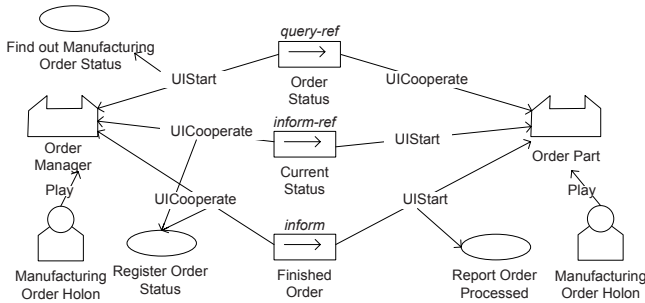


Fig. 8.46 Design, interaction diagram of the client order holon

The *pressing and glazing holon* represents the manufacturing process part for pressing, glazing and drying of ceramic tiles. The holons of this holarchy are the *pressing and glazing manager holon*, the resource holons of the pressing and glazing lines (see Fig. 8.33), and the *manufacturing order holon*. The interface of the *pressing and glazing holon* (tasks and goals) is depicted in Fig. 8.47. In this diagram we can see the decomposition of abstract tasks in terms of work flows and tasks of its constituent holons.

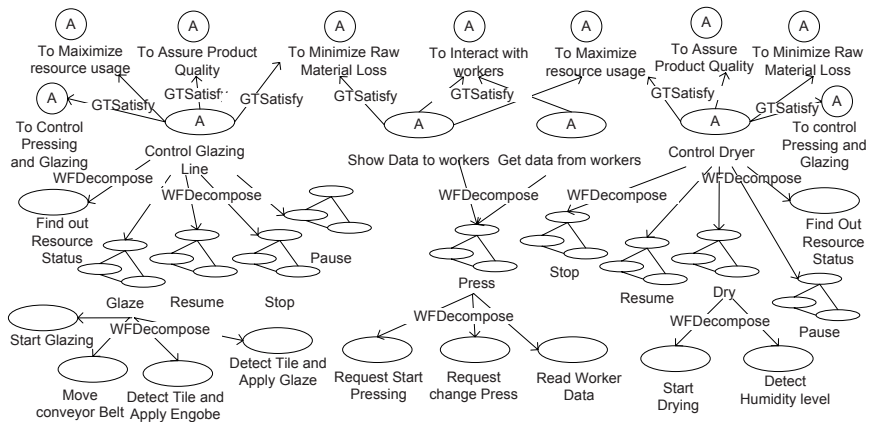


Fig. 8.47 Design, tasks and goals diagram of the pressing and glazing holon

We have to complete all of the internal interactions of the recursion level 1 holons. Figure 8.48 shows an *interaction diagram* for the *pressing and glazing holon*. In this figure we can see the interaction units and their execution sequence for the *resource location* interaction. Similarly, all of the interactions of the holarchy have to be completed with this information.

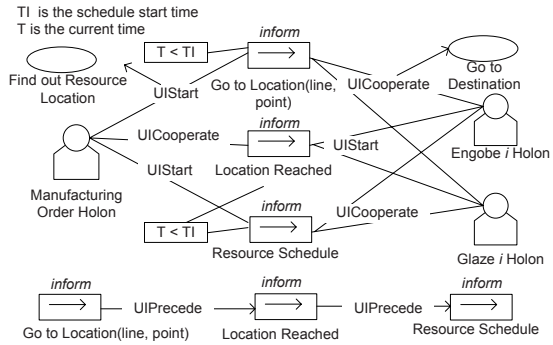


Fig. 8.48 Design, interaction diagram of the pressing and glazing holon

Figure 8.49 shows an *organization diagram* of the *pressing and glazing holon* with the social relations refined to subordinated relations and client–server relations.

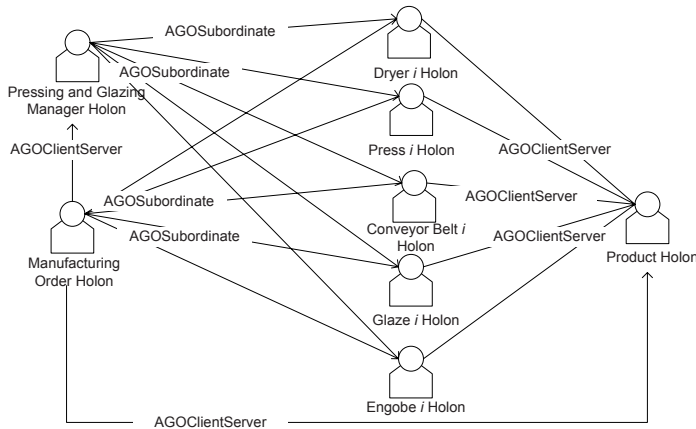


Fig. 8.49 Design, organization diagram of the pressing and glazing holon

Let's assume we have completed the recursion level-1 holons specification, and now we have to complete the *design models* for the holons of recursion level-2. Studying the *analysis models*, the only recursion level-2 holon is the *factory holon*.

Figure 8.50 shows the *factory holon* task decomposition in terms of the tasks of its constituent holons, while Fig. 8.51 shows an adjustment line interaction for the *factory holon*.

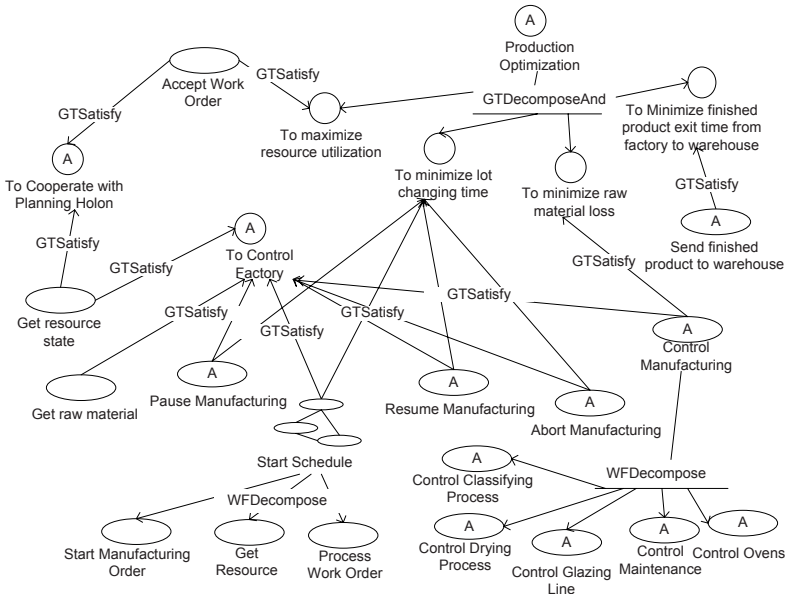


Fig. 8.50 Design, tasks and goals diagram of the factory holon

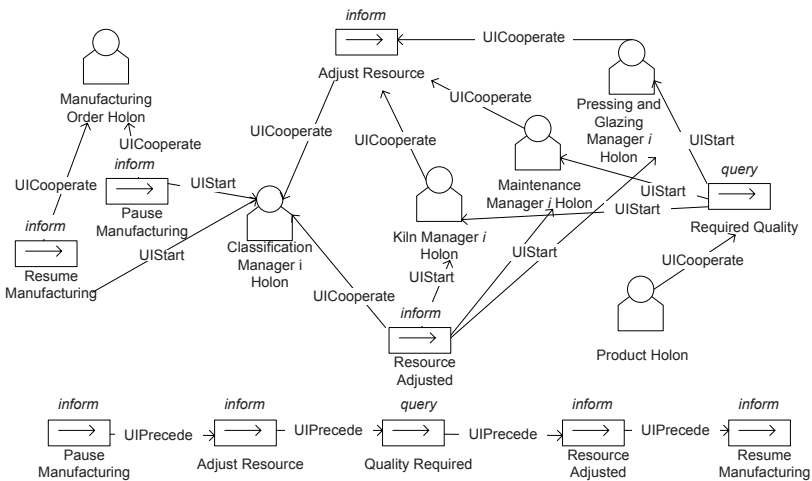


Fig. 8.51 Design, interaction diagram of the factory holon

Figure 8.52 shows the social relations among the holons of the *factory holon* holarchy. These relations are defined by studying the interactions identified in the analysis stage using the *PROSA guidelines*.

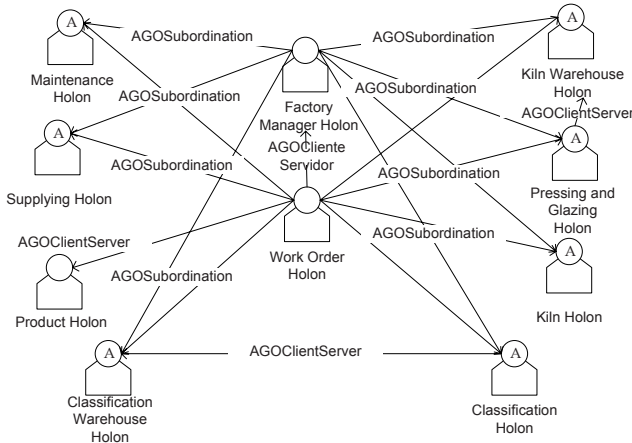


Fig. 8.52 Design, organization diagram of the factory holon

In the *analysis models* of the previous section we can see that the holons of recursion level-3 are: *raw material warehouse holon*, *finished product warehouse holon*, and *scheduling holon*. The *scheduling holon* has recursion level-3 because the *factory holon* (of level-2) takes part in this holarchy. The other holons of the *scheduling* holarchy are atomic holons. We have to refine the interactions in which the *factory holon* participates. Figure 8.53 shows the interaction to assign tasks to resources in which the *manufacturing manager* cooperates with the factory floor status data.

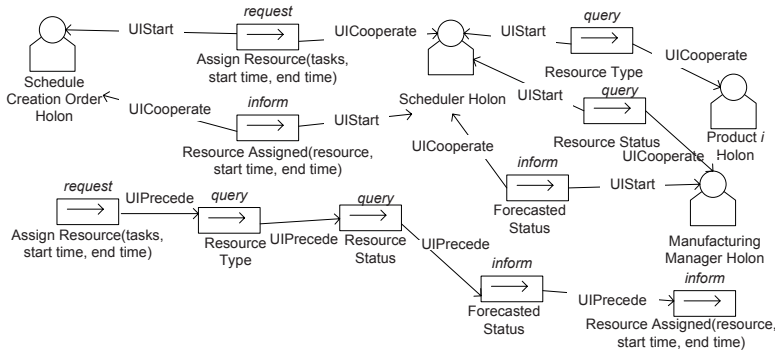


Fig. 8.53 Design, interaction diagram of the scheduling holon

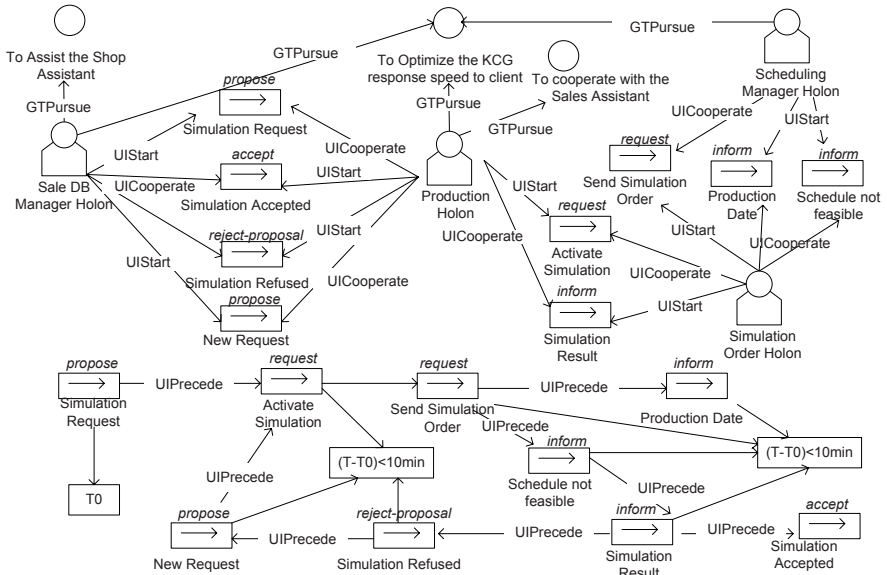


Fig. 8.54 Design, interaction diagram to process a simulation order

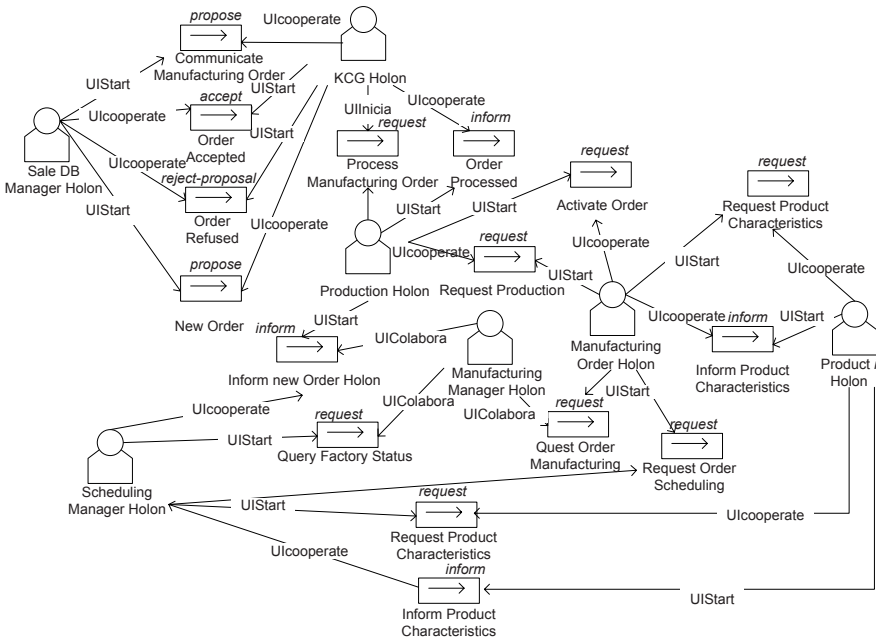


Fig. 8.55 Design, interaction diagram to process a client order

Finally, the holons of recursion level-4 are the *purchase holon* and the *sales holon*, because both of them participates in the *scheduling holon*. In Fig. 8.54 we can see the interaction diagram for the simulation order processing. Figure 8.55 shows the interaction diagram of KCG for processing a client order.

8.3.2 System Architecture

In this section we present the last phase of the ANEMONA design stage. The goal of this phase is to build the *system architecture* with the implementation platform details. To this end we use the *JADE* and *functional block guidelines* (Chap. 6).

Table 8.10 KCG identified agent platforms

Platform	Agents
KCG group	Factory
Sale <i>i</i>	KCG holon, production holon, purchase information management holon, raw material order holon, master plan holon, product <i>i</i> holon, and planning holon.
Scheduling <i>i</i>	Sale DB manager holon, sale order holon.
Factory <i>i</i>	Manufacturing manager holon, press <i>i</i> holon, dryer <i>i</i> holon, engobe <i>i</i> holon, glaze <i>i</i> holon, AGV <i>i</i> holon, conveyor belt <i>i</i> holon, kiln <i>i</i> holon, kiln warehouse <i>i</i> holon, classification <i>i</i> holon, classification warehouse <i>i</i> holon, packager <i>i</i> holon, pressing and glazing manager holon, kiln manager holon, classification manager holon, maintenance order holon, maintenance manager <i>i</i> holon, procurement holon, manufacturing order holon, and product <i>i</i> holon.
Raw material <i>i</i>	Raw material DB manager holon, material output order holon, material input order holon, raw material warehouse manager holon, clay silo holon, spray dryer holon.
Finished product <i>i</i>	Finished product DB manager holon, finished product output order holon, finished product input order holon, finished product warehouse manager holon, transportation unit <i>i</i> holon.

The first decision in this phase is to find out how the agent platforms are distributed. Using *JADE guidelines* 1 to 4 and the *requirements* document, we have defined the platforms list of Table 8.10. *KCG group* is the management platform of KCG. *Sale i* represents the agent platform that executes in each KCG store or warehouse and is in charge of the sales processes. *Scheduling i* is the platform for the scheduling management in a ceramic company. *Factory i* is the platform for the manufacturing management of every factory floor. *Raw material i* is the platform for the management of raw material in a given raw material warehouse. *Finished product i* is the platform for the finished product warehouse management.

The next step in the definition of the *system architecture* is to complete the *JADE agent templates* using *JADE guidelines* 5 to 12. Figure 8.56 shows the *JADE agent template* of the *simulation order holon*.

JADE Agent Template			
1. Agent ID	Simulation Order Holon	2. Platform	Scheduling KT, WD, MT
3. Services			
3.1. Name		3.2. Type	
Activate Simulation		external	
Simulation Data Initialization		external	
Simulation Data Modification		external	
Report Simulation Results		external	
4. Behaviors			
4.1. Name		4.2. Type	4.3. Implemented Services
Simulation Initialization		Sequential	Activate Simulation Simulation Data Initialization Simulation Data Modification
Simulation Controlling		Sequential	Start Simulation Pause Simulation Resume Simulation Stop Simulation
Simulation Ending		Sequential	Report Simulation Result
5. Ontology			
5.1. Name		5.2. Base Ontology	5.3. Schemas
Order			Client, Product, Quantity, Date
Scheduler			Scheduler ID
Sale Assistant			Sale Assistant ID
Simulation Result			Date, Ceramic Company
6. Communication			
6.1. Message	6.2. Type	6.3. Interaction	6.4. Participation Type
Activate Simulation	request	Scheduling Simul	Cooperator
Report Results	inform	Scheduling Simul	Initiator
Send Simul. Order	request	Scheduling Simul	Initiator
Production Date	agree	Scheduling Simul	Cooperator
Simulation Order Unfeasible	agree	Scheduling Simul	Cooperator
7. Does it have a physical processing part?		No	

Fig. 8.56 System architecture, JADE agent template of the simulation order holon

For every holon with a physical processing part we have to complete its *function block interface specification*. Figure 8.57 shows the specification of the physical processing part of the *conveyor belt i holon task: divert tile to conveyor belt*. In this figure we can also see the diverter *function block diagram*.⁵ Figure 8.58 shows the informal algorithm specification of the diverter controller inputs and outputs.⁶

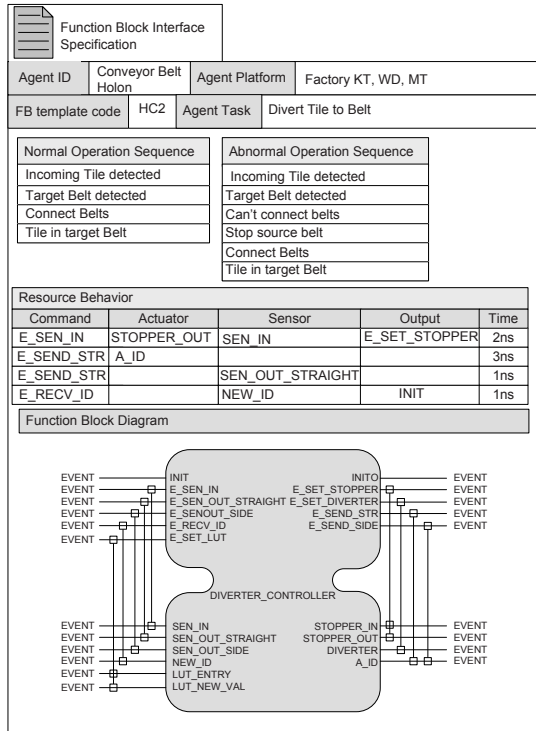


Fig. 8.57 System architecture, function block interface specification of the task divert tile to conveyor belt

The *UML deployment diagram* [105] of Fig. 8.59 shows the *KT factory agent platform* and its network. In this diagram we can see the *JADE agents' assignment* to the network nodes. The holons with a physical processing part will have their function blocks in the same physical device, while the information processing part (*JADE agent*) will be in the network node of Fig. 8.59. On every agent platform of the *KCG HMS* there is a server and all of the platform servers are connected to each other.

⁵ IEC-61499 Specification of the *DIVERTER_BC440* : *BC440* device.

⁶ The syntax used in the algorithm specification is the IEC-61499-1.

```

FUNCTION_BLOCK DIVERTER_CONTROLLER (* Controls the diverter functionality *)
EVENT_INPUT
  INIT; (* init request *)
  E_SEN_IN WITH SEN_IN; (* the input tyle sensor has changed *)
  E_SEN_OUT_STRAIGHT WITH SEN_OUT_STRAIGHT; (* the frontal tile proximity sensor has changed *)
  E_SEN_OUT_SIDE WITH SEN_OUT_SIDE; (* the lateral tile proximity sensor has changed *)
  E_RECVD_ID WITH NEW_ID; (* a tile is being received from the previous station *)
  E_SET_LUT WITH LUT_ENTRY,LUT_NEW_VAL; (* changing line request *)
END_EVENT
EVENT_OUTPUT
  INITO; (* init confirmation *)
  E_SET_STOPPER WITH STOPPER_IN,STOPPER_OUT; (* set the stop tile actuator *)
  E_SET_DIVERTER WITH DIVERTER; (* set the divert tile actuator *)
  E_SEND_STR WITH A_ID; (* send the tile ID to the next frontal station *)
  E_SEND_SIDE WITH A_ID; (* send the tile ID to the next lateral station *)
END_EVENT
VAR_INPUT
  SEN_IN : BOOL; (* tile input sensor *)
  SEN_OUT_STRAIGHT : BOOL; (* frontal proximity tile sensor*)
  SEN_OUT_SIDE : BOOL; (*lateral proximity tile sensor*)
  NEW_ID : SINT; (*tile id that has left the station*)
  LUT_ENTRY : SINT; (*list entry*)
  LUT_NEW_VAL : BOOL; (*the value to insert in the list*)
END_VAR
VAR_OUTPUT
  STOPPER_IN : BOOL; (* extension value of the stop tile sensor *)
  STOPPER_OUT : BOOL; (* contension value of the stop tile sensor *)
  DIVERTER : BOOL; (*diverter actuator value;0=front; 1=lateral *)
  A_ID : SINT; (*the ID of the tile which is being processed in the diverter station *)
END_VAR
FBS
  DIV_LOGIC : DIVERTER_LOGIC;
  FIFO : FIFO_SINT;
  LUT : BOOL_LUT;
  SWITCH : E_SWITCH;
END_FBS
EVENT_CONNECTIONS
  INIT TO FIFO.INIT ;
  FIFO.INITO TO LUT.INIT ;
  LUT.INITO TO INITO ;
  DIV_LOGIC.E_CALC DIV TO FIFO.POP ;
  FIFO.POPED TO LUT.REQ ;
  LUT.CNF TO E_SET_DIVERTER ;
  LUT.CNF TO DIV_LOGIC.E_DIV_SET ;
  DIV_LOGIC.E_STOPPER_IN TO E_SET_STOPPER ;
  DIV_LOGIC.E_SEND_ID TO SWITCH.EI ;
  SWITCH.EO0 TO E_SEND_STR ;
  SWITCH.EO1 TO E_SEND_SIDE ;
  E_SEN_IN TO DIV_LOGIC.E_SEN_IN ;
  E_SEN_OUT_STRAIGHT TO DIV_LOGIC.E_SEN_OUT_STRAIGHT ;
  E_SEN_OUT_SIDE TO DIV_LOGIC.E_SEN_OUT_SIDE ;
  E_RECVD_ID TO FIFO.ADD ;
  E_SET_LUT TO LUT.SETENTRY ;
END_CONNECTIONS
DATA_CONNECTIONS
  I TO FIFO.QI ;
  FIFO.QO TO LUT.QI ;
  I0 TO LUT.TABLESIZE ;
  FIFO.TOPPOP TO LUT.ENTRY ;
  FIFO.TOPPOP TO A_ID ;
  DIV_LOGIC.STOPPER_IN TO STOPPER_IN ;
  DIV_LOGIC.STOPPER_OUT TO STOPPER_OUT ;
  LUT.VAL TO DIVERTER ;
  LUT.VAL TO SWITCH.G ;
  SEN_IN TO DIV_LOGIC.SEN_IN ;
  SEN_OUT_STRAIGHT TO DIV_LOGIC.SEN_OUT_STRAIGHT ;
  SEN_OUT_SIDE TO DIV_LOGIC.SEN_OUT_SIDE ;
  NEW_ID TO FIFO.NEWVAL ;
  LUT_ENTRY TO LUT.NEWPOS ;
  LUT_NEW_VAL TO LUT.NEWVAL ;
END_CONNECTIONS
END_FUNCTION_BLOCK

```

Fig. 8.58 System architecture, specification of the diverter controller inputs and outputs

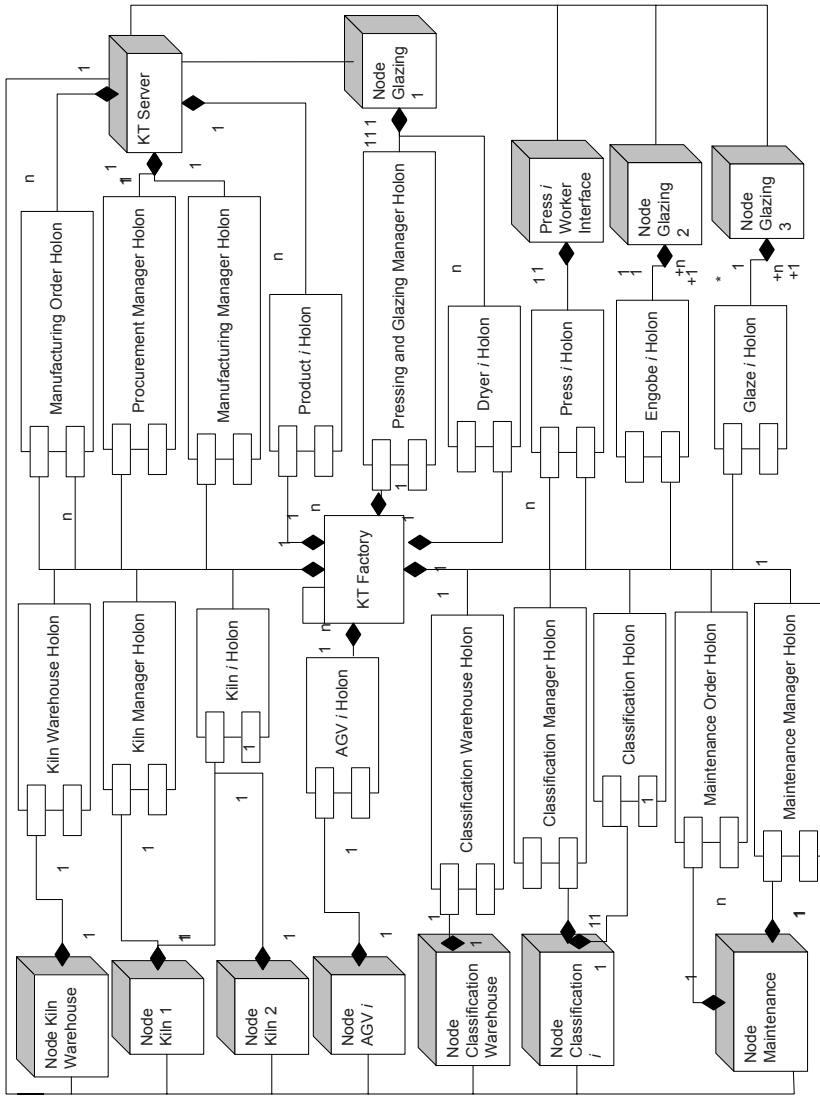


Fig. 8.59 System architecture, deployment diagram of the factory agent platform

8.4 Conclusions

In this chapter we have presented a case study from a ceramic tile factory. In this case study we have demonstrated ANEMONA's applicability to a real industrial case study. We have presented the analysis and design of a HMS for the management of several business and production processes of a group of ceramics companies. The development of the HMS by abstraction levels, guided by the abstract agent notion and the ANEMONA guidelines, help the software engineer to focus on the important features and requirements of the problems. Therefore, HMS development is a simple incremental process of identification and specification of holons.

Chapter 9

Conclusions

This book presented ANEMONA a multi-agent methodology for holonic manufacturing systems (HMS). ANEMONA is a methodology based on HMS modeling requirements. Its central modeling element is the abstract agent notion with a mixed top-down and bottom-up process for HMS development. ANEMONA helps the software engineer with clear and specific HMS modeling guidelines. This chapter concludes the book by reviewing its content and pointing to possible future works.

9.1 Review

First, Part I set the background for this book by reviewing the state-of-the-art in Holonic Manufacturing Systems (Chap. 2) and introducing the abstract agent notion (Chap. 3).

In Chap. 2 we presented and analyzed the state-of-the-art in HMS. We attempted to provide a global overview of the field, presenting the different studies developed in the areas of: architecture (Sect. 2.3.1), holons interconnection (Sect. 2.3.2), holons operation (Sect. 2.3.3), holonic control (Sect. 2.3.4), and methods for HMS development (Sect. 2.3.5). The more active fields are those related to developing holonic control systems. From these developments we can conclude that, currently, multi-agent system technology is the tool most used for developing HMS. Nevertheless, there is very little work on methods for HMS development.

In Chap. 3 we studied the differences among the holon and the agent paradigms. From this study we have concluded that the most important difference is the holon recursiveness feature. This characteristic makes the holon a very useful modeling entity because it simplifies the complex system modeling. This is so because it abstracts away from the less important requirements of inner holonic structures when analyzing a given cooperation or interaction problem. We have defined the abstract agent notion in order to have recursive agents that can be used in the analysis and design phases of HMS development. The abstract agent notion helps in the defi-

inition of a HMS methodology with a “uniformity of concepts” because the same modeling notion can be used from the HMS analysis to its implementation.

Part II was the central contribution of this book. It was organized into three chapters. Its goal was to present the HMS development problem and to detail the ANEMONA methodology.

Chapter 4 presented the HMS development requirements and analyzed state-of-the-art methods from the HMS research field, multi-agent systems methodologies and enterprise modeling techniques. All of the requirements presented in this chapter are specific to HMS. The methods from the HMS field are still in a developmental state and so are incomplete and immature. Therefore we must look for mature, robust methods and with some evidence of application. MAS methods may seem suitable for several reasons: several studies exist, many of which are for general purposes, and there is a reduced group specific for the manufacturing domain. Agent technology is the HMS implementation tool most used. Finally, the area of enterprise modeling offers interesting studies and proposals for the standardization of notation for the modeling of the business processes of a company. With these three groups: HMS methods, MAS methods and enterprise modeling, we carry out a comparison study based on HMS modeling requirements. From this study the need to develop a complete methodology for HMS that deals with all of these requirements was evident.

In Chap. 5 we presented the ANEMONA notation. Its central modeling entity is the abstract agent notion in Chap. 3. In ANEMONA the HMS is divided into different views of the system. These views build up the complete HMS analysis and design specification. They are: the agent model, which is concerned with the functionality of each abstract agent, responsibilities and capabilities; the organization model, which describes how system components (abstract agents, roles, resources, and applications) are grouped together; the interaction model, which addresses the exchange of information or requests between abstract agents; the environment model, which defines the non-autonomous entities with which the abstract agents interacts; the task and goal model that describes the relationships among goals and tasks, goal structures, and task structures.

In Chap. 6 the ANEMONA development process was explained. The development process details how the different ANEMONA models are built and the step-by-step activities and tasks to develop the HMS. This process is a mixed top-down and bottom-up approach. The aim of the analysis phase is to provide high-level HMS specifications from the problem *requirements*, which are specified by the *client/user* and that can be updated at any development stage. The analysis adopts a top-down recursive approach. One advantage of a recursive analysis is that its results, i.e., the *analysis models*, provide a set of elementary elements and assembling rules. The next step in the development process is the *holon design* stage, which is a bottom-up process to produce the *system architecture* from the *analysis models* of the previous stage. The aim of the *holons implementation* stage is to produce an *executable code* for the *setup and configuration* stage. Finally, maintenances functions are exe-

cuted at the *operation and maintenance* stage. Our approach provides HMS-specific guidelines to help the designer in every step of development.

A software engineer who follows the ANEMONA method is able to develop a holonic manufacturing system using multi-agent system technology from the beginning. This is so thanks to the abstract agent notion (Chap. 3). Moreover, the guidelines provided by ANEMONA help the software engineer to identify and specify all of the possible flexible cooperation scenarios and agent features in order to fulfill the manufacturing system requirements. The ANEMONA methodology, is appropriate for the domain of intelligent manufacturing systems and sufficiently prescriptive for a software engineer with minimal training in multi-agent technology.

In Part III of the book an evaluation discussion was presented in Chap. 7 and a complete case study was shown in Chap. 8.

In Chap. 7 we presented an evaluation discussion on ANEMONA's applicability to intelligent manufacturing problems. We also argued that ANEMONA is sufficiently prescriptive for a software engineer with minimal training in multi-agent technology. These claims are supported by two real case studies evaluated by third parties. The case studies were reviewed by: (i) software engineers from a manufacturing research and development institute with no prior experience in agent development; (ii) students with minimal agent technology background, and; (iii) manufacturing engineers with minimal training in multi-agent technology. Moreover, we argue that ANEMONA is more appropriate for intelligent manufacturing problems than other existing (enterprise-based, agent-oriented, and holonic-based) methods by highlighting the way in which the methodology overcomes the limitations of the methodologies discussed in Chap. 4.

In Chap. 8 we presented a detailed case study from a ceramics tile factory. With this case study we have shown the applicability of ANEMONA to a real industrial case study. We have presented the analysis and design of a HMS for the management of several business and production processes of a group of ceramics companies. The development of the HMS by abstraction levels, guided by the abstract agent notion and the ANEMONA guidelines, help the software engineer to focus on the important features and requirements of the problems. In this way the HMS development is a simple incremental process of identification and specification of holons.

9.2 Future Work

Despite the achievements of this book, there are still some issues left for future work. This section points out the most important ones.

We are working on a CASE tool for the ANEMONA methodology that will support all of the development phases using abstraction levels. The tool will manage the decomposition and composition processes in a semi-automatic way. We are extending the INGENIAS IDK [149], including similar techniques to the works of [150, 151, 152]. For the implementation phase we will use the same technique used

in INGENIAS, adding a new module for the function block code generation. We are also working on the definition of a module for design patterns and for reusing parameterized holarchies, in order to define a holarchy library from previous developments.

Another important issue to work on is the application of ANEMONA to more real industrial case studies. A continuous evaluation of the applicability of ANEMONA to new application domains could prove very important, not only to measure the ANEMONA's features, but also to add new modeling guidelines in order to help the software engineer in the development process.

The abstract agent notion is not only useful for HMS modeling but also for the modeling of large-scale complex problems, in which it is necessary to work in different abstraction levels. Studying the modifications or enhancements required in order to adapt ANEMONA to those domains is a very interesting problem. Analyzing the modeling of norms, rules, open systems, etc. and how these features are propagated to the different abstraction levels, are open research problems. The definition of ANEMONA methodology fragments in order to connect its phases or models to other method fragments is also a very interesting line of research [153].

References

- [1] Shen W, Norrie DH. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems, an International Journal*. 1999;1(2):129–156.
- [2] Jin-Hai L, Anderson A, Harrison R. The evolution of agile manufacturing. *Business Process Management Journal*. 2003;9:170–189.
- [3] McFarlane DC, S B. Holonic Manufacturing Control: Rationales, Developments and Open Issues. In: M DS, editor. *Agent-Based Manufacturing. Advances in the Holonic Approach*. Berlin: Springer-Verlag; 2003. p. 301–326.
- [4] HMS PR. HMS Requirements. <http://hms.ifw.uni-hannover.de/>: HMS Server; 1994.
- [5] Wooldridge M, Jennings NR. *Intelligent Agents – Theories, Architectures, and Languages*. vol. LNCS 890. Berlin: Springer-Verlag; 1995.
- [6] Koestler A. *The Ghost in the Machine*. London: Arkana Books; 1971.
- [7] Suda. Future Factory System Formulated in Japan. *Techno Japan*. 1989;22(10):15–25.
- [8] Dilts DM, Boyd NP, Whorms HH. The Evolution of Control Architectures for Automated Manufacturing Systems. *Journal of Manufacturing Systems*. 1991;10(1):79–93.
- [9] Christensen J. HMS/FB Architecture and Its Implementation. In: Deen SM, editor. *Agent-Based Manufacturing. Advances in the Holonic Approach*. Berlin: Springer-Verlag; 2003. p. 53–88.
- [10] Christensen J. Holonic Manufacturing Systems: Initial Architecture and Standards Directions. In: *Proceedings of First European Conference on Holonic Manufacturing Systems, European HMS Consortium*. Hanover; 1994.
- [11] Bussmann S. An Agent-Oriented Architecture for Holonic Manufacturing Control. *Proc of 1st Int Workshop on Intelligent Manufacturing Systems, EPFL*. 1998;p. 1–12.
- [12] Fletcher M, Garcia-Herreros E, Christensen JH, Deen SM, Mittmann R. An Open Architecture for Holonic Cooperation and Autonomy. In: *Proceedings of HoloMAS'2000*. Greenwich: IEEE Computer Society; 2000.
- [13] Fletcher M, Brennan RW. Designing a holonic control system with IEC 61499 function blocks. In: *Proceedings of the International Conference on Intelligent Modeling and Control*; 2001.
- [14] IEC. International Electrotechnical Commission: Function Blocks, Part 1 – Architecture. PAS 61499-1; 2000.
- [15] IEC. International Electrotechnical Commission: Function Blocks, Part 1 – Software Tool Requirements. PAS 61499-2; 2001.
- [16] Deen SM, Fletcher M. Temperature Equilibrium in Multi-Agent Manufacturing Systems. In: *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*. London: IEEE Computer Society; 2000. p. 259–270.

- [17] Brennan RW, Norrie DH. From FMS to HMS. In: Deen SM, editor. *Agent-Based Manufacturing: Advances in the Holonic Approach*. Berlin: Springer-Verlag; 2003. p. 31–49.
- [18] Brennan RW, Hall K, Mark V, Maturana FP, Norrie DH. A Real-Time Interface for Holonic Control Devices. In: Mark V, McFarlane D, Valckenaers P, editors. *Holonic and Multi-Agent Systems for Manufacturing*. vol. LNAI 2744. Berlin: Springer-Verlag; 2003. p. 25–34.
- [19] Fischer K. An Agent-Based Approach to Holonic Manufacturing Systems. In: L M Camarinha-Matos HA, v Marik, editors. *Intelligent Systems for Manufacturing, Multi-Agent Systems and Virtual Organisations*. The Netherlands: Kluwer Academic Publishers; 1998. p. 3–12.
- [20] Müller JP. The design of intelligent agents: a layered approach. In: LNAI 1177.
- [21] FIPA Modeling TC. *Agent Class Superstructure Metamodel – Working Document*. <http://www.auml.org/auml/documents/>. 2005.
- [22] Maturana FP, Norrie DH. Distributed decision-making using the contract net within a mediator architecture. *Decision Support Systems* 20. 1997;p. 53–64.
- [23] Van Brussel H, Wyns J, Valckenaers P, Bongaerts L, Peeters P. Reference Architecture for Holonic Manufacturing Systems: PROSA. *Computers In Industry*. 1998;37:255–274.
- [24] Hadeli S, Valckenaers P, Kollingbaum M, Van Brussel H. Multi-agent coordination and control using stigmergy. *Computers in Industry*. 2004 January;53(1):75–96.
- [25] Leitao P, Restivo F. Holonic Adaptive Production Control Systems. In: *Proceedings of special session on Agent-based Intelligent Automation and Holonic Control Systems of the 28th Annual Conference of the IEEE Industrial Society*. Sevilla; 2002. p. 2968–2973.
- [26] Leitao P, Restivo F. Identification of ADACOR Holons for Manufacturing Control. In: *Proceedings of the 7th IFAC Workshop on Intelligent Manufacturing Systems*. Budapest; 2003. p. 109–114.
- [27] Chirn J, McFarlane DC. A Holonic Component-Based Architecture for Manufacturing Control Systems. In: *Proceedings of MAS99*. New Mexico; 1999.
- [28] Gou L, Hasegawa T, Luh P, Tamura S, Oblak J. Holonic Planning and Scheduling for a Robotic Assembly Testbed. In: *Proceedings of the 4th Rensselaer International Conference on Computer Integrated Manufacturing and Automation Technology*. NY; 1994.
- [29] Hasegawa T, Gou L, Tamura S, Luh PB, Oblak JM. Holonic Planning and Scheduling Architecture for Manufacturing. In: *Proceedings of the 2nd International Working Conference on Cooperating Knowledge-based Systems*. Keele; 1994.
- [30] Deen SM. Cooperation issues in Holonic Manufacturing Systems. *Information Infrastructure Systems for Manufacturing*. 1993;B-14:401–412.
- [31] Deen SM. A cooperation framework for holonic interactions in manufacturing. In *Proceedings of the Second International Working Conference on Cooperating Knowledge Based Systems (CKBS'94)*. 1994.
- [32] Biswas G, Sugato B, Saad A. Holonic Planning and Scheduling for Assembly Tasks. In: TR CIS-95-01, Center for Intelligent Systems, Vanderbilt University. Tennessee; 1995.
- [33] Saad A, Biswas G, Kawamura K, Johnson ME, Salama A. Evaluation of Contract Net-Based Heterarchical Scheduling for Flexible Manufacturing Systems. In: *Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI'95)*. Montreal; 1995. p. 310–321.
- [34] Gou L, Luh P, Kyoya Y. Holonic Manufacturing Scheduling: architecture, cooperation, mechanism, and implementation. *Computers in Industry*. 1998;37:213–231.
- [35] Ramos C. A holonic approach for task scheduling in manufacturing systems. In: *Proceedings of IEEE Conference on Robotics and Automation*. Minneapolis; 1996. p. 2511–2516.
- [36] Sugimura N, Hiroi M, Moriwaki T, Hozumi K. A study on holonic scheduling for manufacturing systems of composite parts. In: *Proceedings of Japan/USA Symposium on Flexible Manufacturing*. Boston; 1996. p. 1407–1410.
- [37] Marcus A, Kis Vancza T, Monostori L. A market approach to holonic manufacturing. *Annals of CIRP*. 1996;45:433–436.
- [38] Heikkilä T, Jarviluoma M, Hasemann J. Holonic Control of a Manufacturing Robot Cell. *VTT Automation*; 1995.

- [39] Heikkila T, Jarviluoma M, Juntunen T. Holonic Control for Manufacturing Systems: Design of a Manufacturing Robot Cell. *Integrated Computer Aided Engineering*. 1997;4:202–218.
- [40] Ng A, Yeung R, Cheung E. HSCS – the design of a holonic shopfloor control system. In: *Proceedings of IEEE Conference on Emerging technologies and Factory Automation*. Hawaii; 1996. p. 179–185.
- [41] Agre J, Elsley G, McFarlane D, Cheng J, Gunn B. Holonic Control of Cooling Control System. In: *Proceedings of Rensselaer Manufacturing Conference*. NY, USA; 1994.
- [42] McFarlane DC. Holonic Manufacturing Systems in Continuous Processing: Concepts and Control Requirements. In: *Proceedings of ASI 95*. Portugal; 1995.
- [43] Brown J, McCarragher B. Maintenance resource allocation using decentralised cooperative control. ANU; 1998.
- [44] Bongaerts LB, Valckenaers VH, Peeters P. Reactive Scheduling in Holonic Manufacturing Systems: Architecture, Dynamic Model and Cooperation Strategy. In: *Proceedings of the Advanced Summer Institute of the Network of Excellence on Intelligent Control and Integrated Manufacturing Systems*; 1997.
- [45] Zhang H, Norrie DH. Autonomous Control for Open Manufacturing Systems. In: *Proceedings of IPMM99*. Hawaii; 1999.
- [46] Zweben M, Fox M. *Intelligent Scheduling*. San Francisco: Morgan Kaufman; 1994.
- [47] Prosser P, Buchanan I. *Intelligent Scheduling: past, present and future*. *Intelligent Systems Engineering*. 1994;3(2):67–78.
- [48] Duffie N, Prabhu V. Real-time distributed scheduling of heterarchical manufacturing systems. *Journal of Manufacturing Systems*. 1994;13(2):94–107.
- [49] Lin GYJ, Solberg JJ. Autonomous Control for Open Manufacturing Systems. In: Joshi S, Smith J, editors. *Computer Control of Flexible Manufacturing Systems*. NY: Chapman & Hall; 1994. p. 169–206.
- [50] Gayed N, Jarvis D, Jarvis J. A Strategy for the Migration of Existing Manufacturing Systems to Holonic Systems. In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. San Diego; 1998. p. 319–324.
- [51] Valckenaers P, Van Brusel H, Bongaerts L, Bonneville F. Programming, Scheduling and Control of Flexible Assembly Systems. *Computers in Industry*. 1995;26:209–218.
- [52] Bengoa A. An Approach to Holonic Components in Control of Machine Tools. *Annals of CIRP*. 1996;45(1).
- [53] Rannanjarvi L, Heikkila T. Software Development for Holonic Manufacturing Systems. *Computers in Industry*. 1998;37(3):233–253.
- [54] Tanaya P, Detand J, Kruth J. Holonic Machine Controller: A Study and Implementation of Holonic Behaviour to Current NC Controller. *Computers in Industry*. 1997;33:325–333.
- [55] Tanaya P, Detand J, Kruth J, Valckenaers P. Object-Oriented Execution Model For a Machine Controller Holon. *European Journal of Control*. 1998;4(4):345–361.
- [56] Zhang H, Norrie DH. Holonic Control at the Production and Controller Levels. In: *Proceedings of IMS99*. Leuven; 1999.
- [57] Leitao P, Restivo F. An Approach to the Formal Specification of Holonic Control Systems. In: Mark V, McFarlane D, Valckenaers P, editors. *Holonic and Multi-Agent Systems for Manufacturing*. vol. LNAI 2744. Berlin: Springer-Verlag; 2003. p. 59–70.
- [58] Fischer K, Schillo M, Siekmann J. Holonic Multiagent Systems: A Foundation fo Organisation of Multiagent Systems. In: Mark V, McFarlane D, Valckenaers P, editors. *Holonic and Multi-Agent Systems for Manufacturing*. vol. LNAI 2744. Springer-Verlag; 2003. p. 71–80.
- [59] Giret A, Botti V. Holons and Agents. *Journal of Intelligent Manufacturing*. 2004;15:645–659.
- [60] Giret A, Botti V. Towards an Abstract Recursive Agent. *Integrated Computer-Aided Engineering*. 2004;11(2):165–177.
- [61] Nwana HS. *Software Agents: An Overview*. *Intelligent Systems Research AA&T, BT Laboratories*. 1996.

- [62] Franklin S, Graesser A. It is an agent, or just a Program?: A Taxonomy for Autonomous Agents. In: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. München: Springer-Verlag; 1996.
- [63] Austin JL. How to Do Things With Words. Cambridge: Harvard University Press; 1975.
- [64] Finin T, Fritzson R, McKay D, McEntire R. KQML as an Agent Communication Language. In: Proceedings of the 3rd International Conference on Information and Knowledge Management CIKM'94. Gaithersburg, Maryland: ACM Press; 1994. p. 456–463.
- [65] FIPA. Arcol. Foundation for Intelligent Physical Agents. <http://www.fipa.org/>. 2002.
- [66] W3C. XML. World-Wide Web Consortium XML. <http://www.w3.org/>. World-Wide Web Consortium XML. <http://www.w3.org/>; 2002.
- [67] OHare GPM, Jennings NR. Foundation of Distributed Artificial Intelligence. USA: John Wiley & Sons; 1996.
- [68] Huhns MN, Singh MP. Readings in Agents. USA: Morgan Kaufman Publishing; 1998.
- [69] Brennan RW, Norrie DH. Agents, Holons and Functions Blocks: Distributed Intelligent Control in Manufacturing. Journal of Applied Systems Studies Special Issue on Industrial Applications of Multi-Agent and Holonic Systems. 2001;2(1):1–19.
- [70] Russell S, Norvig P. Artificial Intelligence: A Modern Approach. Englewood Cliffs, New Jersey: Prentice Hall; 1995.
- [71] Galliers JR. A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict. Open University. UK; 1988.
- [72] Sen S, Weiss G. Learning in multi-agent systems. In: Weiss G, editor. Multi-Agent Systems: A modern Approach to Distributed AI. Cambridge, MA: MIT Press; 1999. p. 259–298.
- [73] Shen W, Maturana F, Norrie DH. Learning in Agent-Based Manufacturing Systems. In: Proceedings of AI & Manufacturing Research Planning Workshop. Albuquerque, NM: The AAAI Press; 1998. p. 177–183.
- [74] Vigna G. Mobile Agents and Security. vol. LNCS 1419. Berlin: Springer-Verlag; 1998.
- [75] Fletcher M, Deen MS. Fault-tolerant holonic manufacturing systems. Concurrency and Computation: Practice and Experience. 2001;13(1):43–70.
- [76] Gasser L. Boundaries, Identity and Aggregation: Plurality Issues in Multi-Agent Systems. In: Werner E, Demazeau Y, editors. Decentralized A.I.-3. Kaiserslauten: Elsevier; 1992. p. 199–213.
- [77] Ocello M. Towards a Recursive Generic Agent Model. In Proceedings of International Joint Conference on Artificial Intelligence. 2000.
- [78] Gmytrasiewicz PJ, Durfee EH. A rigorous, operational formalization of recursive modeling. In: Proceedings of the First International Conference on Multi-Agent Systems. San Francisco; 1995. p. 125–132.
- [79] Tambe M. Recursive agent and agent-group tracking in a real-time dynamic environment. In: Lesser V, Gasser L, editors. Proceedings of the First International Conference on Multiagent Systems (ICMAS'95). San Francisco: AAAI Press; 1995. p. 368–375. Available from: citeseer.nj.nec.com/tambe95recursive.html.
- [80] Parunak VD, Odell J. Representing Social Structures in UML. In: M Wooldridge GW, Ciancarini P, editors. In Agent-Oriented Software Engineering II.
- [81] Bussmann S, Jennings N, Wooldridge M. Multiagent Systems for Manufacturing Control. A design Methodology. Berlin: Springer-Verlag; 2004.
- [82] Lin GYJ, Solberg JJ. Integrated Shop Floor Control Using Autonomous Agents. IIE Transactions: Design and Manufacturing. 1992;26(3):57–71.
- [83] Parunak VD. Manufacturing Experience with the Contract Net. In: Huhns MN, editor. Distributed Artificial Intelligence. London: Pitman; 1987. p. 285–310.
- [84] Cutkosky MR, Engelmores RS, Fikes RE, Genesereth MR, Gruber TR, Mark WS, et al. PACT: An Experiment in Integrating Concurrent Engineering Systems. IEEE Computer. 1993;26(1):28–37.
- [85] Peng Y, Finin T, Labrou Y, Chu B, Long J, Tolone WJ, et al. A Multi-Agent System for Enterprise Integration. In: Proceedings of PAAM'98. London; 1998.

- [86] Maturana F, Norrie D. Multi-Agent Mediator Architecture for Distributed manufacturing. *Journal of Intelligent Manufacturing*. 1996;7:257–270.
- [87] Cockburn D, Jennings N. ARCHON: A DAI system for industrial applications. In: OHare G, Jennings N, editors. *Foundations of Distributed Artificial Intelligence*. NY: Wiley; 1999. p. 319–344.
- [88] Monceyron E, Barthes J. Architecture for ICAD systems: an example form harbor design. *Revue Sciences et Techniques de la Conception*. 1992;1(1):49–68.
- [89] Fischer K. Agent-Based Design of Holonic Manufacturing Systems. *Journal of Robotics and Autonomous Systems*. 1999;27(1-2):3–13.
- [90] Kinny D, Georgeff M. Modelling and Design of Multi-Agent Systems. In: *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures and Language*. London: Springer-Verlag; 1997.
- [91] Georgeff M, Rao A. BDI agents: From theory to practice. In: *Proceedings of the First International Conference on Multi-Agents Systems (ICMAS-95)*. San Francisco; 1995.
- [92] Burmeister B. Models and methodologies for agentoriented analysis and design. DFKI; 1996.
- [93] Iglesias C, Garijo M, Gonzalez JC, Velasco JR. Analysis and design of multi agent systems using MAS-CommonKADS. In: Singh RA M P, Wooldridge MJ, editors. *Intelligent Agent IV*. LNAI 1365. München: Springer-Verlag; 1998.
- [94] Schreiber G. *Knowledge Engineering & Management: The CommonKADS Methodology*. Massachusetts: MIT Press; 1999.
- [95] Wooldridge M, Jennings NR, Kinny D. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*. 2000;3(3):285–312.
- [96] Zambonelli F, Jennings NR, Wooldridge M. Developing Multiagent Systems: the Gaia Methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. 2003;12(3):317–370.
- [97] Juan T, Pierce A, Sterling L. Roadmap: Extending the GAIA methodology for complex open systems. In: *The First International Joint Conference on Autonomous Agents and Multiagent Systems*. Bologna: ACM Press; 2002. p. 3–10.
- [98] Lind J. MASSIVE: Software Engineering for Multi-agent Systems. DFKI; 1999.
- [99] Mylopoulos J, Kolp M, Castro J. UML for agent-oriented software development: The TROPOS proposal. In: *Proceedings of the 4th Int. Conf. on th Unified Modeling Language UML'01*. Toronto; 2001.
- [100] Yu E. *Modelling Strategic Relationships for Process Reengineering*. Department of Computer Science. University of Toronto. Canada; 1996.
- [101] Wood MF. *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*. Air Force Institute of Technology. Ohio; 2000.
- [102] Robinson DJ. *A Component Based Approach to Agent Specification*. School of Engineering. Air Force Institute of Technology; 2000.
- [103] EURESCOM. MESSAGE: Methodology for engineering systems of software agents. Initial methodology. EURESCOM; 2000.
- [104] EURESCOM. MESSAGE: Methodology for engineering systems of software agents (Final). EURESCOM; 2001.
- [105] Jacobson I, Booch G, Rumbaugh J. *The Unified Software Development Process*. USA: Addison Wesley; 1999.
- [106] EURESCOM. Final guidelines for the identification of relevant problem areas where agent technology is appropriate. EURESCOM; 2001.
- [107] Caire G, Leal F, Chainho P, Evans R, Garijo F, Gomez-Sanz J, et al. In: *Agent Oriented Analysis using MESSAGE/UML*. vol. 2222 - LNCS. Berlin: Springer-Verlag; 2001. p. 119–135.
- [108] Julian VJ. RT-MESSAGE: Desarrollo de Sistemas Multiagente de Tiempo Real. Universidad Politcnica de Valencia. Departamento de Sistemas Informticos y Computacin; 2002.
- [109] Soler J, Julian V, Rebollo M, Carrascosa C, Botti V. Towards a real-time MAS architecture. In: *Proceedings of Challenges in Open Agent Systems. AAMAS'02*. Bologna; 2002.

- [110] Botti V, Carrascosa C, Julian V, Soler J. Modelling agents in hard real-time environments. In: MAAMAW'99. vol. 1647 - LNAI. Berlin: Springer-Verlag; 1999. p. 63–76.
- [111] Gomez JJ. Modelado de Sistemas Multi-agente. Universidad Complutense de Madrid. Facultad de Informtica; 2002.
- [112] Moulin B, Brassard M. A scenario-based design method and an environment for the development of multiagent systems. In: Zhang C, Lukose D, editors. Distributed Artificial Intelligence – Architecture and Modelling. vol. 1087 - LNAI. London: Springer-Verlag; 1996. p. 216–232.
- [113] Padgham L, Winikoff W. Prometheus: A methodology for developing intelligent agents. In: Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at AAMAS 2002. Bologna; 2002. p. 135–145.
- [114] Elammari M, Lalonde W. An agent-oriented methodology: high-level and intermediate models. In: Proceedings of First Bi-Conference. Workshop on Agent-Oriented Information Systems (AOIS'99). Heidelberg; 1999.
- [115] Buhr RJA. Use case maps as architectural entities for complex systems. IEEE Transactions on Software Engineering. 1998;24(12):1131–1155.
- [116] Miles S, Joy M, Luck M. Designing agent-oriented systems by analysing agent interactions. In: Ciancarini P, Wooldridge M, editors. Agent-Oriented Software Engineering. vol. 1657 - LNAI. NY: Springer-Verlag; 2001. p. 171–183.
- [117] Collinot A, Drogoul A, Benhamou P. Agent-oriented design of soccer robot team. In: Proceedings of the Second International Conference on Multi Agent Systems (ICMAS'96). USA: AAAI Press; 1996. p. 41–47.
- [118] Dignum V, Meyer J, Weigand H, Dignum F. An Organization-oriented Model for Agent Societies. In: Lindemann D, Moldt M, Paolucci B, Yu, editors. International Workshop on Regulated Agent-Based Social Systems: Theory and Applications (RASTA'02). Bologna; 2002. p. 31–50.
- [119] Dellarocas C, Klein M. Civil agent societies: Tools for inventing open agent-mediated electronic marketplaces. In: ACM Conference on Electronic Commerce (EC-99) (at IJCAI'99). Stockholm; 1999.
- [120] Kendall EA, Malkoun MT, Jiang CH. A methodology for developing agent based systems. In: Zhang C, Lukose D, editors. Distributed Artificial Intelligence – Architecture and Modelling. vol. 1087 - LNAI. Berlin: Springer-Verlag; 1996. p. 85–99.
- [121] Jacobson I. Object-Oriented Software Engineering: A Use Case Driven Approach. USA: Addison Wesley; 1992.
- [122] IDEF. IDEF Family of Methods. <http://www.idef.com/>; 2004.
- [123] Rao AS, Georgeff MP. An abstract architecture for rational agents. In: Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning. Massachusetts; 1992. p. 439–449.
- [124] Wooldridge M. Agent-Based Software Engineering. In: IEEE Proceedings of Software Engineering. vol. 14. London; 1997. p. 26–37.
- [125] Ritter A, Baum W, Hopf M, Westkamper E. Agentification for production systems. In: Second International Workshop on Integration of Specification Techniques for Applications in Engineering. Grenoble; 2002.
- [126] Colombo AW, Neubert R, Sussmann B. A coloured Petri net-based approach towards a formal specification of agent-controlled production systems. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. Tunisia; 2002.
- [127] Feldmann K, Colombo AW. Material flow and control sequence specification of flexible production systems using coloured Petri nets. International Journal of Advanced Manufacturing Technology. 1998;14:760–774.
- [128] Vernadat FB. Enterprise modeling and integration: principles and applications. London: Chapman & Hall; 1996.
- [129] ESPRIT. CIMOSA: Open System Architecture for CIM. vol. 1 of Research Reports ESPRIT, Project 688/5288 AMICE. AMICE EC, editor. Springer-Verlag; 1993.

- [130] UEML. Unified Enterprise Modelling Language. <http://www.ueml.org/>. Unified Enterprise Modelling Language. <http://www.ueml.org/>; 2003.
- [131] Doumeings G. GRAI grid decisional modelling. In: Bernus P, editor. Handbook on Architecture of Information System. Berlin: Springer-Verlag; 1998. p. 313–337.
- [132] Spur G, Mertins K, Jochem R. In: Integrated Enterprise Modelling. Berlin: Beuth Verlag GmbH; 1996.
- [133] ARIS. ARIS. <http://www.aris.com/>; 2004.
- [134] OMG OMG. Software Process Engineering Metamodel Specification Version 1.0. <http://www.omg.org/docs/formal/02-11-14.pdf>. 2002.
- [135] Pavon J, Gomez JJ. Agent Oriented Software Engineering with INGENIAS. In: V Marik MPJ Müller, editor. Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003): Multi-Agent Systems and Applications II, LNAI 2691. Berlin: Springer-Verlag; 2003. p. 394–403.
- [136] Julian V, Botti V. Developing real-time multiagent systems. *Integrated Computer-Aided Engineering*. 2004;11:135–149.
- [137] OMG OMG. UML. Unified Modeling Language. <http://www.uml.org/>. 2005.
- [138] Lyytinen KS, Rossi M. METAEDIT+ – A Fully Configurable Multi User and Multi Tool CASE and CAME Environment. In: *Advanced Information Systems Engineering*. LNCS 1080. Berlin: Springer-Verlag; 1996.
- [139] OMG OMG. MOF. Meta Object Facility. <http://www.omg.org/technology/documents/formal/mof.htm>. 2004.
- [140] Groover MP. *Automation, Production Systems, and Computer Integrated Manufacturing*. Prentice Hall: Englewood Cliffs, NY; 1987.
- [141] Hitomi K. Analysis and design of manufacturing systems. In: Kusiak A, editor. *Handbook of Design, Manufacturing and Automation*. John Wiley & Sons: New York; 1994. p. 405–433.
- [142] Demarco T, Plauger PJ. *Structured Analysis and System Specification*. New Jersey: Prentice-Hall; 1979.
- [143] Castillo I, Smith JS. Formal modelling methodologies for control of manufacturing cells: survey and comparison. *Journal of Manufacturing Systems*. 2002;21(1):40–57.
- [144] Murata T. Petri nets: properties, analysis, and applications. In: *Proceedings of the IEEE*. vol. 77; 1989. p. 541–580.
- [145] Bellifemine F, Poggi A, Rimassa G. Developing multi-agent systems with JADE. In: *Intelligent Agents VII*. Ed. Castelfranchi, C. and Lesperance, Y. LNCS 1571. London: Springer-Verlag; 2001. p. 89–103.
- [146] JADE. Java Agent Development Framework. <http://jadetilab.com/>. 2005.
- [147] McFarlane DC, Kollingbaum M, Matson J, Valckenaers P. Development of algorithms for agent-oriented control of manufacturing flow shops. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*. Tucson; 2001.
- [148] Garcia MA, Valero S, Argente F, Giret A, Julian V. A FAST method to achieve Flexible Production Programming Systems. *IEEE Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*. 2008;38(2):242–252.
- [149] Grasia. INGENIAS IDE. <http://ingenias.sourceforge.net/>; 2005.
- [150] MacMAS. MacMAS. <http://www.tdg-seville.info/joaquimp/MacMAS/index.htm>; 2005.
- [151] Peña J, Corchuelo R, Arjona JL. Towards Interaction Protocol Operations for Large Multi-agent Systems. In: *Proceedings of the Second International Workshop on Formal Approaches to Agent-Based Systems*. vol. LNCS 2699. Berlin: Springer-Verlag; 2002. p. 79–91.
- [152] Peña J, Corchuelo R, Arjona JL. A Top Down Approach for MAS Protocol Descriptions. In: *Proceedings of the ACM Symposium on Applied Computing SAC'03*. Florida: ACM; 2003. p. 45–50.
- [153] FIPA Methodology TC. FIPA Methodology Technical Committee. <http://www.fipa.org/activities/methodology.html>. 2005.

Index

- abstract agent, 32, 63
- abstract belief, 65
- abstract goal, 64
- abstract task, 65
- abstraction level, 33
- AContainE relation, 68
- activity, 92
- adaptability, 22
- agent, 7, 21, 64
- agent model, 66
- agent-based manufacturing system, 7
- aggregation relation, 61
- AGOClientServer relation, 86
- AGOConditionalSubordination relation, 86
- AGONotConditionalSubordination relation, 86
- AGOSubordination relation, 86
- application, 65
- AResponsible relation, 68, 85
- association relation, 61
- autonomy, 10, 21, 23

- belief, 64
- benevolence, 22, 29
- bifurcate relation, 82
- business process, 94

- concurrent relation, 82
- cooperation, 10, 26
- cooperation domain, 12

- development process, 91
- device controlling, 19
- distributed intelligent manufacturing, 7
- divide and conquer, 2
- document, 92

- enterprise modeling, 53

- environment model, 83
- environment resource, 83
- EPerceive relation, 84
- EResourceBelongsTo relation, 83
- event, 66

- function block guidelines, 128
- functional decomposition, 43, 107

- goal, 64, 95
- goal-type property, 71
- group belief, 65
- group goal, 64
- GTAffect relation, 72
- GTCreat relation, 73
- GTDecompose relation, 76
- GTDecomposeAnd relation, 76
- GTDecomposeOr relation, 76
- GTDepend relation, 76
- GTDependAnd relation, 76
- GTDependOr relation, 76
- GTDestroy relation, 73
- GTFail relation, 73
- GTModify relation, 73
- GTPursue relation, 79, 85
- GTPursues relation, 67
- GTSatisfy relation, 73
- guidance, 92

- heterarchical control system, 7
- HMS-UC guidelines, 102
- holarchy, 8, 10
- holon, 8, 10
- holonic manufacturing system, 9, 10

- ICooperate relation, 79
- information processing part, 11, 30

- inheritance relation, 61
- Intelligent Manufacturing Systems, 8
- interaction, 65
- interaction model, 78
- interaction scenario, 78
- interaction specification, 82
- interaction unit, 65, 81
- IPursue relation, 79
- IStart relation, 79

- JADE agent template, 127
- JADE guidelines, 125, 126
- job-shop control, 19

- learning, 28

- max-deadline property, 70, 71
- mental attitude, 28
- mental state, 64
- metamodel, 60
- mobility, 22, 29
- model, 92
- multi-agent system, 7, 22, 32

- OContainA-Agent relation, 85
- openness, 26
- operation condition, 95
- organization, 64
- organization model, 84
- organization structure, 85
- organizational chart, 94

- periodic property, 70
- phase, 92
- physical decomposition, 42, 107
- physical processing part, 10, 30
- play relation, 67
- precede relation, 82
- pro-active, 21
- pro-active behavior, 78
- pro-activeness, 22
- pro-activity, 24
- process, 92
- process package, 92
- process role, 92
- product holon, 17
- PROSA, 17

- PROSA guidelines, 110–112

- rationality, 21, 27
- reactive, 21
- reactive behavior, 78
- reactivity, 21, 23
- real-time goal, 71
- real-time task, 70
- recursion level, 33
- recursiveness, 29, 30
- resource, 66
- resource holon, 17
- role, 63

- scheduling, 19
- sociability, 25
- social, 21
- social ability, 21
- SPEM, 91
- staff holon, 17
- system scope, 94

- task, 65
- task specification, 73
- task-type property, 70
- task/goal model, 70

- UICooperate relation, 81
- UIIteration relation, 82
- UIStart relation, 81

- veracity, 22

- WFConnect relation, 88
- WFConsume relation, 73, 88
- WFDecompose relation, 75, 76, 85
- WFProduce relation, 73, 88
- WFPursue relation, 79
- WFResponsible relation, 69
- WFUse relation, 73, 88
- work definition, 92
- work flow, 65, 88
- work product, 92
- work-order execution, 19
- work-order holon, 17
- work-order programming, 19