# 1st International Workshop on Organizational Modeling

# »ORGMOD 2009«

**A satellite event of the**

30th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency

**and**

the 20th IEEE/IFIP International Symposium on Rapid System Prototyping

**Paris, France, June, 2009**

# Preface by the Organisers

This volume consists of the contributions to the *first international workshop on organizational modeling* (OrgMod'09). The workshop took place as a satelite event of the 30th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN'09) and the 20th IEEE/IFIP International Symposium on Rapid System Prototyping (RSP'09) in Paris, France.

Enterprises and any kind of public bodies are supported by information systems. An important question is, how to align the enterprise with the supporting software and IT systems. Modeling the essential *organizational* structures, behaviors, concepts and entities is a prerequisite for a thorough understanding of this alignment. Numerous techniques such as Petri nets, Event-Process Chains (EPC), Unified Modeling Language (UML), BPEL, WSDL provide some means to build models in general. The supporting tools, methods and conceptual frameworks are also important factors for the provision of good models.

The workshops aimed to allow practitioners and researchers discuss the use of Petri nets, other formal and semi-formal techniques and their supporting frameworks and approaches for organizational modeling.

Topics of interest included modeling techniques (formal or semi-formal); different paradigms for modeling (object-oriented, process-oriented, agent-oriented, organization-oriented); technical frameworks and tools to support organizational modeling, verification, validation, simulation, and code generation for organizational modeling; middle-ware for organization-centred software; technologies for the Corporate Web; organizational case studies; evaluation methods and metrics for the Business/IT-alignment; organizational theories; E-Government; and organizational modeling problems from the perspective of other disciplines than computer science.

We like to thank all authors who have submitted papers. Each paper was reviewed by at least fourreferees. During the reviewing process the program comittee selected seven out of twelf contributions for publication. We wish to thank all members of the program comittee for their effort.

Finally we like to thank our invited speaker, Jacques Ferber, for his lecture: *From AGR to MASQ: Understanding organizations from a multi-agent point of view.*

June 2009

Michael Köhler-Bußmeier
Daniel Moldt
Olivier Boissier

# Conference Organization

## Programme Chairs

Michael Köhler-Bußmeier
Olivier Boissier
Daniel Moldt

## Programme Committee

Didier Buchs
Christine Choppy
Virginia Dignum
Marlon Dumas
Marc Esteva
Berndt Farwer
Jacques Ferber
Giuliana Franceschinis
Paolo Giorgini
Emmanuelle Grislin-le Strugeon
Vincent Hilaire
Thomas Hildebrandt
Jomi Fred Hübner
Catholijn Jonker
Eric Matson
Markus Nüttgens
James Odell
Andrea Omicini
Pascal Poizat
Oana Prisecaru
Costa Antonio Carlos da Rocha
Heiko Rölke
Amal El Fallah Seghrouchni
Alexei Sharpanskykh
Christophe Sibertin-Blanc
Natalia Sidorova
Jaime Simão Sichmann
Harald Störrle
Catherine Tessier
Wamberto Vasconcelos
Danny Weyns
Karsten Wolf
Secq Yann
Christian Zirpins

# Table of Contents

# An Approach for Building Holonic Organizational Models of Design Processes for Knowledge Management

Achraf Ben Miled[1], Davy Monticolo[1], Vincent Hilaire[1], and Abderrafiaa Koukam[1]

[1]Systems and Transports laboratory
Université de Technologie de Belfort Montbéliard.
90010 Belfort Cedex, France
vincent.hilaire@utbm.fr
(33) 384 583 009

**Abstract.** To survive in an increasingly competitive business environment, manufacturing enterprises are under unprecedented pressure to become leaner and more agile. Product leadership companies must continue to enter new market with innovative products. This requires optimising process and methodologies used by engineering departments. The design process has to be rationalized in capitalizing knowledge, know-how and technological patrimony. A solution in order to capitalize knowledge is to model the design process used and use this model as a context of reference. A first experiment was done to model a real enterprise design process. The model issued from this work was the result of the immersion in the enterprise and of the observation of professional workers. This empiric approach is error prone, time consuming and not easily repeatable. In this paper an approach based on goal oriented analysis is proposed in order to build such organizational models of design processes. These models ease the capitalization and reuse of knowledge. Indeed, the knowledge is capitalized according to a context defined by the design process model. Specifically the use of holons allows to define several levels of abstractions and to take into account knowledge at different levels of granularity.

## 1 Introduction

Nowadays, in a financial crisis and high business competition context, enterprises have to innovate in order to improve their business value. The product range must be updated permanently and production costs the lowest possible. Product leadership companies must continue to enter new market with innovative products. This requires optimising process and methodologies used by engineering department. The design process has to be rationalised in capitalising knowledge, know-how and technological patrimony. A solution to this problem consists in using Knowledge Management techniques. Several works have introduced the corporate memory as a support for Knowledge Management. A corporate memory is an explicit representation of pertinent knowledge of an organisation [16]. This memory, explaining the organizational knowledge (called equally collective Knowledge), may be considered as a knowledge base of the organisation. Such knowledge base can be specific to a project and so be called project memory. Project memories are memories of knowledge and information acquired and produced during the realisation of projects [3, 16]. Moreover a complementary approach in knowledge engineering claims that knowledge is a personal interpretation of information. This theory is defended in works which consist in searching for pertinent information instead of explanation. One can distinguish for example the works of M. Grundstein with Gameth [13]. For these authors, the knowledge is strongly dependant to a personal interpretation linked to a specific context. Knowledge Management is then meaningless without dealing with the context of this knowledge. We have proposed in an earlier work an organizational model based upon the concepts of role, interactions and organisations for the modelling of products design processes [22]. This model allow the capitalisation of knowledge during design projects according to interactions between project actors. The context for the capitalised knowledge is then defined by the different organisations. However this approach suffer from two drawbacks. The first is that it is a tedious work to produce organisations that model design processes from observing human actors at work. The second drawback is that the organizational models produced are flat models in the sense that they do not enable several levels of granularity in the design process.

In this paper we propose a methodological approach to build holonic models of collaborative design activities. Holons introduced in [15] are defined as self-similar structure composed of holons as substructures. The organizational structure defined by holons, called holarchy, allows the modelling at several granularity levels. These granularity levels in our case would be helpful to distinguish different levels of knowledge and take into account multi-enterprise or multi-group collaborations. In order to build such models for collaborative design activities we propose to reuse the ideas underlying the ASPECS methodology [9]. We integrate within this methodology concepts taken from a goal oriented requirements approach [28]. As stated in [25] Goals are useful for knowledge management systems as they enable to focus on strategic knowledge. They represent actor's strategic interests. Actors model dual entities that have strategic goals and intentionality in the system. In our case, goals will be used for model interest of all actors involved in a collaborative design project. Actors will represent all entities, single human or group, that have a stake in the project.

This paper is organised as follows: section 2 introduces the concepts underlying our approach: the *-design approach which was our first experiment for knowledge capitalization, the holonic concepts and the goal oriented analysis approach. Section 3 presents our approach and illustrate it on an example. Section 4 presents some related works. Eventually section 5 concludes.

## 2 Concepts

### 2.1 *-Design approach

Our knowledge management approach, called *-Design, aims to capitalize and reuse knowledge in taking into account the social and cooperative aspect of the engineering projects. This approach leads the definition of three models which will guide a Multi-Agent System (MAS from now on), called KATRAS which stands for Knowledge Acquisition Traceability by Agent System, to carry out the knowledge management process. In order to identify and reuse the knowledge which the professional actors need, the MAS has to possess a description of knowledge used for each professional activities. We consider these activities like organizations where professional actors have one or several roles and for each role they create, share and use knowledge. In studying the professional actors' roles and their interactions, describing the knowledge exchange in the professional activities, we obtain a model leading the knowledge capitalization and reuse all along the design projects. We call this organizational model "OrgaDesign". It describes the professional actors' roles, their interactions, competences and knowledge, updates throughout the projects. This analysis makes it possible to obtain a cartography of the knowledge created, used and shared in the objective to be capitalized and reused. In a second way, the agents have to know how to organize the knowledge identified in the organizational model. Thus we propose a knowledge typology based on the characteristics of the cooperative activities (roles of the actors in the activity, objectives of the activity, etc). This work allows defining a project memory model we have called "MemoDesign". This model leads to organize information and knowledge in order to present them to the professional actors. MemoDesign is used by the agents to store the knowledge created by a project team. The project memory model with the knowledge taxonomy not allow to the agents to handle information and knowledge. In order to achieve this task, we have completed the approach by a definition of the concepts used in MemoDesign, their relations and attributes constituting a mechanical design project ontology called OntoDesign. Thus the semantic and the vocabulary described in the ontology allow to the agents to organize the information capitalized according to the project memory model and to handle the knowledge elements in order to reuse them.

Consequently the MAS dedicated to the knowledge management at the time of project memory has to consider the three following models (figure 1) ; the organizational model of the design process OrgaDesign, the project memory model MemoDesign and the domain ontology OntoDesign.

As the organizational models produced by this approach are "flat" we use of the holonic paradigm in order to consider several granularity levels. The concepts underlying the holonic paradigm are presented in the next section.

### 2.2 Holons

The concept of holon is central to our discussion and therefore a definition of what is a holon should be helpful before proceeding. In multi-agent systems,

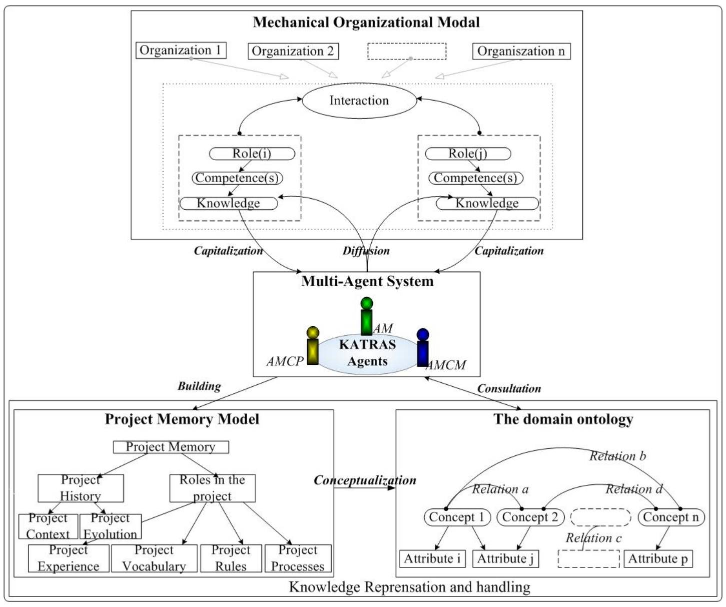


**Fig. 1.** Overview of the *-design approach

the vision of holons is much closer to the one that MAS researchers have of *Recursive* or *Composed* agents. A holon constitutes a way to gather local and global, individual and collective points of view. An holon is thus a self-similar structure composed of holons as sub-structures and the hierarchical structure composed of holons is called an *holarchy*. An holon can be seen, depending on the level of observation, either as an autonomous “atomic” entity or as an organisation of holons (this is often called the *Janus effect*).

Two overlapping aspects have to be distinguished in holons: the first is directly related to the holonic nature of the entity (a holon, called super-holon, is composed of other holons, called sub-holons or members) and deals with the government and the administration of a super-holon. This aspect is common to every holon and thus called the *holonic* aspect. The second aspect is related to the problem to solve and the work to be done. and depends on the application or application domain. It is therefore called the *production* aspect.

Holonic Systems have been applied to a wide range of applications, Manufacturing systems [6, 17], Health organisations [27], Transportation [7], Adaptive Mesh Problem [24] to mention a few. Thus it is not surprising that a number of models and frameworks have been proposed for these systems, for example PROSA [6], MetaMorph [17]. However, most of them are strongly attached to their domain of application and use specific agent architectures. In order to allow a modular and reusable modelling phase that minimises the impact on the underlying architecture, a meta-model based on an organizational approach is proposed. The adopted definition of role comes from [12]: "*Roles identify the activities and services necessary to achieve social objectives and enable to abstract from the specific individuals that will eventually perform them. From a society design perspective, roles provide the building blocks for agent systems that can perform the role, and from the agent design perspective, roles specify the expectations of the society with respect to the agent's activity in the society*". The figure

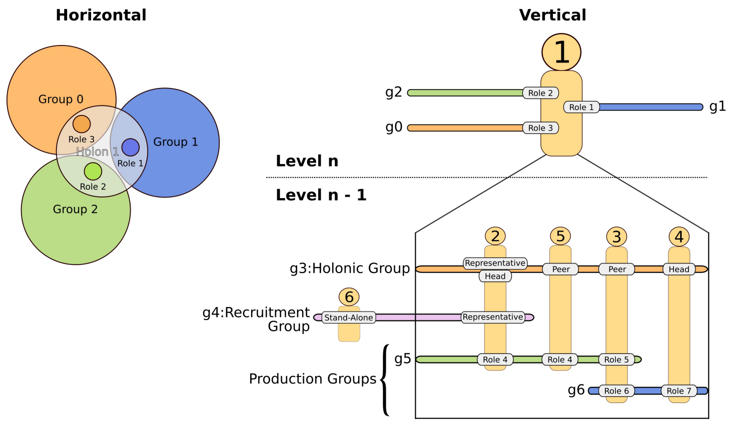


**Fig. 2.** The two different holon perspectives.

2 illustrates the two perspectives of a holon. On the left part, the horizontal perspecive, a single holon is presented as an entity which plays roles within different groups. Namely, this holon plays *Role*1 in *Group*1, *Role*2 in *Group*2 and *Role*3 in *Group*3. On the right part of the figure, the vertical perspective, an holarchy is represented with two levels. The holon named 1 at level $n$ is composed of several holons named from 2 to 6 at level $n - 1$. Each of these holons play different roles in different groups.

## 2.3 Goal oriented analysis

The $i^*$ framework [28] has been chosen to support goal oriented analysis in our approach. This framework provides an intuitive diagrammatic representation of goal oriented analysis through the use of goals, dependencies, roles, actors and positions. An actor abstracts an active entity. A goal is a condition or a state of affairs in the world that the actors would like to achieve and that has a criteria to determine if it is achieved or not. A softgoal is a state of affair such as a goal but with no clear achievement criteria. A resource is an entity physical or informational.
The $i^*$ framework proposes two types of diagram: the *Strategic dependency diagram* and the *Strategic rationale diagram*. The former focus on the identification

of dependency between actors. Indeed, one actor may depend on another for the realization of a goal or the availability of a resource. The actor which depends on another is called the depender while the other is called the dependee. The object (goal, resource) around which the dependency is defined is called the dependum. The figure 3 presents an example of abstract dependency diagram. In this figure there are two actors, *Actor*1 and *Actor*2, represented by circles. The first actor, *Actor*1, depends on the second, *Actor*2, for the realization of the goal named *Goal*1. The second type of diagram, the *Strategic rationale diagram*, is

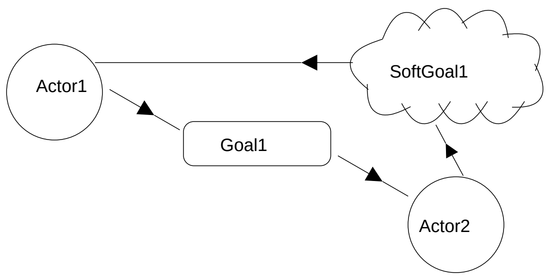


**Fig. 3.** Strategic dependency diagram

a refinement of the first and consists in an analysis of goals according to the point of view of actors. It means that goals are refined by using goal analysis techniques such as means-end analysis or AND/OR decompositions to define decompose a goal. An example of such a diagram is presented in figure 4. This figure details the analysis of the goal *Goal*1 assigned to *Actor*1. The analysis presented is realized with an AND/OR decomposition which results in a tree of goals. AND decomposition is represnted by linked arrows. It is the case for the decompostion of *Goal*1 in *Goal*2, *Goal*3 and *Goal*4. OR decompositions are represented by simple arrows as it is the case for decomposition of *Goal*3 in *Goal*5 and *Goal*6.

## 3 Methodological approach for building holonic models

### 3.1 Principles

The approach presented in this paper relies on the holonic modelling of design process. The resulting holarchy is used to support knowledge management. Indeed, this holarchy enables the definition of a context of activities and knowledge exchange between professional actors. This section presents a structured approach which allows the construction of the holarchy.

The holarchy represents a design process. As mechanical design process are often represented by IDEF0 diagrams [1] we base our methodological approach

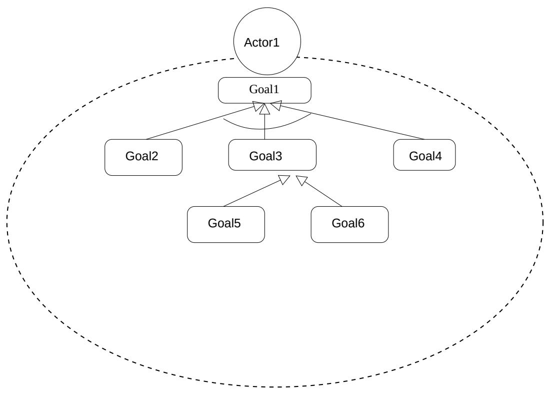


**Fig. 4.** Strategic rationale diagram

on them. An IDEF0 diagram presents activities with inputs, outputs, controls and supporting mechanism (in our case professional actors) involved.

**The first step** consists in defining a strategic dependency diagram to model the system-under-consideration. The goal of this model is to identify the involved actors and the top-level goals and softgoals and their dependencies. These top-level goals and the corresponding actor in charge of their realization is the holon root of the holarchy.

**The second step** consists in defining a strategic rationale diagram that refines the goals and actors by using one of the goal analysis technique and the IDEF0 diagrams that specify the design process. Each activity output of the IDEF0 diagrams is a contribution to the upper-level goal. The activity is the mean by which this goal is achieved. The upper-level goal is then decomposed using a specific goal oriented analysis technique. Each activity output of the IDEF0 diagram is a sub-goal.

**The third step** aims to define the organizational part of the holarchy. The root of the holarchy is the top-level actor in charge of the top-level goals. It represents the Project team. This top-level holon is composed of several organizations. Each organization is in charge of goals achievement. The principle for defining the organizations composing this level is to assign the goals identified in the strategic rationale diagram to activities able to fullfil them. These activities are extracted from the IDEF0 diagram. An organization thus represents an activity through the roles that will carry it. For the Project team holon, only the goals of the immediate level below the top-level goals are taken into consideration. This step is iterated while there are goal levels which are not assigned to organizations.

**The fourth step** insert specific roles in the holarchy, their respective constraints and the interactions which fullfil the organization goal. Indeed, each organization is composed of a set of interacting roles. Each role represents an abstract behavior, such as project leader or market analyst, with specific responsibilities, skills and knowledge. Each role may be part of a single or several activities and thus be part of a single or of several organizations at different levels of the holarchy. In order to define these roles scenarios may be helpful. Indeed, recurrent practices of the enterprise, formalized with sequence diagrams for example or textual description, should provide the different roles and their required knowledge and competences.

## 3.2 Example

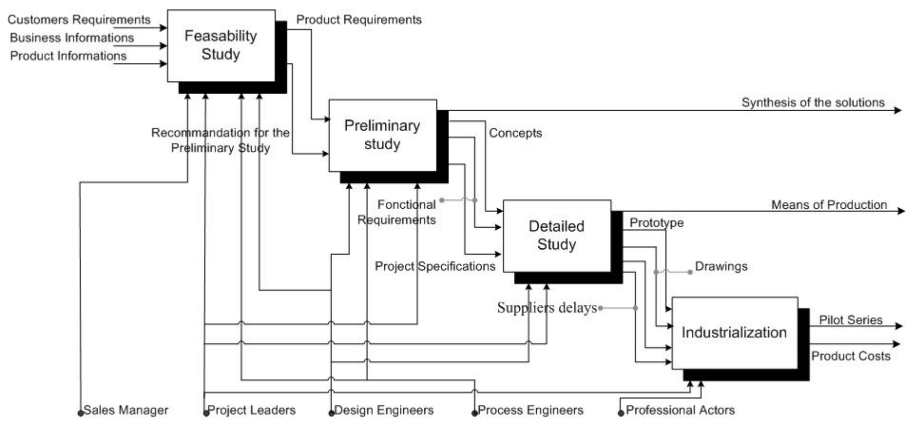


**Fig. 5.** IDEF0 diagram

An example of strategic dependency diagram is given in figure 6. This simple example introduces two actors, namely the client who wants a product to be realised and a provider who will realise the product. The goal "product realisation" introduces a dependance of the client to the provider. We have also defined two soft-goals "Increase benefits" and "Happy customer" which make the provider dependant on the client.

This Strategic dependency analysis enables to describe with a high level point of view the requirements of the model-to-be. Indeed, we do not want to design a system but an holonic model of collaborating humans engaged in a joint project. This diagram is the first step of our approach.

This model is enriched by goal modelling. The goal modelling consists in the analysis of an actor by either means-end analysis, AND/OR decomposition or contribution analysis. This modelling, represented as a strategic rationale diagram, is illustrated in figure 7. It is the second step of our approach. In this figure the provider actor is refined and some internal goals are elicited. The analysis relies on the IDEF0 diagram presented in figure 5. The product realisation goal of actor provider is decomposed in four sub-goals by an AND decomposition.

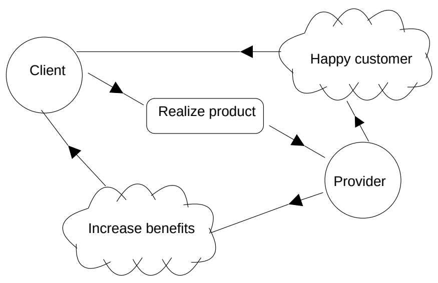


**Fig. 6.** Strategic dependency diagram for a collaborative design project

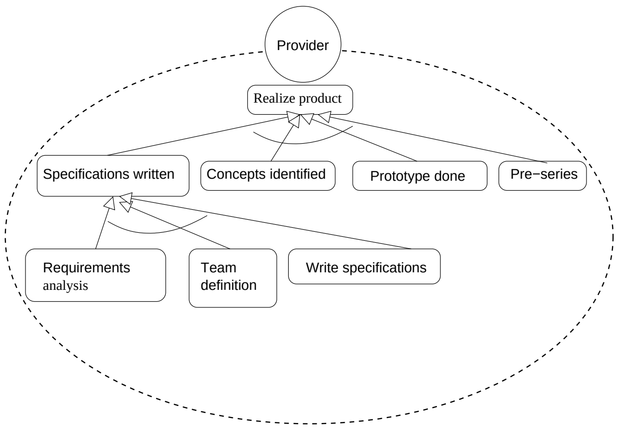


**Fig. 7.** Strategic rationale diagram

It means that in order to fulfil the Product realisation goal the four sub-goals need to be achieved. These four sub-goals are: Specifications written, Concepts identified, Prototype done and Pre-series. Each of these goals may in turn be refined and decomposed in sub-goals as illustrated for the Specifications written goal which is decomposed, again with an AND decomposition, in three sub-goals. These sub-goals are: Requirements analysis, Team definition and Write specifications. The goal modelling may continue until a fine level of detail is reached. In our example, for the sake of clarity we have omitted a large part of the goal analysis.

The next step of our approach consists in organizations identification according to the goal analysis and the IDEF0 diagram that describes the activities that can achieve goals. In our example, the Project team holon is at the root of the holarchy. The second level is composed of four organizations which correspond to the four activities of the IDEF0 diagram of figure 5. The feasibility study activity is in turn decomposed as this activity is decomposed in sub-activities.

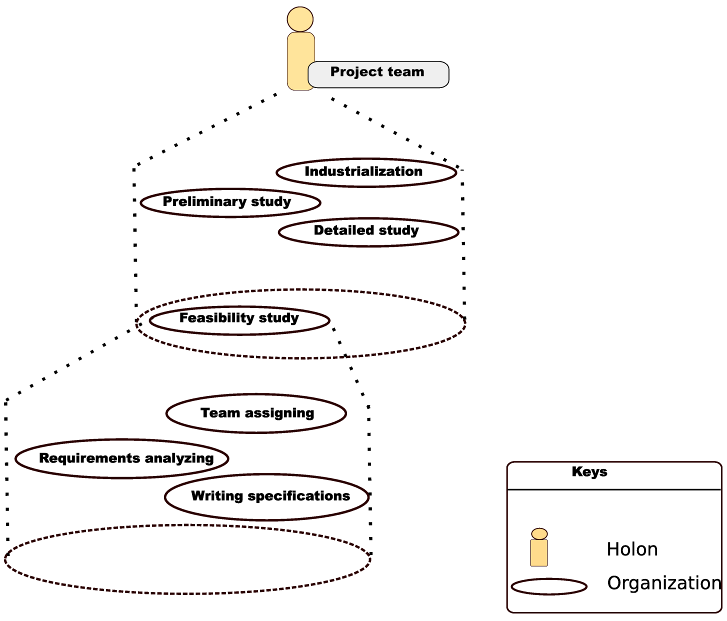


**Fig. 8.** Organizations identification

The fourth and last step of the methodology is dedicated to the refinement of organizations with roles, interactions and constraints. This step is illustrated by the organization detailed in figure 9. This figure details the Writting specifications organization. It is composed of three roles: the technical commercial assistant, the responsible for study and the project leader roles. The result of

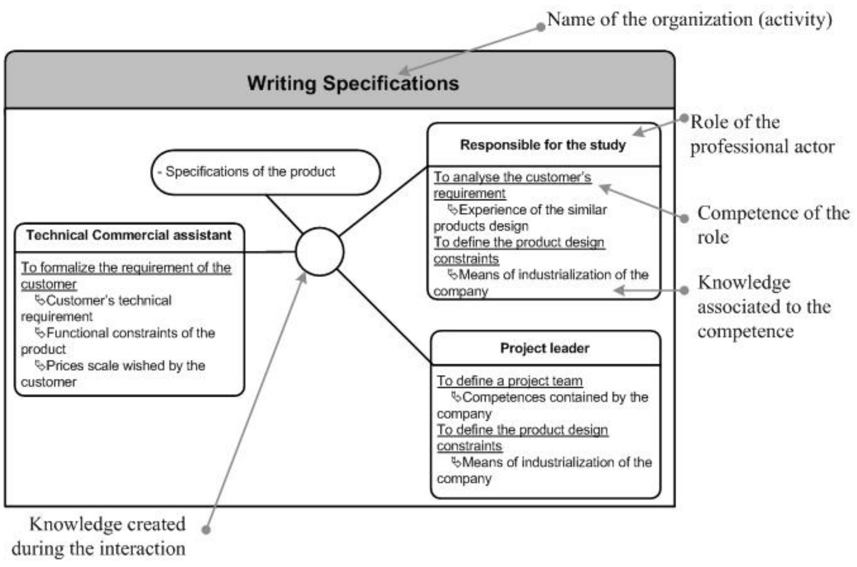

Name of the organization (activity)
Writing Specifications
Role of the professional actor
Responsible for the study
To analyse the customer's requirement
Experience of the similar products design
To define the product design constraints
Means of industrialization of the company
Competence of the role
Knowledge associated to the competence
- Specifications of the product
Technical Commercial assistant
To formalize the requirement of the customer
Customer's technical requirement
Functional constraints of the product
Prices scale wished by the customer
Project leader
To define a project team
Competences contained by the company
To define the product design constraints
Means of industrialization of the company
Knowledge created during the interaction

**Fig. 9.** Writting specifications organization

the roles interactions is the specifications document. The project leader role of this organization must be played by the same holon who plays the project leader role of the Project team holon. This constraint aims to ensure the coherence of the project since it beginnings.

## 4 Related works

Several works were realized in this domain: Belkadhi [4] proposes a meta-model to model the design process which is a graphic representation of the key concepts (situation, entity and specific roles) and proposes a method of knowledge capitalization by basing itself on this model. The proposed method integrates the various present aspects into a design project context. It consists in representing a design situation in the form of a set of interactions of various natures by clarifying the participation of every element in each of the interactions and consequently in the global progress of the project. In this sense, the passage of a given situation (at the moment t) to another situation (moment t +1) is the result of this set of interactions. . This result can be observed by the creation of new elements, the deletion or the modification of some or still on the evolution of the existing links between the various elements. It is what Bekhti and Matta [2] defines as the mutual influences of the elements of the project. Some adopted the organizational approach during the modelling of design process and knowledge management. By presenting the concept of the learner company Nonaka [23], takes into account the knowledge in the company, and brings to light the mechanisms of transmission and evolution of the knowledge in organizations. Organizational approach takes into account the social character in the knowledge management which has as aim, rather than to try to formalize, to model knowledge and know how, allowing the various holders and the experts to share them and to pass on them [10]. The organizational approaches focus essentially on design process creatures of habit. It leads to rethink the organization of the company. Several works were interested in the capitalization and the re-use of the knowledge in design to facilitate the decision-making during the projects. The main part of the identified contributions integrates a representation of the decision-makings and leans on the models of design logic [11]. We can quote the most known models such as QOC [26], DRCS [14], IBIS [8] and DRAMA [5].

## 5 Conclusion

The approach presented in this paper aims at building holonic organizational models of design processes in order to assist knowledge management. This approach is based upon the description of design processes activities with IDEF0 diagrams, which are a commonly used representation in enterprises, and a goal oriented analysis based upon the $i^*$ framework. A four steps methodology enables to identify the organizations and roles composing a holarchy representing the design process.

The holarchy resulting from the application of the presented methodology constitutes a context of reference for knowledge management approaches. Indeed, in the *-design approach, an organizational model, namely OrgaDesign, was used to identify the knowledge to capitalize [20], build the structure of a project memory [19] and define a conceptualization origin of an ontology of enterprise knowledge [21]. In this paper, the contribution versus the existing *-design approach is twofold. On the one hand a methodology is proposed to build the organizational model. In the previous *-design works the organizational models were the result of the monitoring and interviews of the professional actors. It was a tedious and error prone work. On the other hand, the original "flat" organizational models are now replaced by an holonic organizational model which allows the representation of different granularity levels.

Future works will consist in establishing an approach to validate the models produced by the methodology. This validation process will be based on simulations of the roles behaviors. We will also adapt and enrich the knowledge reuse, transfer and sharing approach, already defined for the original "flat" organizations [18] to the holonic organizational model.

## References

1. Dr Ang and Cheng Leong. Idef* : a comprehensive modelling methodology for the development of manufacturing enterprise systems. *International Journal of Production Research*, 37:3839–3858, 1999.

2. S. Bekhti and N. Matta. A formal approach to model and reuse the project memory. In K. Tochtermann and H. Maurer, editors, *Proceedings of I-Know '01, International Conference on Knowledge Management*. Springer, 2003.
3. S. Bekhti and N. Matta. Project memory: An approach of modelling and reusing the context and the design rationale. In *Proceedings of IJCAI'03 (International joint of conferences of Artificial Intelligence) Workshop on knowledge management and organisational memory*, 2003.
4. Farouk Belkadi, Eric Bonjour, and Maryvonne Dulmet. Modelling framework of a traceability system to improve knowledge sharing and collaborative design. In Weiming Shen, Kuo-Ming Chao, Zongkai Lin, Jean-Paul A. Barthès, and Anne E. James, editors, *CSCWD (Selected papers)*, volume 3865 of *Lecture Notes in Computer Science*, pages 355–364. Springer, 2005.
5. A. Brice. Design rationale management (drama). http://www.quantisci.co.uk/drama.
6. H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37:255–274, 1998.
7. Hans-Jürgen Bürckert, Klaus Fischer, and Gero Vierke. Holonic transport scheduling with teletruck. *Applied Artificial Intelligence*, 14(7):697–725, 2000.
8. J. Conklin and L. Begemann. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6(4), October 1988.
9. Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. A holonic metamodel for agent-oriented analysis and design. In *HoloMAS'07*, 2007.
10. W.L. Currie. A knowledge-based risk assessment framework for evaluation of web-enabled application outsourcing project. In *International conference of project management*, 2003.
11. Rose Dieng, Olivier Corby, Alain Giboin, and Myriam Ribière. Methods and tools for corporate knowledge management. *Int. J. Hum.-Comput. Stud*, 51(3):567–598, 1999.
12. Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. OMNI: Introducing social structure, norms and ontologies into agent organizations. In *PROMAS*, volume 3346. Springer, 2004.
13. Michel Grundstein. *From capitalizing on Company Knowledge to Knowledge Management*, chapter 12, pages 261–287. The MIT Press, 2000.
14. Mark Klein. Capturing geometry rationale for collaborative design. In *WETICE*, pages 24–28. IEEE Computer Society, 1997.
15. Arthur Koestler. *The Ghost in the Machine*. Hutchinson, 1967.
16. N. Matta, M. Ribiere, O. Corby, M. Lewkowicz, and M. Zaclad. *Project Memory in Design, Industrial Knowledge Management - A Micro Level Approach*. Springer-Verlag, 2000.
17. F. Maturana, W. Shen, and D. Norrie. Metamorph: An adaptive agent-based architecture for intelligent manufacturing, 1999.
18. Achraf Ben Miled, Vincent Hilaire, Davy Monticolo, and Abderrfiaa Koukam. Reusing knowledge by multi agent system and ontology. In *fourth IEEE International Conference on Signal-Image Technology and Internet Based Systems, workshop Knowledge Acquisition Reuse and Evaluation*, 2008.
19. D. Monticolo, V. Hilaire, S. Gomes, and A. Koukam. A multi-agent system for building project memories to facilitate design process. *Integrated Computer-Aided Engineering*, 15(1):3–20, 2008.

20. Davy Monticolo, Samuel Gomes, Vincent Hilaire, and Patrick Serrafero. A multi-agent architecture to synthesize industrial knowledge from a plm system. In *PLEDM'06*, 2006.
21. Davy Monticolo, Vincent Hilaire, Abder Koukam, and Samuel Gomes. Ontodesign; a domain ontology for building and exploiting project memories in mechanical design projects. In *Knowledge Management in Organizations*, 2007.
22. Davy Monticolo, Vincent Hilaire, Patrick Serrafero, and Samuel Gomes. Knowledge capitalization process linked to the design process. In *KMOM'07*, 2007.
23. Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-Creating Company.* Oxford University Press, 1995.
24. Sebastian Rodriguez, Vincent Hilaire, and Abderrafiaa Koukam. Towards a methodological framework for holonic multi-agent systems. In *Proceedings of the ESAW'03 workshop*, pages 179–185, 2003.
25. Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Simple and minimum-cost satisfiability for goal models. In Anne Persson and Janis Stirna, editors, *CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2004.
26. Shum, Simon J. Buckingham, Allan MacLean, Bellotti, Victoria M. E., Hammond, and Nick V. Graphical argumentation and design cognition. *Human-Computer Interaction*, 12(3):267–300, 1997.
27. M. Ulieru and A. Geras. Emergent holarchies for e-health applications: a case in glaucoma diagnosis. In *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]*, volume 4, pages 2957– 2961, 2002.
28. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *3rd IEEE Int. Symp. on Requirements Engineering*, pages 226–235, 1997.

# Support for Modeling Roles and Dependencies in Multi-Agent Systems

Lawrence Cabac, Daniel Moldt

University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, D-22527 Hamburg
`http://www.informatik.uni-hamburg.de/TGI`

**Abstract** This paper describes the usage of component diagram models for the analysis and design of dependencies in multi-agent systems. In all software paradigms there exist dependencies between compositional system artifacts. In agent orientation, agents that require services to fulfill their goals, depend on other agents (or other entities, e.g. platforms) that offer those services. Even in simple settings the explicit modeling of these relationships is useful for analysis and construction. In complex settings, however, these models become crucial. Especially in large development groups with independent development, dependency relationships can be confusing, leading to cyclic or other unwanted dependencies in the system.
Here we present two tools which illustrate the enhancement for multi-agent systems development. The first tool introduces roundtrip engineering for modeling of dependencies. The second tool introduces new features to agent modeling. It introduces the Roles/Dependencies Diagram (R/D Diagram). This technique allows for the explicit representation of services and their relations to the roles of agents. In addition, the editing and validation of the content of the initial knowledge bases (Services, State Description, Protocols and Messages) as well as a specialization hierarchy of roles is covered by the tool. This introduces, in combination with other tools from our Paose tool set, an improved overview and insight into agent systems and their specification.

**Keywords:** nets-within-nets, net components, Renew, modeling, agents, multi-agent systems, Roles/Dependencies Diagram

## 1 Introduction

One key factor for the successful operation of multi-agent systems is the smooth communication between agents. Usually, interactions are modeled in detail using interaction diagrams and agent protocols [1,13,16]. Besides the definition of sound interactions also structural aspects are of major importance for the understanding of the architecture of a multi-agent system. One example of such a structural aspect is the dependency relation that exists between agents (often also referred to as *acquaintance*). In order to attain a goal, most agents have to rely on other agents. Thus, there exist dependencies between agents in almost every multi-agent system.

Especially in multi-agent-based applications, where the global structure (macro level) emerges from local information (micro level) the analytical modeling of resulting dependencies becomes crucial. Here we focus on the dependencies that are related to the services offered and required by the agents in the system. These relationships are often modeled in agent-oriented methodologies in acquaintance models (Prometheus) or service models (Gaia). For the multi-agent architecture Mulan we provide tool integration for the presented techniques as plugins for Renew [15], the Dependencies Plugin and the Knowledge Base Editor (KBE). In Mulan [18], offered and required services are explicitly defined in the agents' configuration files (the agents' initial knowledge bases).

We start this paper by pointing out that during the design of agent services the right level of abstraction and its variation is of great importance for the resulting system design. The distinction between soft and hard dependencies is introduced in Section 2. We propose the dependency diagram for the modeling of hard service dependencies, which resembles a composite structure diagram in UML 2.0. Section 3 presents the dependency diagram that clearly shows the dependency hierarchy of agents. With the Dependencies Plugin it is possible to create dependency diagrams from existing sources, from scratch (Section 4) or from other sources of information of dependency. The enhancement of the Dependency Diagram the Roles/Dependencies Diagrams is presented in Section 6. We present the tools that support the Dependency Diagram and the R/D Diagram and their capabilities in Section 4. As an example of the latter we present the model of the plugin structure extracted from the real dependencies of the Renew plugins.

## 2 Service Dependencies

In the context of multi-agent systems we understand services as collections of agent actions that serve a common purpose. A service is realized through one or more agent protocols and through the capabilities of the serving agent. Our notion of a service fits very well with the notion given by Zambonelli et. al. [23, p. 21]:

> The [...] services model [...] identifies the main services – intended as coherent blocks of activity in which agents will engage – that are required to realize the agent's roles, and their properties.

In general, the action to perform a service may be requested by any agent. This implies an interaction of (at least) one agent with (at least) one other agent. As a consequence, to be able to access a service, the interface (protocol description) and address of the provider has to be published. The user of the service expects that the service will be performed under certain conditions (e.g. payment, quality of service, availability). The reactive behavior of an agent is initiated by triggers (sometimes also called events). For this a mapping from (received) message types to predefined behavior exists.

During the design phase of the multi-agent system the developers have to decide on the level of abstraction of the services and their published interface. As an example we consider an agent that wants to play a board game. To be able to play the game he has to be able to access the *board game service*. This may include in its interface description all necessary interaction protocols for the whole game. However, this does not take into account composability or scalability. In opposition, the game services can be implemented as several smaller services, which can be offered by different agents. Thus, in a finer level of abstraction, the developers can decide to design following services for the board game: *board control*, *accounting*, *game control*, *trading*. Now, several distinct parts of the system are identified, with certain responsibilities and capabilities.

The corresponding design artifacts for the (early) structural design in the development of multi-agent system are roles. They are typically defined as role descriptions. In a flexible way the developers may decide that all roles can be executed by one agent or each role is implemented by one single agent.

The challenge for the developers is to find the right level of abstraction, i.e. abstract enough to get an idea of the offered services as a whole and detailed enough to recognize if two agents perform similar tasks. Through their flexibility in regard to the choice of the level of abstraction the services are suitable for modeling the overall structure of big as well as small systems without designing too complex or too trivial representations in the models. Most agents use services of other agents to accomplish their goals or even to provide their own services (by delegation). Thus, if an agent requires a service from another agent, we recognize a dependency between agent role (which is responsible for the agents behavior) and offered service. This we call hard dependency because this is defined during design time and is thus of a static nature.[1] Hard dependencies describe a minimum set of services that are required by an agent to fill out a role.

Another dependency exists between agents that are not in a service provider-/requester relationship but communicate with each other on a different basis. These agents are on a personal acquaintance level between the two identities of agents. We call these dependencies soft dependencies (sometimes dynamic dependencies).

While soft dependencies are dynamic and thus not modeled a priori, hard dependencies are explicitly specified by the developer. However, information about soft dependencies can easily be gathered during runtime of the multi-agent system and may result in an acquaintance model describing the communication structure and can be presented as a social network.[2]

[1] Note that this is a static dependency between agent roles and services and not between agents or roles. The requester can always choose from a number of service providers.

[2] This is done within our tool set by another tool (MulanSniffer, see[6]) and not the topic of this paper.

In MULAN/CAPA applications, hard dependencies (*required services*) are defined in the initial knowledge base file as FIPA[3]-compliant *service descriptions*. These service descriptions are published by the providers at the directory facilitator (DF) and this data can be queried by the requester.[4] Thus the service provider can be found by the service requester. To request the service the agent has to abide by well-defined protocols. A typical means to trigger service is to send an *action request*, which tells the receiver to perform a task.

We apprehend protocols as implementations of a complex agent actions that are assigned to one or more services. In many methodologies, as also in MULAN/CAPA, conversation patterns are typically defined as interaction protocols [5].

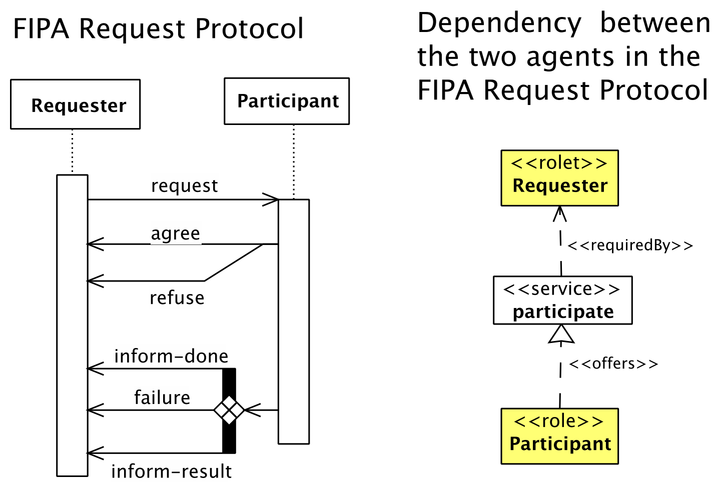


**Figure 1.** FIPA Request protocol and a representation of dependencies.

In this paper we propose a modeling technique for hard dependencies. We recognize a dependency between an agent and a service that is offered by another agent, if one agent requests another agent to perform a task or an action.

In many cases – but not always – the interactions between requester and provider follow the FIPA Request protocol or a similar one and use an action request as a performative. To illustrate the technique we use the FIPA Request protocol [14] as a prototypical protocol presented in Figure 1. The Participant in the Request protocol offers a service to perform a certain task – let's say the service *participate*. The Requester wishes a task *task* to be performed by the

[3] Foundation for Intelligent Physical Agents, [13]

[4] Note that the directory facilitator offers the service of registering and querying information about services in the multi-agent system. However, this is a mandatory service that is always accessible on FIPA-compliant platforms.

Participant. This implies that the Requester sends a message (action request) to the Participant and waits for an answer if necessary.

The service offered by the Participant is completed with an answer to the initial request. Thus a hard dependency exists, which is modeled in the right part of the figure as a fragment of a dependency diagram. Services can be required by several agents and they can also be offered by multiple agents. Thus, the dependency does not exist directly between the two agents (or their roles), instead – as pointed out above – the dependency exist between an agent and an offered service.

In general, we seek for a hierarchical structure in a dependency diagram. This allows for code reuse in the system, composability and easy reconfiguration. Interdependencies (cyclic dependencies) between agents are undesirable, firstly because they can cause deadlocks in the systems configuration and secondly because they complicate the substitution of agents. Also unmet dependencies usually cause trouble in the system configuration. We believe that through explicit modeling such problematic aspects in a system design can be found and developers can be supported in the process of eliminating them. Figure 2 shows

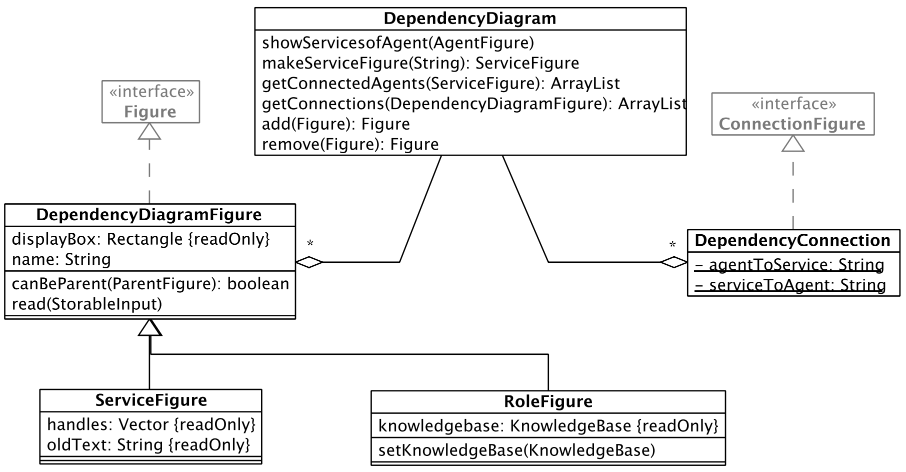

DependencyDiagram
showServicesofAgent(AgentFigure)
makeServiceFigure(String): ServiceFigure
getConnectedAgents(ServiceFigure): ArrayList
getConnections(DependencyDiagramFigure): ArrayList
add(Figure): Figure
remove(Figure): Figure
«interface»
Figure
«interface»
ConnectionFigure
DependencyDiagramFigure
displayBox: Rectangle {readOnly}
name: String
canBeParent(ParentFigure): boolean
read(StorableInput)
*
*
DependencyConnection
- agentToService: String
- serviceToAgent: String
ServiceFigure
handles: Vector {readOnly}
oldText: String {readOnly}
RoleFigure
knowledgebase: KnowledgeBase {readOnly}
setKnowledgeBase(KnowledgeBase)

**Figure 2.** Meta model of dependecies diagrams.

the elements of the diagram as a class diagram. A dependency diagram consists of role figures, service figures and dependency connections. They own the stereotypes «role»,«service», «offers» and «requiredBy».

## 3 Modeling Service Dependencies

For the modeling of service dependencies we employ UML component diagrams. However, we exclusively use the detailed version, so that the service is represented explicitly in the model and we modify the syntax slightly.

Usually, component diagrams are used to model the constitution of replaceable software constructs and their relationships. Besides the components and the interfaces also classes and objects are used in component diagrams [20, p. 139-171]. In the agent context, where we deal with agents, services and the dependency relations, we use the elements of the component diagram adapted to those needs.

A service is an abstraction of a set of (complex) agent actions that serve a common purpose. Several services may be provided by one agent and several agents may offer the same services. Our notion of a *service* (see Section 2) is very similar to the concept of an *interface* in the UML superstructure [20, p. 82]:

> An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract.

Thus, it seems straightforward to model services in a way that is similar to the modeling of interfaces. We add the stereotype «service» to the elements in our models. The stereotype is meant to express the differences. First, we model in a completely different context. Second, in contrast to an object, an agent has the ability to break contracts, so services are indeed also an obligation to fulfill a specified task but there is no definitive certainty that this will (or can) be done. Third, we like to use classes and interfaces together with agents and services within one model. The introduction of a new stereotype gives us the ability to do so without introducing confusion into the models.

From the software engineering viewpoint agents are often regarded as special components. If one is to take this position, again it seems straightforward to model agent roles as components together with the offered and required services. We introduce the stereotype «role» for the depicted agent roles.

In the detailed version of the component diagram the relationship of agents to implemented interfaces are depicted as dashed arcs with a triangle arrow tip. We follow this notation and draw a dashed arc from the agent that offers the service to the service itself. Again, we explicitly distinguish between arcs and provide the stereotype «offers».

Required interfaces are modeled in the component diagram by a dashed arc with an open line arrow tip (stereotype «use»). This arc points at the interface used. In contrast, in Dependency Diagrams we draw the arcs in the opposite direction and – accordingly – offer the stereotype «requiredBy»[5]. By doing so we get a relation chain from offering role to offered service and from the service

[5] Note that we sometimes omit the stereotypes on the arcs, if the context is clear.

to the role that requires the service. At first glance this change seems odd but we achieve the possibility to model hierarchical structures. In Section 2 we pointed out the benefits of hierarchical dependencies.

Figure 3 shows an example dependency diagram as described above. To better distinguish between agents and services the roles possess a colored background. The figure shows a snapshot of a workflow management system in development,

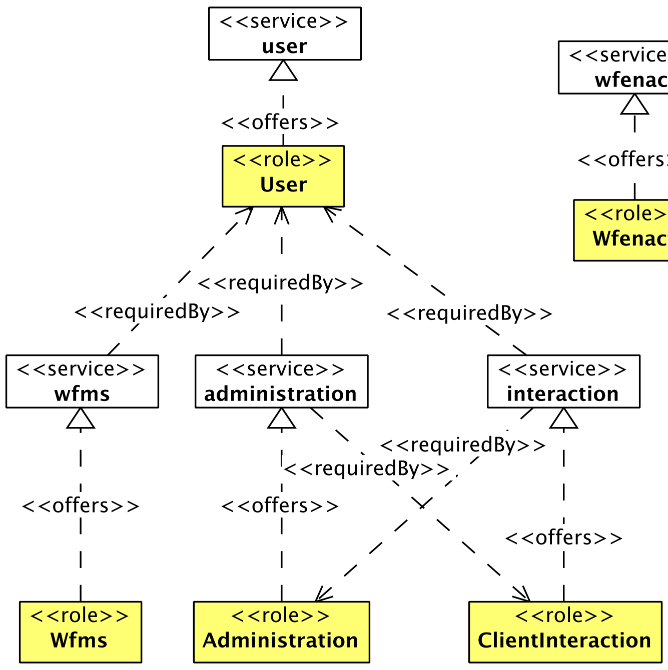


**Figure 3.** Dependencies of a workflow management system in development.

giving an overview of the agent roles in the system.

In the model, a developer can easily identify potentially problematic areas. Usually, this is a hard task because the information resides distributed in several local property files, the initial knowledge bases, which can be distributed. In this very simple example we are able to identify two problems: First, there exists a cyclic dependency between the roles `Administration` and `ClientInteraction`. A cyclic dependency may indicate that the agent roles could be fused to one role since they are so tightly coupled and can only act as pair or, as in this case, that the developers of one of the roles have a misconception of the tasks of the designed role. Second, the agent `Wfenact` is not connected to the other agents.

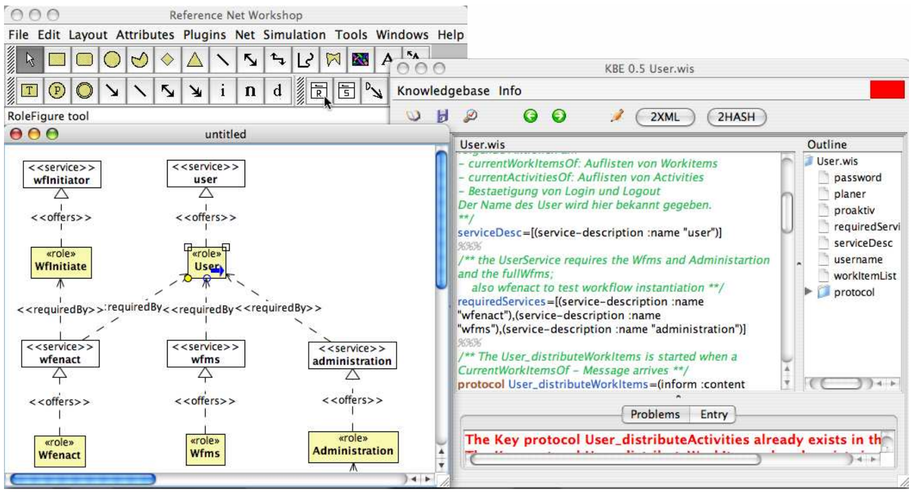
An isolated agent that offers a service means that this role does not interact with other roles within the service relationship.[6]

Here, since the diagram has been taken during the development phase, the depicted configuration is not final. During analytical examination of the designed structure of the service dependencies this means that still some work has to be done by the developers. Both situations are undesirable and should be changed in the further development process. The developers may be supported in finding such structural anomalies automatically by the modeling tool that is described in the next section.

We employ for the purposes of finding anomalies two simple checks (acyclic property check and connectedness property check) from another plugin of Renew, the NetAnalysis Plugin. For this means we have implemented a conversion of the diagram into a net structure, which can directly be analyzed by the NetAnalysis Plugin. Another possibility – yet to be implemented – is a direct feeding of the graph structure into the checking algorithms.

## 4 Description of Tools

The tool that handles Dependency Diagrams is the Dependencies Plugin for Renew [15]. The plugin has two main functionalities. It generates the dependency diagram from existing Mulan knowledge base files and it offers tools for creating and editing Dependency Diagrams.



```
- currentWorkItemsOf: Auflisten von Workitems
- currentActivitiesOf: Auflisten von Activities
- Bestaetigung von Login und Logout
Der Name des User wird hier bekannt gegeben.
**/
serviceDesc=[(service-description :name "user")]
%%%
/** the UserService requires the Wfms and Administartion
and the fullWfms;
   also wfenact to test workflow instantiation **/
requiredServices=[(service-description :name
"wfenact"),(service-description :name
"wfms"),(service-description :name "administration")]
%%%
/** The User_distributeWorkItems is started when a
CurrentWorkItemsOf - Message arrives **/
protocol User_distributeWorkItems=(inform :content
```

```
The Key protocol User_distributeActivities already exists in th
```

**Figure 4.** Screenshot of the development with the dependency diagram tool

[6] Maybe it interacts on a different basis, e.g. in negotiations or as user.

Figure 4 shows a screen shot of the development of the workflow management system with the dependency diagram tool. In the upper left corner is the RENEW menu bar with the standard palettes and the dependency diagram palette. Beneath, the dependency diagram is shown. In the diagram the role figure `User` is selected. On the right hand side of the figure the KBE with the knowledge base of the agent `User` is depicted.

For the generation of a diagram the tool searches (recursively) for knowledge base files in a user-defined directory. For each knowledge base found a role figure is created. A knowledge base contains a list of offered services and a list of the agent's required services. For each service in the lists the corresponding service figure and the role figure are connected. A new service figure is created, if the service is not already present in the drawing. For the user's convenience the tool provides a simple automatic layout mechanism.

In addition to the possibility to use all standard drawing tools of RENEW, the plugin offers new editing functions. These are offered as three tools for editing dependency diagrams: a *role figure* tool, a *service figure* tool and a *dependency connection* tool (see the last three items in the lower RENEW tool bar in Figure 4). The dependency connection tool is used for drawing the arcs between agent role and service figures. The arrow type and the inscription depend on the direction of the arc – see Section 3. Arrows are adapted automatically while they are drawn so that the two arc types can be drawn with the same tool.

A special function of the dependency diagram is the KBE handle, which is part of the role figure. It connects the Dependency Diagram to the KBE. With a click on the handle (a blue arc in the bottom right corner of the figure that is visible when the figure is selected), the knowledge base of the agent role is opened in the KBE for further inspection or editing. This is especially useful for debugging purposes and during the design of new knowledge base files.

By first generating a dependency diagram and then editing it, one faces an inconsistency between the diagram and the code it is generated from. To minimize such conflicts between diagrams and knowledge bases the tool realizes a round-trip engineering system. It preserves the consistency of knowledge bases and dependency diagram by automatically transferring changes in the dependency diagram to the knowledge bases. For example, when the service `administration` is connected to the agent role `User` via a dependency arc, a new service-description is inserted in the list of the required services in the knowledge base. This also works in the other direction. However, changes in the knowledge base are not transferred immediately to the dependency diagram, but as soon as the knowledge base is saved. A detailed description of the dependency diagram tool and the round-trip engineering system can be found in [10].

R/D Diagrams are drawn directly in the enhanced version of the KBE. The KBE started out as a simple but convenient editor for the original properties knowledge base files. It included syntax checking for ontology concepts represented in the Semantic Language (SL) by employing the parser as validator.

The enhanced version of the KBE was grounded on three aspects. First, the redefinition of the knowledge base format and by that the possibility to allow specialization/generalization of roles. Second, the implementation of the editor as diagram reflecting the hierarchy of roles (i.e. a class diagram-like model). Third, the integration of the representation of the macro level, i.e. the dependencies between agent roles.

The enhanced version features interactive drawing of R/D Diagrams (including role definitions, service definitions, dependencies relations), direct inspection of the XML code, generation of agent role descriptors and initial knowledge bases and the validation of the graphs and entries.

## 5 Dependency Diagrams for Plugin Systems

The Dependencies Plugin is not bound to agent role dependencies but can also be used to model other component-based, hierarchically structured systems that own inter-dependencies. Another example of such a hierarchical system in our context is the plugin structure of Renew. Similar to the agents in the Mulan-system every plugin contains a configuration file in which the required and offered services (besides other things) of the plugins are declared. Therefore the dependency diagram tool can be used to generate a dependency diagram of the plugin structure without much additional effort. A function to remove these transitive arcs in the diagram is therefore very useful but has not been realized yet, since this is not the main focus of the Dependencies Plugin.

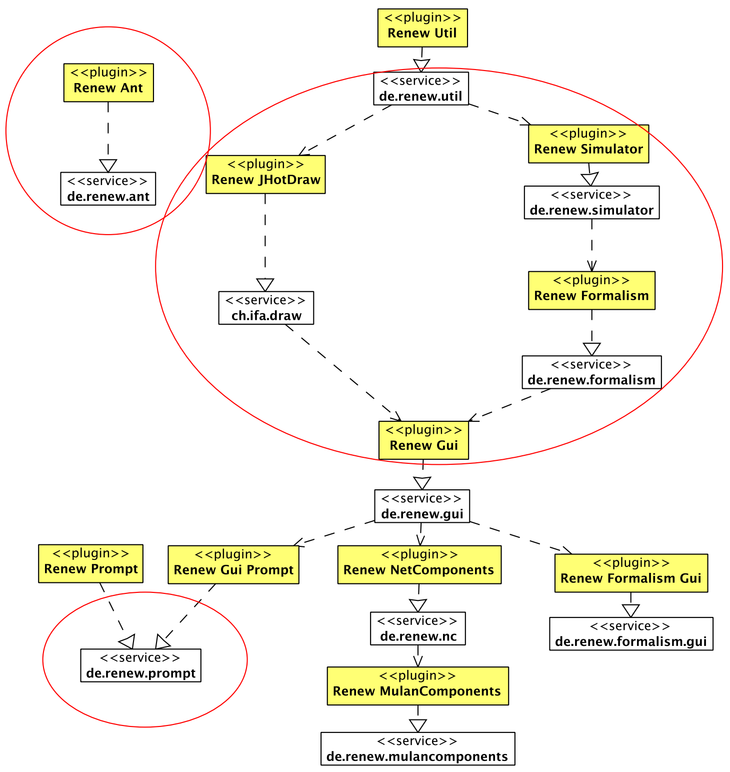
The example in Figure 5 shows a fragment of the Renew plugin structure without transitive arcs (manually beautified).

The model shows the connection between the plugins GUI and Simulator as well as several other related extensions. Renew Util is a plugin that encapsulates basic libraries used throughout the system. The GUI itself is divided into two parts one is an adaption of the graphical framework JHotDraw (Renew JHotDraw) the other (Renew Gui) offers Petri net specific drawing features.

We can identify some oddities from the diagram. First, Renew Ant is not connected. This is perfectly all right because this plugin provides build support of which no functionality is used by any other plugin at runtime. Second, Renew Prompt and Renew Gui Prompt implement the same service (which is actually not used by any other plugin but offers a user interface for the user (developer) of Renew, the only difference is that one provides a command line prompt convenient for remote access and the other is implemented as *Swing* GUI.

Third, the Simulator/Formalism and the JHotDraw plugin are independent of each other, meaning one can run the system and use the common drawing features of Renew JHotDraw without the Simulator plugin loaded and it is also possible to run a simulation without the GUI.[7] However, if the Renew Gui plugin is loaded, both sides of the system (Simulator and JHotDraw Gui) have to be loaded.

[7] It is actually possible to attach or detach the GUI, while the simulation is running.



<<plugin>> Renew Util
<<service>> de.renew.util
<<plugin>> Renew Ant
<<service>> de.renew.ant
<<plugin>> Renew JHotDraw
<<service>> ch.ifa.draw
<<plugin>> Renew Simulator
<<service>> de.renew.simulator
<<plugin>> Renew Formalism
<<service>> de.renew.formalism
<<plugin>> Renew Gui
<<service>> de.renew.gui
<<plugin>> Renew Prompt
<<plugin>> Renew Gui Prompt
<<service>> de.renew.prompt
<<plugin>> Renew NetComponents
<<service>> de.renew.nc
<<plugin>> Renew MulanComponents
<<service>> de.renew.mulancomponents
<<plugin>> Renew Formalism Gui
<<service>> de.renew.formalism.gui

**Figure 5.** Fragment of the Renew plugin dependencies.

## 6 Roles and Dependencies

Dependencies of agent roles can be modeled nicely with the Dependencies Plugin as a Dependency Diagram. Through the round-trip integration together with the Knowledge Base Editor we achieve a convenient way to model constructively (during design tasks) and analytically (during evaluation, testing).

The main disadvantage in this approach does not arise from the techniques. It lies in the artifacts that are used to represent the initial knowledge bases. These are simple property files, which are transformed into knowledge base instances by an assisting process during initialization. This process makes use of the reflection mechanism to instantiate runtime objects.

This is a flexible and simple technique to realize instances of knowledge bases. However, since property files are not extensible there is no ground to support the modeling of role hierarchies or even multiple knowledge base files per agent type, which leads to code duplication for agent types that share roles. The relationship of one knowledge base file (agent role descriptors) for one agent type is not feasible for an efficient and scalable implementation. Instead, we prefer to use extensible artifacts, which can be modeled in an inheritance hierarchy similar to class inheritance. The technique of choice for such extensible artifacts is the Extensible Mark-up Language (XML). Thus we are able to make use of composed as well as specialized role types for the agents.

Being able to implement the code for a role hierarchy, we are able to model not only the dependencies of agent roles, but also the hierarchy of roles. Moreover, we are able to model both hierarchy and dependencies in one diagram. This leads to the Roles and Dependencies Diagram (R/D Diagram). Here the different roles appear in a hierarchy of (specialized) role types together with services and the dependencies of the role types. To model these two aspects in one diagram makes sense, since dependencies are also inherited, thus in the combination of both aspects the inherited dependency is easily found and the structure of the dependencies benefits from the clustering of roles through generalization.

A prototypical implementation of a Knowledge Base Editor (Version 2) exists that supports the XML knowledge base format. It incorporates the model in its user interface. The technique is designed to show the dependencies as explained in Section 3 as well as the hierarchies of roles.

Role descriptors (initial knowledge base files in XML notation) can be created and edited directly in the tool, which also validates entries on the fly. Models can be centralized as well as fragmented. If fragmented the joining (distributed) elements of the diagrams are the services and the abstract roles. Because of the generality these can be in several models and if the models are distributed those artifact have to be present in all affected models.

From the initial models (R/D Diagram), which are the main design artifacts, agent role descriptors (ARD in XML notation) can be generated. These artifacts are sufficient to initialize agent types defined with the same tool in a simple agent model, which maps agents to roles (compare with the agent models of Gaia [23]). Alternatively, also the merged ARD descriptions can be created for the convenient use of agent knowledge bases (KB) in other contexts.

We return to the example of a workflow management system to illustrate the power and the usage of the R/D Diagram. Figure 6 shows the dependencies of the reimplementation of the WFMS done with the enhanced tools. The system features several agent roles to form the WFMS, an agent role for the workflows and an agent role as a placeholder (proxy) for the user of the system. The user can be in different user roles (administrator, executor, initiator, etc.), which is reflected by the fact that user agents also own user roles.

The WFMS core is represented by an agent that owns the role WFMS (the WFMSAgent, which can be regarded as a singleton agent). This agent delegates several tasks of the WFMS to its participants: e.g. agents that own the role WFEngine or WFES (workflow enactment service).[8]

The dependencies in the model reflect the domain specific constraints. For instance the AccountManager is capable to authenticate the permissions of the participants of the WFMS and it offers this as a service. The WFES can thus delegate the task of authentication to the AccountManager.

[8] All notions and qualifiers are directly taken from the definition of a WFMS given by the Workflow Management Coalition (WfMC [22,21]).

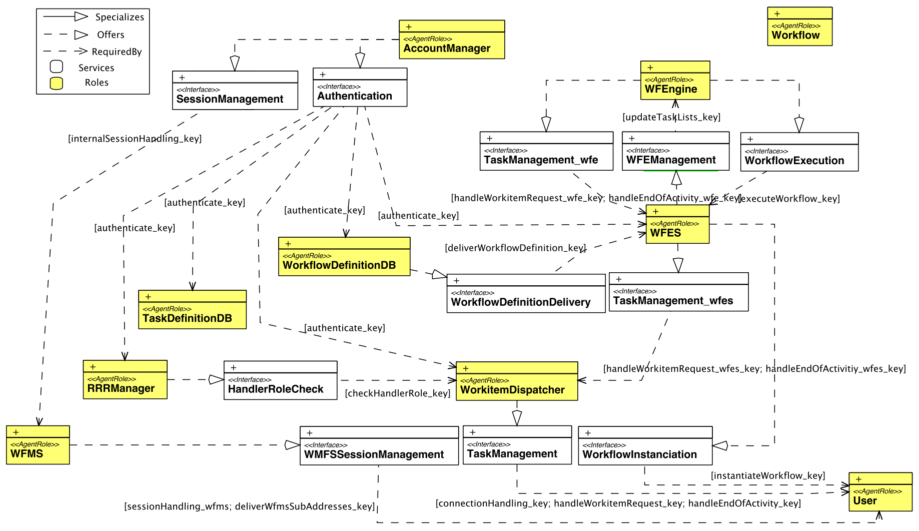

Specializes
Offers
RequiredBy
Services
Roles
<<AgentRole>> AccountManager
<<AgentRole>> Workflow
<<Interface>> SessionManagement
<<Interface>> Authentication
<<AgentRole>> WFEngine
[updateTaskLists_key]
<<Interface>> TaskManagement_wfe
<<Interface>> WFEManagement
<<Interface>> WorkflowExecution
[internalSessionHandling_key]
[handleWorkitemRequest_wfe_key; handleEndOfActivity_wfe_key]
[executeWorkflow_key]
[authenticate_key]
[authenticate_key]
[authenticate_key]
[authenticate_key]
<<AgentRole>> WFES
<<AgentRole>> WorkflowDefinitionDB
[deliverWorkflowDefinition_key]
<<Interface>> WorkflowDefinitionDelivery
<<Interface>> TaskManagement_wfes
<<AgentRole>> TaskDefinitionDB
[authenticate_key]
[handleWorkitemRequest_wfes_key; handleEndOfActivitiy_wfes_key]
<<AgentRole>> RRRManager
<<Interface>> HandlerRoleCheck
[checkHandlerRole_key]
<<AgentRole>> WorkItemDispatcher
<<AgentRole>> WFMS
<<Interface>> WMFSSessionManagement
<<Interface>> TaskManagement
<<Interface>> WorkflowInstanciation
[instantiateWorkflow_key]
<<AgentRole>> User
[sessionHandling_wfms; deliverWfmsSubAddresses_key]
[connectionHandling_key; handleWorkitemRequest_key; handleEndOfActivity_key]

**Figure 6.** Dependencies in the WFMS (version 2).

Note that the agent roles WFES and WFEngine are in a cyclic dependency meaning that they are tightly coupled. One cannot perform the tasks in the WFMS context without the other. However, this cycle is not resolvable and also those agent roles cannot (normally) be included within one agent as the WFES takes up a manager role for the WFEngine agents, which execute the workflows. Also the WFES could create new agents for the execution of workflows on demand.

Figure 7 shows a possibility for generalization of agent roles. Basically all agents that perform a task within the WFMS (and offer services to other WFMS agents) have to able to use the authentication. Here a generalization is obvious. Also all agents are CapaAgents because we are running a Capa engine solely with standard agents.

Figure 8 integrates both into one R/D Diagram. Through the generalization of agent roles we are able to reduce the number of dependency relations considerably. However, much of this improvement is again taken back since we have to add generalization arcs to the diagram. Also the fact that we combine (ideally) two hierarchies in one diagram does not make the layout easier, since sometimes the resulting graph is not even planar. Nevertheless, the diagram shows a numerous aspects of a complex system and we are also able to switch to one or the other perspective (hierarchy of roles, or dependencies).

So far we have only discussed the macro structure of the R/D Diagram. To inspect an edit the micro structure the elements can be expanded (the "+" at the top of the figure acts as handle). The expanded role descriptor in Figure 9 reveals several sections (such as RequiredServices, StateDescription, Protocols

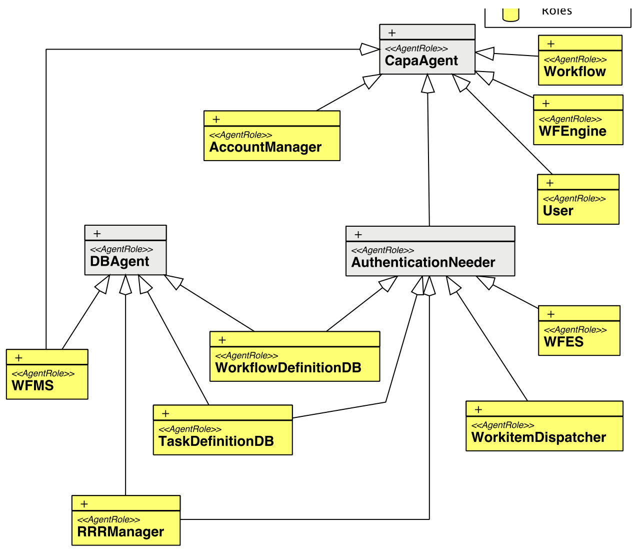


**Figure 7.** Role hierarchy of the model.

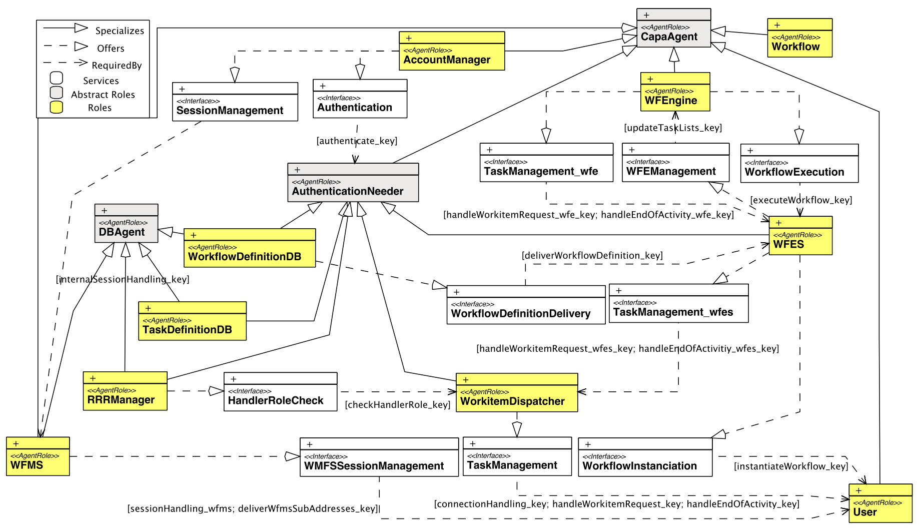
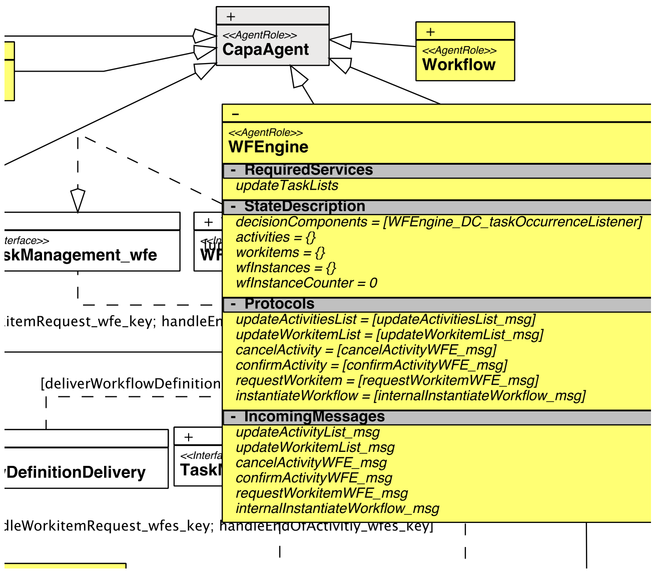

Specializes
Offers
RequiredBy
Services
Abstract Roles
Roles
<<AgentRole>> CapaAgent
<<AgentRole>> Workflow
<<AgentRole>> AccountManager
<<Interface>> SessionManagement
<<Interface>> Authentication
<<AgentRole>> WFEngine
[updateTaskLists_key]
[authenticate_key]
<<AgentRole>> AuthenticationNeeder
<<Interface>> TaskManagement_wfe
<<Interface>> WFEManagement
<<Interface>> WorkflowExecution
[executeWorkflow_key]
[handleWorkitemRequest_wfe_key; handleEndOfActivity_wfe_key]
<<AgentRole>> DBAgent
<<AgentRole>> WorkflowDefinitionDB
<<AgentRole>> WFES
[deliverWorkflowDefinition_key]
[internalSessionHandling_key]
<<AgentRole>> TaskDefinitionDB
<<Interface>> WorkflowDefinitionDelivery
<<Interface>> TaskManagement_wfes
[handleWorkitemRequest_wfes_key; handleEndOfActivitiy_wfes_key]
<<AgentRole>> RRRManager
<<Interface>> HandlerRoleCheck
[checkHandlerRole_key]
<<AgentRole>> WorkitemDispatcher
<<AgentRole>> WFMS
<<Interface>> WMFSSessionManagement
<<Interface>> TaskManagement
<<Interface>> WorkflowInstanciation
[instantiateWorkflow_key]
<<AgentRole>> User
[sessionHandling_wfms; deliverWfmsSubAddresses_key]
[connectionHandling_key; handleWorkitemRequest_key; handleEndOfActivity_key]

**Figure 8.** R/D Diagram of the WFMS (version 2).

and IncommingMessages) and in each section several entries. Sections and entries can be added or removed or edited directly within the tool.



<<AgentRole>>
CapaAgent
<<AgentRole>>
Workflow
<<AgentRole>>
WFEngine
- RequiredServices
updateTaskLists
- StateDescription
decisionComponents = [WFEngine_DC_taskOccurrenceListener]
activities = {}
workitems = {}
wfInstances = {}
wfInstanceCounter = 0
- Protocols
updateActivitiesList = [updateActivitiesList_msg]
updateWorkitemList = [updateWorkitemList_msg]
cancelActivity = [cancelActivityWFE_msg]
confirmActivity = [confirmActivityWFE_msg]
requestWorkitem = [requestWorkitemWFE_msg]
instantiateWorkflow = [internalInstantiateWorkflow_msg]
- IncomingMessages
updateActivityList_msg
updateWorkitemList_msg
cancelActivityWFE_msg
confirmActivityWFE_msg
requestWorkitemWFE_msg
internalInstantiateWorkflow_msg
terface>>
skManagement_wfe
itemRequest_wfe_key; handleEr
[deliverWorkflowDefinition
DefinitionDelivery
<<Interfa
TaskM
dleWorkitemRequest_wfes_key; handleEndOfActivity_wfes_key]

**Figure 9.** Fragment of the R/D Diagram showing the WFEngine role expanded to show details.

This the R/D Diagram offers for the developer of multi-agent system the possibility to inspect the system roles and dependencies an a broad level (macro, see Figure 8) and on a detailed level as well (micro level, see Figure 9). Additionally, the other views (role hierarchy view and dependencies view) are also present in the models.

## 7 Related Work

Our definition of hard dependencies is comparable with the definition of service dependencies in [3]. A definition of soft dependencies can be found in the PASSI method. There a soft dependency exists if a service is not required, but "helpful or desirable" [9, p. 6]. This notion conflicts with ours where soft dependencies subsume the hard dependencies.

Most software developing methodologies contain a technique for modeling some kind of dependencies between their components. In the following paragraph we will consider several examples from the agent oriented context, TROPOS and AGR (Agent/Group/Role), and have a look how the dependency diagram can be used in other component based domains.

The TROPOS methodology distinguishes four kinds of dependencies between agents, from hard dependencies (resource) to soft ones (soft-goal). [19] shows how TROPOS dependency relations can be expressed in UML for real time systems.

A hard dependency in our definition could be a resource dependency, a goal dependency or a task dependency in TROPOS, depending on the kind of service. We want to abstract as much as possible from the agent internals to get a clear image of the system structure, so the distinction between different kind of services in terms of the underlying action is not useful for our needs.

Another agent oriented modeling technique, that describes dependencies between agents is AGR (Agent/Group/Role). Ferber et. al. [12] show how the organizational structure of an agent-based system can be modeled using the AGR technique. One of the proposed diagrams, the organizational structure diagram, shows roles, interactions and the relations between roles and interactions. This diagram is comparable to the Dependency Diagram. In both diagrams an arc from an agent (or from the agent role) means that the agent starts an interaction. Differences between the diagrams come from additional elements in the organizational structure diagram. First also the groups to which the roles belong are modeled. Second, the situation that every agent in a specific role must be member in another role is modeled as a direct relation between the two roles. In Mulan/Capa-systems there are (for now) no elements like groups or roles, so the advanced modeling possibilities of an organizational structure diagram is not suitable in this context.

As well as in the agent context, also in other component based systems it is important to model the dependencies between components. One example of a well known component system is the Eclipse framework with its numerous plugins. The visualization of the dependencies between different plugins is complex and no sufficient commercial tools exist that can visualize the structure of the whole system appropriately.

Gaia focuses strongly on the organizational modeling [23]. One of the important models is the service model. Our Dependency Diagram can be regarded as an implementation of the Gaia service model. However, Gaia does not recognize hierarchical roles.

In Prometheus [17] acquaintances are modeled but they do not result in agent (role) dependencies. Roles are not modeled here, instead the focus lies on agents. The system model in Prometheus gives a good overview of the system comparable with the overview of the R/D Diagram. It is much more detailed and but does not explicitly show any dependencies except the interaction protocols or messages that connect agents. The structure of the system model reflects the one of the acquaintances model.

## 8 Conclusion and Outlook

In this work we present techniques to explicitly model the dependencies between agent roles and services. The benefit of these techniques are intuitive diagrams, which are close to UML standard diagrams, derived from component and class diagrams and at the same time suitable for expressive modeling in the context of agent-oriented methodologies. The use of the proposed diagrams helps software developers of multi-agent systems to get an overview of the overall structure of a system and to identify desired or undesired dependencies hidden in the (possibly distributed) source code. The presented techniques can be applied in constructive as well as analytical modeling. Furthermore, the diagrams can also be valuable during presentation and documentation purposes.

Mainly, we have shown that common standard methods of modeling can be successfully applied to system modeling in the agent-oriented context. Moreover, even for this area, where the standard notions, concepts and perspectives fail to be effective, the commonly known techniques can be applied, if they are adapted to the needs apparent in the paradigm of multi-agent system. With the R/D Diagram one perspective that is often addressed in agent-oriented methodologies is covered in a suitable way which is not alien to object-oriented developers either.

With the Dependencies Plugin for Dependency Diagrams, including the round-trip engineering system capabilities, developers can generate and use dependency diagrams without additional effort. The Dependency Diagram always shows an up-to-date documentation of the system due to its round-trip engineering integration. The application of the Dependency Diagram technique can be extended to other domains such as component-based, plugin-based or service-based software systems. The current version of the Dependencies Plugin, for example, can generate diagrams that show the Renew plugin dependencies. Because the Renew plugins and the plugin system were conceptually based on agent technology, this additional functionality was achieved with only little effort (compare with [8]).

Standard validation techniques can be applied to the generated diagrams directly within our tool set. Thus an analysis of the structure of the diagram is possible, easily applied and offers valuable additional information about the system. As a presented example Net Analysis Plugin allows to check for simple properties in graphs, properties such as the absence of cycles or connectedness.

With the Roles/Dependencies Diagram (R/D Diagram) it is possible to integrate the advantages of dependencies modeling together with the structural modeling of specialization hierarchies of agent roles.

In parts the changing of the implementation language for the design artifacts to XML, which is mergeable, played a major role in this improvement. The enhanced version of the Knowledge Base Editor (KBE) includes the diagram in its interface, allows to edit knowledge base entries directly within the diagram and offers on-the-fly syntax checking and validation. However, the analytical modeling, i.e. the round-trip engineering capabilities, of the Dependencies Plugin are not yet integrated into the KBE. While the tools have been successfully applied in some experimental projects, we currently aim for an integration of the round-trip engineering system of the Dependencies Plugin into the KBE Plugin, which already offers flexible editing and verification power.

Future work aims at an integration of techniques within our tool set. A connection between services as used in the R/D Diagrams with the interaction diagrams could fill the gap between service descriptions and service / role implementations, i.e. triggers for certain actions. We would like to apply our techniques to other domains and evaluate them against other agent-oriented methods, for instance Jadex [4,2]. The work on dependency modeling and roles and dependencies modeling presented in this paper are examples of the block of a broader approach on agent-oriented software engineering based on Petri nets and other graphical modeling formalisms Paose (Petri net-based Agent-Oriented Software Engineering). Included in this field of research are the frameworks Mulan and Capa as well as efforts of multi-agent application development (see [7,11,18]).

## References

1. AUML. Agent UML. Webpage, 2004. `http://www.auml.org/`.
2. Lars Braubach and Alexander Pokahr. Jadex. website, July 2008. `http://vsis-www.informatik.uni-hamburg.de/projects/jadex/`.
3. Lars Braubach, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. Deployment of distributed multi-agent systems. In Franco Zambonelli Marie-Pierre Gleizes, Andrea Omicini, editor, *5th International Workshop on Engineering Societies in the Agents World*, pages 261–276. Springer-Verlag, Berlin, 2005.
4. Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. *Software Agent-Based Applications, Platforms and Development Kits*, chapter Jadex: A BDI Agent System Combining Middleware and Reasoning. Birkhäuser Book, 2005.
5. Lawrence Cabac. Modeling agent interaction protocols with AUML diagrams and Petri nets. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, December 2003.
6. Lawrence Cabac, Till Dörges, and Heiko Rölke. A monitoring toolset for Petri net-based agent-oriented software engineering. In Rüdiger Valk and Kees M. van Hee, editors, *29th International Conference on Application and Theory of Petri Nets, Xi'an, China*, volume 5062 of *Lecture Notes in Computer Science*, pages 399–408. Springer-Verlag, June 2008.
7. Lawrence Cabac, Michael Duvigneau, Michael Köhler, Kolja Lehmann, Daniel Moldt, Sven Offermann, Jan Ortmann, Christine Reese, Heiko Rölke, and Volker Tell. PAOSE Settler demo. In *First Workshop on High-Level Petri Nets and Distributed Systems (PNDS) 2005*, Vogt-Kölln Str. 30, D-22527 Hamburg, March 2005. University of Hamburg, Department of Computer Science.
8. Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Applying multi-agent concepts to dynamic plug-in architectures. In Joerg Mueller and Franco Zambonelli, editors, *Agent-Oriented Software Engineering VI: 6th International Workshop, AOSE 2005, Utrecht, Netherlands, July 21, 2005. Revised Selected Papers*, volume 3950 of *Lecture Notes in Computer Science*, pages 190–204. Springer-Verlag, June 2006.
9. M. Cossentino and C. Potts. PASSI: a process for specifying and implementing multi-agent systems using UML. `http://www-static.cc.gatech.edu/classes/AY2002/cs6300_fall/ICSE.pdf`.
10. Ragna Dirkner. Roundtrip-Engineering im PAOSE-Ansatz. Diploma-thesis, University of Hamburg, Department of Informatics, 2006.

11. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions*, volume 2585 of *Lecture Notes in Computer Science*, pages 59–72. Springer-Verlag, 2003.
12. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizational view of multi-agent systems. In Paolo Giorgini, Jörg Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV*, pages 214–230, 7 2003.
13. FIPA. Foundation for Intelligent Physical Agents, 2007. `http://www.fipa.org`.
14. Foundation for Intelligent Physical Agents. *FIPA Request Protocol Specification*, version 2002/12/06 edition, 2002.
15. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – the Reference Net Workshop. Available at: `http://www.renew.de/`, July 2008. Release 2.1.1.
16. James Odell, H. Van Dyke Parunak, and Bernhard Bauer. Extending UML for agents. In Gerd Wagner, Yves Lesperance, and Eric Yu, editors, *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, 2000.
17. Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems : A Practical Guide.* Wiley Series in Agent Technology. Chichester [u.a.] : Wiley, 2004. isbn:0-470-86120-7, Pages 225.
18. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications.* Logos Verlag, Berlin, 2004.
19. Carla T. L. L. Silva and Jaelson Castro. Modeling organizational architectural styles in UML: The Tropos case. In Oscar Pastor and Juan Sánchez Díaz, editors, *Anais do WER02 - Workshop em Engenharia de Requisitos*, pages 162–176, 11 2002.
20. Unified modeling language: Superstructure. `http://www.omg.org/docs/formal/05-07-04.pdf`, July 2005.
21. WfMC. Workflow Management Coalition (WfMC). `http://www.wfmc.org/`, 2005.
22. WfMC. Workflow reference model. `http://www.wfmc.org/standards/model.htm`, 2005.
23. Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, July 2003.

# Multi Agent Organisation for Coevolutionary Optimization

Grégoire Danoy[1], Pascal Bouvry[1], and Olivier Boissier[2]

[1] FSTC/CSC/ILIAS, University of Luxembourg
6 Rue R. Coudenhove Kalergi, L-1359 Luxembourg
gregoire.danoy@uni.lu pascal.bouvry@uni.lu
[2] SMA/G2I/ENSM.SE, 158 Cours Fauriel
42023 Saint-Etienne Cedex, France
olivier.boissier@emse.fr

**Abstract.** Coevolutionary Genetic Algorithms (CGAs) focus on the coevolution of competing or cooperating populations of individuals that represent specific parts of the global solution. In the current existing platforms, the use of such algorithms for evolutionary optimization suffers from the lack of expressiveness in the definition of the interaction structure between the populations. In order to overcome this problem and to allow an easy management of the interaction patterns of such algorithms, we propose a Distributed Agent Framework for Optimization (DAFO) in which the modelling of the interaction structure of CGAs is realized with a multiagent organisation modelling language. We show how this explicit modelling makes it possible to define multiple CGAs cooperation patterns.

## 1 Introduction

Coevolutionary Genetic Algorithms (CGAs) have been introduced as an extension of Genetic Algorithms (GAs) by focusing on the coevolution of competing or cooperating populations of individuals. In such an approach, instead of having one homogeneous population of individuals that represents a global solution, there are several populations that represent specific parts of the global solution.

Few of the existing libraries and frameworks that are proposed [1] allow the use of CGAs. Even if subpopulations are described as agents in the related literature, the model that is used in such frameworks is mainly based on an object-oriented approach. We consider that basing our approach on multi-agent system (MAS) technologies could help in making explicit the structure of the CGAs (e.g. the interactions and cooperation patterns between agents) and in adapting this structure according to the dynamics of the optimisation problem (i.e. the MAS environment) and the local computations realized in each GA .

In this paper we propose a Distributed Agent Framework for Optimization (DAFO) that consists in a multi-agent system specialized and dedicated to evolutionary optimization. A key element of this framework resides in its multiagent organisation model, extension of the Moise+ model [11]. We chose the Moise+ organisation model since it is well adapted to declare the coordination patterns that govern cooperation between different autonomous agents in terms of structure and collective plans. We extended it to specify the interaction patterns governing the information exchanges between the different algorithms of the CGAs. As we show, the use of such a model overcomes the lack of explicitness at the level of the algorithms of the structure and interactions. It has been used to build an hybrid variant [5] and a dynamic one [6] of a competitive CGA.

The paper is organized as follows. In section 2, we introduce first coevolutionary genetic algorithms. We present then the foundations and basic components of the DAFO framework in section 3, focusing on its organisation model and illustrating its usage for modelling a simple genetic algorithm (SGA). Section 4 demonstrates its use for modelling two existing CGAs and two novel ones (one hybrid and one dynamic). In section 5, we compare our approach to other existing agent frameworks for optimisation. Finally, section 6 presents our conclusions and future works.

## 2 Coevolutionary Genetic Algorithms

Genetic Algorithms (GAs) are heuristic methods based on Darwin's principle of evolution [7] (survival of the fittest) among candidate solutions (known as "individuals") with the stochastic processes of selection, recombination and mutation. Unfortunately, "classical" GAs tend to perform poorly or are difficult to apply on some problems especially when they have very large search spaces like many real-world problems such as inventory management [5] or topology control in mobile ad hoc networks [4] . In order to address these kinds of problems, researchers referred again to a nature inspired process so as to extend evolutionary algorithms: coevolution (i.e. the coexistence of several species [9]).

The main differences between Coevolutionary Genetic Algorithms (CGAs) [15] and GAs come from the adaptive nature of fitness evaluation in coevolutionary systems: the fitness of an individual is based on its interaction with other individuals from other so-called subpopulations. Thus, instead of evolving a population of similar individuals representing a global solution like in classical GAs, CGAs consider the coevolution of subpopulations of individuals representing specific parts of the global solution and evolving independently with a genetic algorithm. Individuals are evaluated based on their direct interactions with other individuals. These interactions can be either positive or negative depending on the consequences that such interaction produces on the population: (*i*) if positive (case of *cooperative coevolution*), the presence of each species stimulates the growth of the other species, (*ii*) if negative (case of *competitive coevolution*), the presence of each species implies the limitation of the growth of another species.

In the following we describe two CGAs, a cooperative one called Cooperative Coevolutionary GA (CCGA) and a competitive one based on non-cooperative models of game theory called Loosely Coupled GA (LCGA).

### 2.1 Cooperative Coevolutionary Genetic Algorithm

In [17], Potter and De Jong proposed a general framework for cooperative coevolution that they applied to test problems of function optimization (De Jong's test suite).

In this approach, multiple instances of GAs are run in parallel. Each population contains individuals that represent a component of a larger solution. Complete solutions are obtained by assembling representatives from each of the species (populations). The fitness of each individual depends on the quality of some/complete solutions the individual participated in. In some sense, it measures how well the individual cooperates to solve the problem. In the initial generation *(t=0)* individuals from a given subpopulation are matched with randomly chosen individuals from all other subpopulations. The best individual in each subpopulation is then retrieved based on its fitness value. The process of *cooperative coevolution* starts at the next generation *(t=1)*. For this purpose, in each generation the operations follow a round-robin schedule. Only one current subpopulation is active in one cycle, while the other subpopulations are frozen. All individuals from the active subpopulation are matched with the best values of the frozen subpopulations. When the evolutionary process is completed, the solution of the problem consists in the composition of the best individuals from each subpopulation. Cooperative coevolution showed to be efficient on different problems like static function optimization [16], rule learning [18] [20], neural network learning [19], and multiagent learning problems [21].

### 2.2 Loosely Coupled Genetic Algorithm

The Loosely Coupled Genetic Algorithm (LCGA) [25] is a medium-level parallel and distributed coevolutionary algorithm. It explores the paradigm of competitive coevolution algorithms motivated by non-cooperative models of game theory. For an optimization problem described by some function (a global criterion) of N variables, local chromosome structures are defined for each variable and local subpopulations are created for each of them. A problem to be solved is thus firstly analyzed in terms of possible decomposition and relations between subcomponents. They are expressed by a communication graph $G_{com}$, known as graph of interaction. The function decomposition and the definition of the interaction graph aim at minimizing communications while still ensuring that the fact of reaching local optima for all different players (being a Nash equilibrium point) still leads to a global optimum of the initial function. LCGA was applied to dynamic mapping and scheduling problems [23], a distributed scheduling problem [24] and to test functions [25]. However, the interaction graph is hard-coded and this process has still to be done manually by taking into account information on the internal structure of the cost function, i.e. of the problem.

In the CGAs literature, the term *Agent* is very redundant when referring to the different players (subpopulations) of these metaheuristics. However, none of the available platforms dedicated to evolutionary computation use the agent paradigm, but the object paradigm, which prevents them from providing high level and explicit models allowing an easy development of new generations of CGAs (e.g. hybrid or dynamic CGAs). Indeed, subpopulations composing a CGA can be modelled as autonomous agents. The topology and content of their interactions are driven by the cooperative or competitive nature of the CGA. Therefore modelling the interaction structure of CGAs with a multiagent organisation modelling language would allow bringing simplicity and self-expresiveness. It would consequently ease the definition and adaptation of the algorithms. We present our approach in the following section.

## 3 Multi-Agent Organisation Model for Optimisation

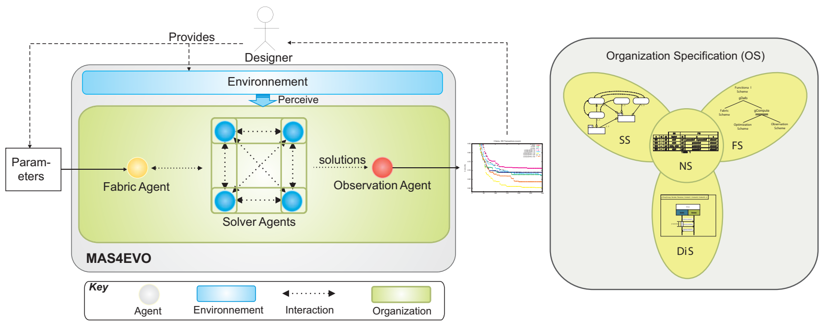


**Fig. 1.** MAS4EVO Overview with its organisation model

A general view of DAFO's underlying model is provided in Fig. 1. It is composed of three types of agents interacting together and able to perceive and react to the environment in which they are situated. The environment represents the optimisation problem which is provided and specified by the user. Given the considered problem, it can be either static or dynamic. The considered types of agents are: (*i*) *problem solving agents* in charge of optimizing a fitness function using a metaheuristic (i.e. executing the algorithms considered in GA or in CGAs), (*ii*) *fabric agents* which instantiate and configure the running application and (*iii*) *observation agents* which observe the problem solving agents and provide output interfaces to the end user(s).

The interactions between the agents use the FIPA ACL restricted to the *inform* and *agree* performatives. They also use FIPA-SL propositions to express the content of the messages. This content is either composed of organisational information or of computational information (i.e. individual(s)). Interaction protocols structure and express the sequences of messages according to the strategy of solution exchanges between the different GAs embedded in the agents (best, random, etc).

Finally, the coevolutionary strategies are represented by defining a multiagent organisation that is used to impose global patterns of cooperation on the behaviours of the agents. Thanks to its explicit representation, agents are able to change and reorganise the global functioning of the system. We focus here on this last model since it is the core of the DAFO Framework. The organisation model is based on $\mathcal{M}$OISE$^{+}$ [11] and dedicated to evolutionary optimization by the set of roles, groups and goals that are used. The abstract definition of cooperation patterns of the MAS builds what we call the Organisation Specification (OS). When a set of agents adopts an OS, they form an Organisation Entity (OE). The OS is defined using four different specifications: structural, functional, dialogic and normative.

The following sections provide a detailed description of those four specifications. The example of the usage of this model to represent a simple GA (SGA) will be used along these descriptions in order to illustrate the different concepts those specifications are based on.

## 3.1 Structural Specification

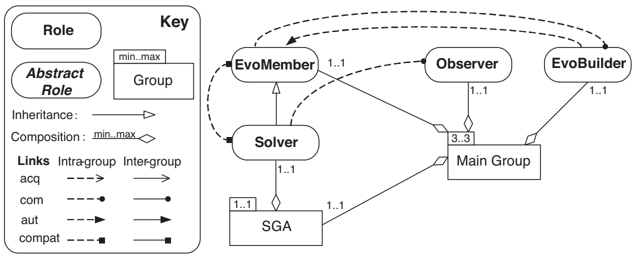


**Fig. 2.** Graphical Representation of the Structural Specification of the SGA Example

The Structural Specification (SS) expresses the structure of an organisation in terms of *roles*, *links* between roles and *groups*. A *role* is a label which allows attaching a set of constraints described in the other specifications of $\mathcal{M}$OISE$^{+}$. An agent is expected to respect these constraints as soon as it accepts to play that role. A *group* is defined by a composition of non-abstract roles (composition link), a set of intra-group links, a set of inter-group links, roles and groups cardinalities and agent cardinality (i.e. the number of agents which can play a role in the group). *Links* are the relations which have a direct influence on the agents' interactions. They can be of three types: (*acq*) for constraining the agent acquaintances graph, (*com*) for the interaction graph between the agents and (*aut*) for the authority and control structure bearing on the agents. A set of constraints is defined and used when the OS is enacted when the agents adopt roles in the created groups. These constraints express inter-roles compatibilities, links scope (intra-group and inter-group) and maximum/minimum number of adopted roles and created groups that could exist in the OE.

In Fig. 2 is shown the example of the SS of a simple GA. The root group is the *Main Group* group. It is composed of a single group *SGA* (cardinality "1..1"). The *Main Group* group is composed of three agents (cardinality "3..3"): one playing the *EvoBuilder* role, another one the *Observer* role and another one the *EvoMember* role. Each of these roles can only be adopted once (cardinality "1..1"). The *SGA* group is composed of one agent (cardinality "1..1") playing the role *Solver*. The *Solver* role inherits its constraints from the *EvoMember* role. The *EvoBuilder* role possesses an authority link on the *EvoMember* role since it will control the lifecycle of the *EvoMember* (e.g. instantiation). The *EvoMember* possesses a communication link with the *EvoBuilder* role. This link will be used to provide information concerning the *EvoMember* lifecycle (e.g. ready to compute). The *Solver* role also has a communication link but with the *Observer* role so as to send results of its computation process (e.g. the best individual per generation). Finally the *EvoMember* role and the *Solver* role have a compatibility link. This means that the same agent can play both roles in the same instance of the *Main Group* (we do not mention the *SGA* group since it is a subgroup of *Main Group*).

### 3.2 Functional Specification

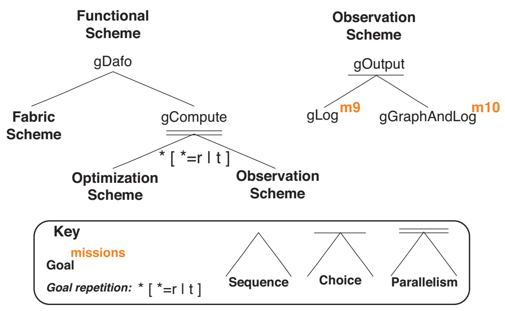


**Fig. 3.** Graphical Representation of the Functional Specification of the SGA Example

The Functional Specification (FS) describes a set of social schemes corresponding to the collective plans to be executed by the OE. These goals are grouped into missions which are attached to the roles in the normative specification. There missions are assigned to the agents as soon as they adopt the corresponding roles. Different types of goals and social schemes have been introduced. They can be functional ones as in $\mathcal{M}$OISE$^+$ but also interactional, organisational or regulatory. We have also added to the initial version of $\mathcal{M}$OISE$^+$ the possibility to associate a repetition constraint on a goal: guard condition that will stop the iterated achievement of the goal according to some repetitions number or time constraint. Interactional goals use one generic interaction protocol specified in the dialogic specification. It must mention the source and destination roles as well as the information exchanged if necessary. The organisational goals imply an action on the organisation entity (e.g. instantiation of new agents). Therefore they provide reorganisation capabilities. Regulatory goals imply a monitoring of the organisation so as to start a plan once a predefined criterion is reached. They can be used for instance to trigger a reorganisation process. Finally, artificial goals are goals that can not be reached. They are only used for specification purposes.

In Fig. 3 two out of the four social schemes of the FS of the SGA are presented. The functional scheme (left side) is considered as the main scheme of the FS. Its root goal, *gDafo*, is to run the DAFO framework. It is satisfied when the *Fabric Scheme* and the artificial goal *gCompute* are satisfied in sequence. The goal *gCompute* is itself achieved once its repetition condition is satisfied. An iteration of the *gCompute* goal will be achieved when goals *Optimization Scheme* and the *Observation Scheme* are achieved in parallel.

Concerning the *Observation Scheme* (right side of Fig. 3), its artificial root goal *gOutput* is satisfied if one of the two functional goals *gLog* or *gGraphAndLog* are achieved, i.e. an output of the results is obtained by the GA. The *gLog* goal saves the received best individual of one generation and the calculated average in each generation in log files. The *gGraphAndLog* additionally draws the corresponding graphs.

## 3.3 Dialogic Specification

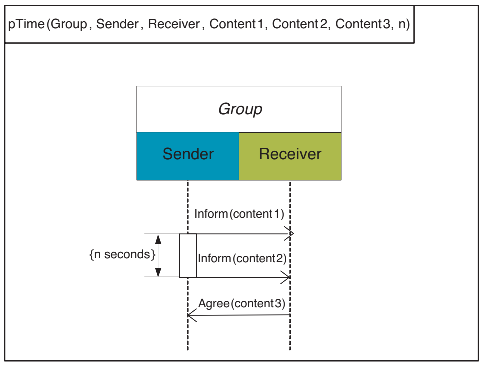


**Fig. 4.** Graphical Representation of the Dialogic Specification of the SGA Example

The Dialogic Specification (DiS) is an extension to the initial $\mathcal{M}$OISE$^+$ model. It is used to specify parameterizable generic interaction protocols in an independent manner of the roles and groups specified in the structural specification (SS). These interaction protocols are a variant of both AUML sequence diagrams [2] and AGR organisational sequence diagrams [10]. In Fig. 4, is presented a protocol defined using the DiS. This protocol requires seven parameters, a *group name*, a *sender role*, a *receiver role*, three different *message contents* and a time constraint. The *n* value represents a timing constraint, i.e. the number of seconds between the first *Inform* message containing the parameter *content1* sent from the sender to the receiver and the second one containing the parameter *content2*. Finally, the receiver has to send back an *Agree* message to the sender containing the parameter *content3*.

The graphical representation of the DiS has two dimensions: (*i*) the vertical dimension represents the time ordering and (*ii*) the horizontal dimension represents generic groups and roles that will be specified as parameters. Messages in sequence diagrams are ordered according to a time axis. This time axis is usually not rendered on diagrams but it goes according to the vertical dimension from top to bottom. Message ordering is expressed by the time axis.

### 3.4 Normative Specification

During the execution of the organisation, the agents will have: (*i*) to play roles within groups as specified in the structural specification (SS), (*ii*) to commit to missions and to achieve the corresponding goals defined in the functional specification (FS) and, (*iii*) finally to communicate following the interaction protocols depicted in the dialogic specification (DiS). In order to connect and glue these three specifications, we define the Normative Specification (NS) adapted from the one introduced in $\mathcal{M}\text{OISE}^{Inst}$ [3]. This specification is composed of a set of *normative expressions* involving a deontic operator (permission, obligation or interdiction) with terms from the SS, FS and DiS.

A norm $n$ is defined as:

$$n : \varphi \rightarrow op(bearer, m, p, s)$$

where $\varphi$ is the norm validity condition, *op* is the deontic operator defining an obligation, a permission or an interdiction, $bearer$ refers to an entity of the SS (role or group) on which the deontic operator is applied, $m$ refers to a mission defined in the FS, $p$ is an optional set of parameters used for the instantiation of the mission $m$, $s$ is the functional scheme of the FS to which the mission $m$ belongs.

For instance, the following norm:

$$N04 : true \rightarrow obl(EvoMemeber, m4, -, Fabric)$$

is an obligation for every agent playing the $EvoMember$ role to perform the mission $m4$ of the $Fabric$ social scheme. According to the mission $m4$ definition, the agent playing the $EvoMember$ role is endorsed of the obligation to achieve two organisational goals (creation of the subgroup SGA and adoption of the Solver role in this new group). Given the validity condition of this norm $N04$, the norm is valid.

## 4 Multi-Agent Organisation modelling of existing and new CGAs

In the preceding section we have described the different specifications provided by the organisation modelling language of DAFO. We have illustrated their use to model a simple GA. We show in this section how it is possible to model different CGAs by using a unique generic agent (i.e. the Problem Solving Agent) and a library of organisational models. This section presents the simple changes needed in the structural specification to switch from the CCGA to the LCGA (cf. Sec. 2). It additionally introduces the novel hybrid LCGA and the changes in the structural specification compared to the LCGA and the novel dynamic LCGA and the changes necessary in the functional specification to provide reorganisation capabilities to the LCGA.

### 4.1 CCGA MultiAgent Organisation Model

CCGA is a cooperative CGA using a complete graph as topology of communicaiton between the different subpopulations. It is therefore necessary to have at least three Problem Solving Agents (PSAs) to represent and compute these subpopulations. Since all PSAs interact with each other, we only use one group to manage them.

A specific feature of CCGA is the round-robin process realized in each generation of the algorithm. In this process only one subpopulation after the other is active at one time, receiving individual(s) from all the other subpopulations to evaluate its own individuals. This is modelled first at the structural level by adding two new roles, *Consumer* and *Producer*, and, second, at the functional level by introducing a regulatory goal. With this goal, each PSA can monitor if it is its turn to become active (i.e. playing role *Consumer*). organisational goals permit the PSAs to change role (i.e. switch from *Producer* to *Consumer* and the opposite).

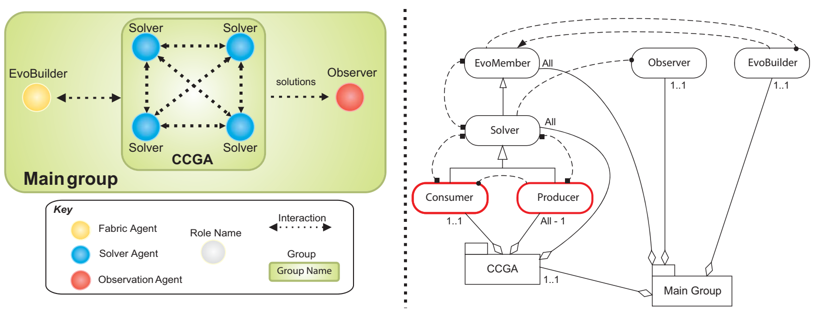


**Fig. 5.** Example of Organisation Entity and Graphical representation of the Structural Specification for the CCGA

In order to model the complete graph topology of the CCGA in terms of an organisational structure, new roles *Consumer* and *Producer* (compared to the SGA) inheriting from the *Solver* role have been added: (see highlighted roles on the right side of Fig. 5). The *Consumer* role can only be played by a single PSA (cardinality 1..1). It represents the active subpopulation while the *Producer* role is played by all the other PSAs participating to the system (cardinality All-1). Agents playing the *Producer* role are able to send some information (generally individuals) to the agent playing the *Consumer* role. Let us notice that an agent playing the Solver role can also play the *Consumer* or the *Producer* role (compatibility link).

### 4.2 LCGA Model

LCGA is a competitive CGA. It has no restriction concerning the topology of communication between the different subpopulations. Due to the optimization problems features we tackled, i.e. Inventory Management [5] and Injection Networks [4], we modelled a LCGAs using ring or complete graph topologies. We therefore chose to use topologies based on a ring, in which it is possible to augment the number of neighbours, from 1 to n-1 (n being the number of PSAs) as illustrated in the Fig. 6.

As for the CCGA, it is necessary to have at least three PSAs to represent the subpopulations. Since in these possible topologies one PSA will communicate with one or several neighbours, a new type of group is introduced: *Solving Unit*. This group contains one *Producer* role (cardinility 1..1) and one to several *Consumer* roles (cardinality 1..N). In order to create a ring (see left side of Fig. 6), the same PSA will play the *Consumer* role in one *Solving Unit* group and the *Producer* role in the next *Solving Unit* group. This Solving Unit group is a subgroup of LCGA. These two new groups are highlighted on the right side of Fig. 6. An inter-group compatibility link between the *Producer* and *Consumer* roles has also been added so as to express the possibility for an agent to play both roles in two different *Solving Unit* groups.

Contrary to CCGA, in LCGA there is no synchronous exchange of individuals between the subpopulations and thus no monitoring of the organisation and no modification of the OE (i.e. no swapping from Producer to Consumer roles and vice versa). The individuals exchanged between subpopulations are also different since in LCGA random individuals are sent contrary to the best or best and random of the CCGA. This is taken into consideration with a new parameter in the pInform interaction protocol (i.e. random).

Another difference is the collaborative process introduced in LCGA. Once one subpopulation has calculated the fitness values of its individuals, based on the individuals of its neighbour, these fitness values are sent to the same neighbour population which will use these fitness values to re-evaluate its own individuals (typically the average between the fitness it calculated and the fitness it received). This will be tackled in the functional specification as a new functional goal (gReEvaluate) and new possible parameters in the pInform interaction protocol (i.e. fitnesses).

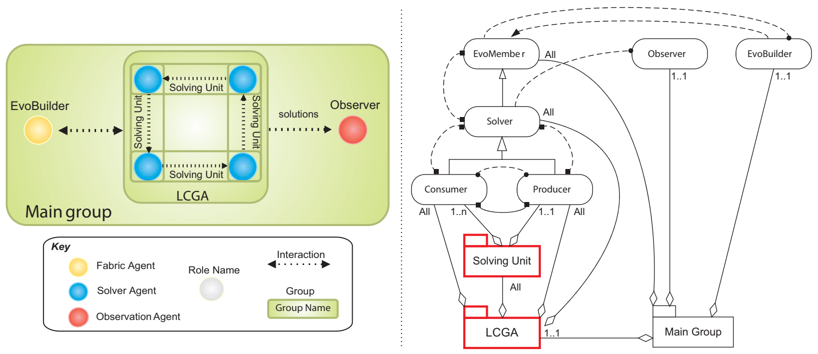


**Fig. 6.** Example of Organisation Entity and Graphical representation of the Structural Specification for the LCGA

## 4.3 hLCGA Model

Hybridization of GAs has been a very active research area. However, hybrid coevolutionary genetic algorithms have been rarely tackled, except in [26] where a CCGA has been hybridized with a hill-climbing algorithm. Due to the few existing researches in this area and taking profit from our multiagent organisation approach, we decided to investigate the hybridization of the LCGA with different local search algorithms.

Although it is possible to make a GA hybrid in different ways, latest published articles [26] have shown that combining GAs with local search algorithms are one of the best approaches for improving the results. The model of this hybrid LCGA is therefore based on the LCGA's presented in the previous section. To hybridize this LCGA, each PSA running a SGA and representing one subpopulations of the LCGA will communicate with another new PSA running one local search algorithm (LS). This topology is represented on the left side of Fig. 7.

As for the LCGA, it is necessary to have at least three PSAs to represent the subpopulations. In our hybrid approach, we therefore have to add three PSAs to run the local search algorithm. In order to allow a communication between these PSAs, a new type of group has been added in the SS, *LocalSearchUnit*. In this group, the PSA running a SGA will play the *Solver* role and the PSA running the LS algorithm will play a new *LocalSearcher* role (see right side of Fig. 7). These new *LocalSearchUnit* groups will have to be created and these new *LocalSearcher* roles will have to be adopted by the additional PSAs. This will be considered in the FS with the addition of the organisational goals CreateSubGroup(LocalSearch, DAFO) and the gAdoptRole(LocalSearcher, SolvingUnit).

Once the OE is instantiated, it is necessary to run the LS algorithm. This is also achieved in the FS by adding the exchange of individual(s) (best individual or population rate defined by the parameter alpha) from the *Solver* role to the *LocalSearcher* with the interactional goal pInform(alpha). Then the LS algorithm is run with the functional goal *gRunLS* and finally the optimized individual(s) are sent back using the same interactional goal *pInform(alpha)*. These new goals will be part of new missions which will be attached to roles in new norms of the NS.

The SS of the hLCGA presented in Figure 7 is thus similar to the LCGA's SS plus the addition of the new group *LocalSearchUnit* which is a subgroup of the *DAFO* group and of the new role *LocalSearcher*. All the instances of the *LocalSearchUnit* subgroup are contained in the *DAFO* group (cardinality "All"). The *LocalSearch Unit* group contains exactly one *Solver* role and one *LocalSearcher* role (cardinality "1..1"). They both inherit from the *EvoMember* role. Those two roles have to be played by two different agents due to the cardinality "2" on the group. The *LocalSearcher* role and the *EvoMember* role have a new compatibility link which means that the same agent can play both roles in the same instance of the *DAFO* group (we do not mention the *LCGA* group since it is a subgroup of the *DAFO* group). Finally, a new intra-group communication link between the *LocalSearcher* and the *Solver* roles and vice versa is added. This will allow the exchange of individual(s) necessary for the LS algorithm.

## 4.4 dLCGA Model

Few previous researches have considered the adaptive CGAs, also known as dynamic CGAs. The related works focused either on the adaptation of the number of populations [19] or on the adaptation of the parameters [12]. Our contribution consists in

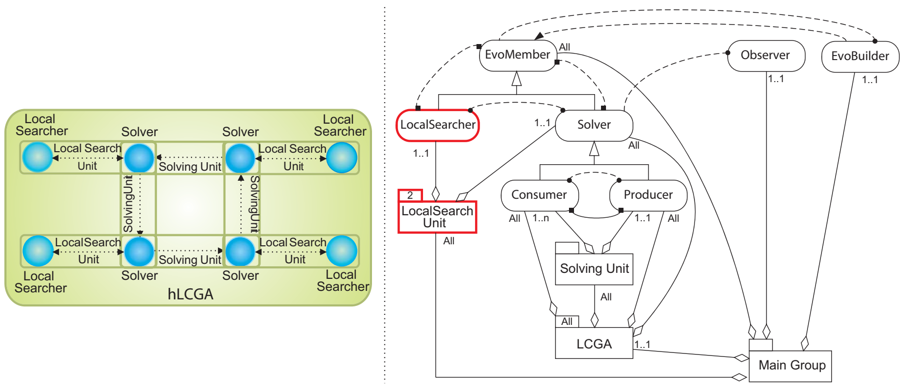


**Fig. 7.** Example of Organisation Entity and graphical representation of the Structural Specification for the hLCGA

building a dynamic LCGA, in which the topology of communication between the populations evolves during runtime. Indeed, contrary to CCGA where the topology is fixed (i.e. fully connected graph), using LCGA makes no restriction on the communication graph, since it fully depends on the decomposition of the optimized problem.

dLCGA is a new dynamic version of LCGA. Its interaction structure is modified each *n* generations of the algorithm. The modification is achieved through a cooperative process starting with the first player who randomly chooses a new position in the graph of interaction and informs all the other players of his local decision. The next player in the graph will then randomly choose a new position among the remaining available ones and inform the other players. This process is iteratively executed by all players. Once finished, each player goes to its new position and the algorithm runs again for n generations. Through this random process, each population exchanges information with different populations during runtime, and thus has to evaluate its individuals using different parts of the solution.

Left side of Fig. 8 shows an example of a dLCGA with a ring topology using a simplified view of an organisational entity (OE). After *n* generations of the algorithm, all *PSAs* leave the roles they play in the *Solving Unit* groups. Consequently, they only play a role in the "base" group *Main Group* and this way they communicate all together in order to define the groups in which each of them will play a role. Once they all know their new location, they take their roles in the newly defined groups. Since the topology changes consist in moving the PSAs on the ring, this only affects the organisational entity (OE) and not the organisational specification (OS). Therefore, the structural specification of the dLCGA is similar to the LCGA's.

We therefore only describe the changes needed in the FS. The functional scheme is presented on the right side of Fig. 8. A single social scheme has been added to the FS of the LCGA in order to manage the three steps of the reorganisation describe hereinbefore (monitoring, negotiation and reorganisation). The following details the content of this new *reorganisation scheme*. Each PSA will have to monitor its computation in order to verify if the condition is met or not. This is achieved by a new regulatory goal *gMonitoring*. Once this condition is met, each PSA leaves its *Consumer* and *Producer* roles in its *SolvingUnit* groups. This is achieved with two new organisational goals *gLeaveRole(Producer, SolvingUnit)* and *gLeaveRole(Consumer, SolvingUnit)*. Then the PSAs will have to negotiate with each other to find their new groups. This negotiation is achieved with a new interactional goal *pNegotiate*. This goal implies the definition of a new interaction protocol in the dialogic specification. Finally the PSAs will adopt their *Consumer* and *Producer* roles in their newly defined *SolvingUnit* groups using the *gAdoptRole(Producer, SolvingUnit)* and *gAdoptRole(Consumer, SolvingUnit)* organisational goals. All these new goals are grouped in the *Reorganisation Scheme*.

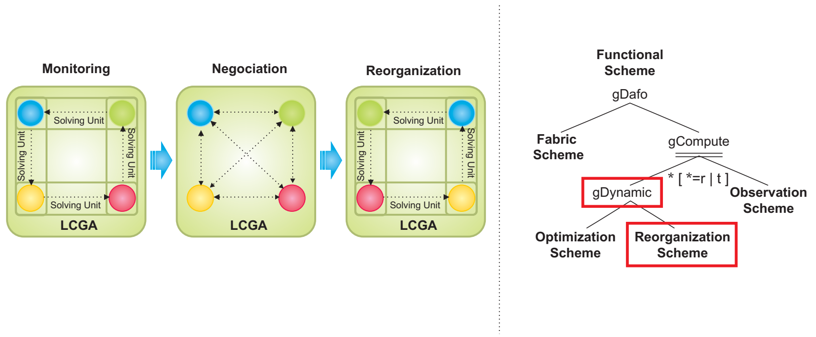


**Fig. 8.** Example of organisation entity evolution and graphical representation of the Functional Specification for the dLCGA

This section has demonstrated the usage of our approach to model two existing CGAs, the CCGA and the LCGA. Through these first two examples it was shown that with few changes in the organisation specifications it is possible to switch from one CGA to the other. We also introduced two new competitive coevolutionary genetic algorithms, hLCGA and dLCGA, respectively hybrid and dynamic variants of the LCGA. Their detailed model using MAS4EVO has been presented, demonstrating the capacity of this new organisational model to describe such new coevolutionary genetic algorithms. Indeed, for the hLCGA it was proved that by adding a new group and a new role in the SS and a few new goals in the FS to the LCGAs (and adapting the NS accordingly) it is possible to create a new hybrid variant. Similarly, by keeping the same SS as the LCGA and adding a reorganisation scheme in its FS it is possible to create a new dynamic variant of LCGA.

## 5 Related Work

In order to facilitate the use and comparison of EAs, many different libraries and frameworks have been proposed in the literature. Only a few of these platforms allows the use of some CGAs, but none of them implements “standard” CGAs like the CCGA [17] or the LCGA introduced in section 2. The vast majority is based on the object oriented paradigm. They consequently can neither provide the expressiveness of high-level organisational specifications of the algorithms, nor the autonomy of computational agents and the distribution inherent to multiagent systems. According to the literature, only three frameworks use a multi-agent architecture for metaheuristics algorithms.

MAGMA (MultiAGgent Architecture for Metaheuristics), introduced in Roli’s PhD thesis [22] and later in [13], consists in a multi-level architecture where each level contains one or several specialized agents implementing an algorithm. This architecture provides a high modularity but does not use any organisational model which would for instance allow to specify in an explicit way the interactions. In MAS-DGA (Multi-Agent System for Distributed Genetic Algorithms) [14], each basic GA is encapsulated into an agent that must keeps knowledge of the search, learning, or optimization problem it operates on. Agents are coordinated by a set of rules stipulating the topological and communication (migration) aspects These rules can be fixed a priori or set during run-time. MAS-DGA therefore includes a simple organisational and reorganisational model based on a structural specification. However, no detail concerning this model or its implementation are given in the single paper mentioning MAS-DGA. Finally, AMF (Agent Metaheuristic Framework) presented in [8] proposes a framework based on an organisational model which describes a metaheuristic in terms of roles. These roles correspond to the main components or tasks in a metaheuristic: intensification, diversification, memory and adaptation or self-adaptation. AMF is the only framework based on a specific organisational model, however it is limited to its structural specification (i.e. roles and interactions between roles).

To conclude, only three EAs platforms use the agent paradigm, among which only AMF introduced a first approach of using an organisational MAS for representing metaheuristics, taking into consideration their flexibility, robustness and modularity. However its organisational model is limited to a structural specification (definition of roles and interactions between roles). Only little information is available concerning the implementation and the configuration/usage of these platforms. In addition, none of them allow the use of CGAs. Finally there is no available version of these implementations.

## 6 Conclusion and Future Works

We have proposed in this paper a multi-agent approach dedicated to evolutionary optimization. The resulting DAFO (Distributed Agent Framework for Optimization) framework provides a novel way of modelling CGAs as a multi-agent organisation using the MAS4EVO model where agents correspond to CGAs and organisation explicitly defines their cooperation structure. We have shown how it is possible to define multiple CGAs, existing ones but also novel ones, as different organisation specifications. Thanks to it, the understanding and the manipulation of CGAs structures are facilitated.

In our future work we plan to develop a more adaptive dLCGA by increasing the autonomy of the Problem Solving Agents, allowing them to adapt their parameters and/or their organisation according to the problem and/or to their neighbours in the organisation.

## References

1. Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 6(5):443–462, 2002.
2. B. Bauer, J. Muller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. *International Journal on Software Engineering and Knowledge Engineering*, 11(3):1–24, 2001.
3. Olivier Boissier and Benjamin Gâteau. Normative multi-agent organizations: Modeling, support and control, draft version. In Guido Boella, Leon van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems*, number 07122 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. <http://drops.dagstuhl.de/opus/volltexte/2007/902> [date of citation: 2007-01-01].
4. Grégoire Danoy, Pascal Bouvry, and Luc Hogie. Coevolutionary genetic algorithms for ad hoc injection networks design optimization. In *IEEE Congress on Evolutionary Computation*, pages 4273–4280. IEEE, 2007.
5. Grégoire Danoy, Pascal Bouvry, and Tomy Martins. hlcga: A hybrid competitive coevolutionary genetic algorithm. In *HIS*, page 48. IEEE Computer Society, 2006.
6. Grégoire Danoy, Pascal Bouvry, and Franciszek Seredynski. Evaluation of strategies for co-evolutionary genetic algorithms: Dlcga case study. In *Proceedings of the 16th international conference on Artificial Neural Networks In Engineering - ANNIE*, Saint-Louis, USA, 2006. ASME Press.
7. C. Darwin. *The Origin of Species by Means of Natural Selection*. Mentor Reprint, 1958, NY, 1859.
8. Jean-Charles Créput David Meignan and Abderrafiaa Koukam. An organizational view of metaheuristics. In *AAMAS'08: Proceedings of First International Workshop on Optimisation in Multi-Agent Systems,*, pages 77–85, 2008.
9. Paul R. Ehrlich and Peter H. Raven. Butterflies and plants: A study in coevolution. *Evolution*, 18(4):586–608, 1964.
10. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *AOSE*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 2003.
11. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. pages 118–128.
12. Antony Iorio and Xiaodong Li. Parameter control within a co-operative co-evolutionary genetic algorithm. In *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 247–256, London, UK, 2002. Springer-Verlag.
13. M. Milano and A. Roli. Magma: A multiagent architecture for metaheuristics. *IEEE Trans. on Systems, Man and Cybernetics – Part B*, 34(2):925–941, April 2004.
14. E. Noda, A. L. V. Coelho, I. L. M. Ricarte, A. Yamakami, and A. A. Freitas. Devising adaptive migration policies for cooperative distributed genetic algorithms. In *Proc. 2002 IEEE Int. Conf. on Systems, Man and Cybernetics*. IEEE Press, 2002.
15. Jan Paredis. Coevolutionary life-time learning. In *Parallel Problem Solving from Nature – PPSN IV*, pages 72–80, Berlin, 1996. Springer.
16. Mitchell A. Potter. *The design and analysis of a computational model of cooperative coevolution*. PhD thesis, 1997.
17. Mitchell A. Potter and Kenneth De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature – PPSN III*, pages 249–257, Berlin, 1994. Springer.

18. Mitchell A. Potter and Kenneth A. De Jong. The coevolution of antibodies for concept learning. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 530–539, London, UK, 1998. Springer-Verlag.
19. Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.
20. Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 366–372, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.
21. Mitchell A. Potter, Lisa Meeden, and Alan C. Schultz. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *IJCAI*, pages 1337–1343, 2001.
22. A. Roli. Metaheuristics and structure in satisfiability problems. Technical Report DEIS-LIA-03-005, University of Bologna (Italy), May 2003. PhD Thesis - LIA Series no. 66.
23. Franciszek Seredynski. Competitive coevolutionary multi-agent systems: the application to mapping and scheduling problems. *J. Parallel Distrib. Comput.*, 47(1):39–57, 1997.
24. Franciszek Seredynski, Jacek Koronacki, and Cezary Z. Janikow. Distributed scheduling with decomposed optimization criterion: Genetic programming approach. In *Proceedings of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, pages 192–200, London, UK, 1999. Springer-Verlag.
25. Franciszek Seredynski, Albert Y. Zomaya, and Pascal Bouvry. Function optimization with coevolutionary algorithms. In *Proc. of the International Intelligent Information Processing and Web Mining Conference*, Poland, 2003. Springer.
26. You Seok Son and Ross Baldick. Hybrid coevolutionary programming for nash equilibrium search in games with local optima. *IEEE Trans. Evolutionary Computation*, 8(4):305–315, 2004.

# A Tool for Creation and Deployment of Organization Models

Endri Deliu[1] and Michael Köhler-Bußmeier[2]

[1] Otto Group, Wandsbeker Str. 3-7, 22172 Hamburg, Germany
`endri.deliu@ottogroup.com`
[2] University of Hamburg, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
`koehler@informatik.uni-hamburg.de`

**Abstract.** The need for handling the increasing complexity in software systems has allowed the introduction and establishment of an organizational paradigm as an alternative in software modeling and development. Especially within the multi-agent systems community, organizational concepts are enjoying increasing popularity for efficiently structuring multi-agent behavior. Organizational specifications and their implementation as multi-agent systems lack however a streamlined transition between each other. In this paper we address this problem by introducing a software tool capable of creating and editing organization models as well as deploying such models as multi-agent systems. The tool is built on SONAR [1], a formal organizational specification based on Petri nets. By unifying in one tool the organizational specification and deployment process quick reaction cycles to incremental changes of system design become possible.

## 1 Introduction

Important influxes from sociology and organization theory have begun delineating what may dissolve the trade-off between agent autonomy and multi-agent system reliability and predictability. Between the system and the agents composing it, other levels of control have been introduced which are mainly derived from sociological concepts. The concept of organization is used as an umbrella term for groups of agents and their dependencies, interaction channels or relationships. As a result, an organizational perspective on multi-agent systems has gradually emerged which focuses on organizational concepts such as groups, communities, organizations, etc., in contrast to the former focus of multi-agent systems on the agent's state and its relationship to the agent's behavior [2].

Modeling agent organizations requires a modeling language that is able to express most (possibly all) of the notions that the concept of organization encompasses in an intuitive and easily understandable way. Petri nets are well suited for use in modeling systems and simultaneously offer a complete formal frame. In this context, a framework for the development of concurrent and distributed software systems has been built as a multi-agent system basing on reference nets[3]

[3] Reference nets are high level Petri nets.

[3]. *Mulan* (**M**ulti **A**gent **N**ets) [4] provides the framework's reference architecture used for the the multi-agent system. *Mulan* is built on Java and reference nets and can be executed in *Renew* [5], a Petri net editor and simulator.

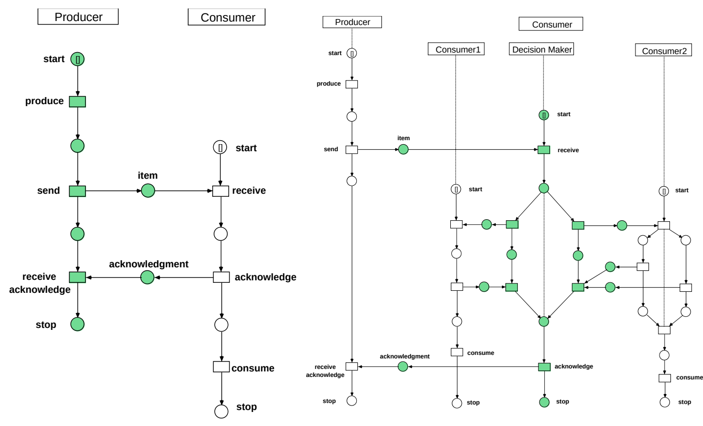
In this work, OREDI, a Petri net based tool will be presented. It enables editing organization models as well as deploying such models as multi-agent systems. OREDI is built on top of Renew and relies on SONAR, a formal organizational specification based on Petri nets. Section 2 will shortly introduce the main concepts of SONAR which are used in OREDI. In Sect. 3, the main workflow of creating organization models with OREDI is described. In Sect. 4 and Sect. 5, the deployment of SONAR organization models into agent organizations with OREDI is presented.

## 2 SONAR, a Formal Model of Organizations

SONAR [1] is a Petri net model which is used to define *formal organizations.* A formal organization in SONAR is the name for the combination of organization structure and organizational services in a multi-agent system [1]. Members of the multi-agent systems are not included in a formal organization. Organizational services are represented in SONAR through *service nets* [1] while the organization structure is represented by formal concepts such as organization nets and R/D nets. Each one of these formal concepts will be shortly presented in this section.

### 2.1 Service Nets

Service nets are Petri net models that specify how agents interact with each other. Nevertheless, service nets abstract from agents and describe interaction between roles. Roles describe the agents behavior in a multi-agent system. All agents that acquire a specific role have to comply to the role's constraints and fulfill the role's requirements. Defining interaction protocols between roles makes such protocols independent of the agents that are assigned to the role. In the example service net given in [1] (Fig. 1a) the roles *producer* and *consumer* interact with each other. In [1], $\mathcal{R} := 2^{Rol}$ is defined as the role universe where *Rol* is a set of roles. Every $R \in \mathcal{R}$ is called a role profile. The set operation $\subseteq$ defines for $\mathcal{R}$ a partial ordering. If $R_1, R_2 \in \mathcal{R}$ and $R_1 \subseteq R_2$ then $R_1$ is said to be more specialized than $R_2$. Besides, roles in a service net are assumed to be different from roles in other service nets. Formally, service nets are defined as a tuple $D = (N, r)$ where $N = (P, T, F)$ is a Petri net and $r$ is a function $r : T \to Rol$. The function $r$ assigns a role $r(t) \in Rol$ to every transition $t$ of the net $D$. This means that the task $t$ is executed only by the agents that implement the role $r(t)$. A role function for the whole service net is defined as $R(D) := r(T)$. For a service net $D$ and a role profile $R \subseteq R(D)$, $D$ can be restricted to $D[R]$ which is a subnet of $D$ defined as a tuple $D[R] = (P_R, T_R, F_R)$. The subnet is determined by the nodes $T_R := r^{-1}(R)$ and $P_R := ({}^\bullet T_R \cup T_R^\bullet)$ and the arcs linking these nodes. In Figure 1a, the subnet $PC[producer]$ is shown by the filled nodes.



(a) Simple service net, PC. (b) Refined service net, PC2.

Fig. 1: Simple and refined service nets of producer-consumer ([1]).

*Refinement of Service Nets.* For the service nets $D_1, D_2$ and the role profiles $R_1 \subseteq R(D_1)$ and $R_2 \subseteq R(D_2)$ the relation

$$\langle\!\langle D_1; R_1, D_2; R_2 \rangle\!\rangle$$

is defined if $D_1[R_1]$ and $D_2[R_2]$ can replace each other in $D_1$ or $D_2$ without changing the behavior. That means that $D_1[R_1]$ and $D_2[R_2]$ are the same in terms of the input and output. In [1], an example net named $PC_2$ is given where the role *consumer* is refined (Fig. 1b) when compared to the *consumer* of the net $PC$ (Fig. 1a).

## 2.2 R/D Nets and Organization Nets

R/D nets are Petri nets $(P, T, F)$ in which the role profiles are modeled by places and the tasks by transitions. The preset of a transition should contain exactly one place ($|{}^\bullet t| = 1$). A transition $t \in T$ is said to be *executive* if $t^\bullet = \emptyset$ and *delegative* if $t^\bullet \neq \emptyset$. Every place is labeled by a role profile and the transitions are labeled by service nets. Formally, a R/D net is a tuple $(R, N, D)$ where:

- $N = (P, T, F)$ is a Petri net with $|{}^\bullet t| = 1$ for all $t \in T$ and $p^\bullet \neq \emptyset$ for all $p \in P$.
- $R : P \to \mathcal{R} \backslash \{\emptyset\}$ where $\forall t \in T \ : \forall p, p' \in t^\bullet \ : \ p \neq p' \Rightarrow R(p) \cap R(p') = \emptyset$.
- $D : T \to \mathcal{D}$.

Well-formed R/D nets are defined as R/D nets with additional formal properties which allow some kind of type checking. In well-formed R/D nets, the service net labeled to a transition as well as the role profiles labeled to places are consistently related to the structure of the R/D net. The formal definition of well-formedness can be found in [1].

Organization nets [1] represent the organizational structure. Central to this formalism is the notion of *organizational positions* which represent positions in real organizations. Organization nets involve *organizational positions*. Organizational positions are responsible for several *tasks* and can also delegate tasks to other organizational positions. Formally, an *organization net* is a tuple $(N, \mathcal{O})$ where $N$ is a Petri net $N = (P, T, F)$ and $\mathcal{O}$ is a partition on the set $P \cup T$ where for all $O \in \mathcal{O}$ the following is true:

$$\forall p \in O \cap P : {}^{\bullet}p \subseteq O \wedge p^{\bullet} \subseteq \overline{O} \quad (\text{with } \overline{O} = (P \cup T) \setminus O)$$
$$\text{and } \forall t \in O \cap T : {}^{\bullet}t \subseteq \overline{O} \wedge t^{\bullet} \subseteq O \ .$$

The elements $O \in \mathcal{O}$ describe the *organizational positions*. Organization nets combined with R/D nets form *formal organizations*. Formal organizations are defined as tuples $Org = (N, \mathcal{O}, R, D)$ where $(N, R, D)$ is a R/D net and $(N, \mathcal{O})$ is an organization net. An example of a formal organization is shown in Fig. 2. The gray boxes represent the organizational positions.

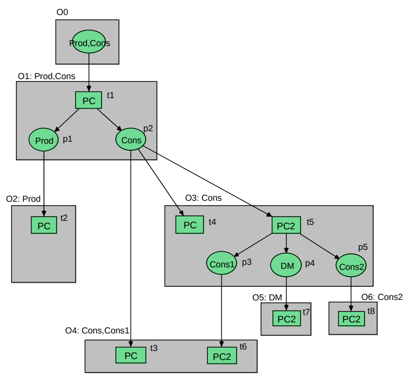
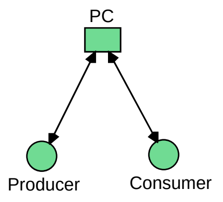


Fig. 2: The organization net of producer-consumer ([1]).

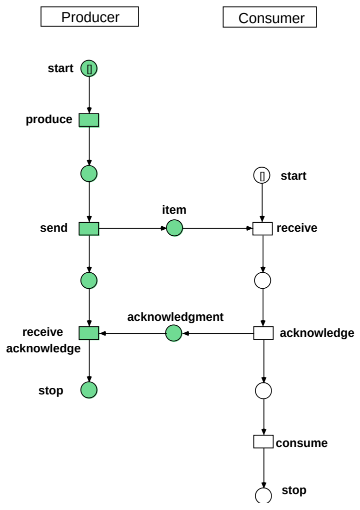
## 3 Creating Organization Models with OREDI

OREDI contains an editing tool for SONAR formal organizations. It is built as a set of Renew plugins. OREDI users can create SONAR formal organizations without being directly aware of their underlying rules. The process of creating formal organizations involves two steps. The first step being the creation of service definition nets and of refinement relationship nets. Service definitions nets represent service nets while refinement definition nets represent refinement relationships between role profiles as described in the previous subsections. The second step being the creation of organization nets and the assignment of service nets and role profiles to respectively transitions and places. The completion of both steps leads users to formal organizations. Each step is handled in separate editors. Thus, an editor for modeling service and refinement definition nets is used first. Then the results of the first step are loaded in a second editor where the organization and R/D nets are modeled.

Service definition nets can specify one or more service definitions. Service definitions displayed in service definition nets are basically an abstraction over service nets. Hence, service definition nets can provide an overview of many service nets. Service definition nets specify service definitions which contain services and the roles involved in these services. The service is represented by a transition and the roles involved in the service are represented by places connected to the transition. The transition of a service definition corresponds to a service net. The places of a service definition correspond to the roles involved in the service net. In Fig. 3a and Fig. 3b, a service definition net with one service definition and the corresponding service net producer-consumer (PC) are shown.



(a) Service definition net with one service.



(b) Service net, PC.

Fig. 3: Service definition net and service net.

Refinement definition nets model the refinement relationships between sets of roles involved in service nets. Refinement relationships are used in well-formed R/D nets. They describe the equality of input and output behavior of sets of roles involved in different service nets. Formally, a refinement definition net over the refinement $\langle\!\langle D_1; R_1, R_2; D_2 \rangle\!\rangle$ is a tuple $(N, r)$ where N is a Petri net $N = ({}^\bullet t \cup t^\bullet, \{t\}, F)$ and additionally $r$ is a function which assigns places to role inscriptions where $\forall p \in {}^\bullet t \cup t^\bullet$ it is true that $r(p) \in R_1 \cup R_2$ and $r({}^\bullet t) = R_1, r(t^\bullet) = R_2$. In Fig. 4, a refinement net is displayed showing the refinement relationship between the sets of roles:

$$\langle\!\langle PC; \{Cons\}, \{Cons1, DM, Cons2\}; PC_2 \rangle\!\rangle$$

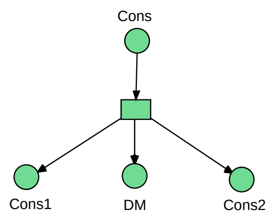


Fig. 4: Refinement definition net.

After completing the creation of service and refinement definition nets OREDI users can start modeling SONAR formal organizations based on the service nets and the refinement relationships previously modeled. The formal rules during the creation of organization nets are enforced through an interaction policy of allowing only manipulations such as deleting, moving or creating elements that do not violate the SONAR rules for organization nets. During modeling of (well-formed) R/D nets the correct assignment of service nets and roles is enforced by OREDI through serving context based assignment suggestions the selection of which can only lead to formal organizations. Assignment suggestions are served in a top-down fashion where the preset of a net element has to be already assigned for OREDI to make suggestions for the element itself. In Fig. 5 a list of assignment suggestion for a transition is shown. The inscriptions at the top of Fig. 5 are imported service and refinement definition nets which are used for the generation of the assignment suggestions.

At the end of the modeling process, OREDI supports the deployment of the user created formal organizations as agent organizations consisting of Organization Position Agents (OPA) and Organization Member Agents (OMA).

## 4 Agent Organizations as OPA/OMA Networks

Deploying SONAR formal organizations as multi-agent organizations requires the selection of a multi-agent organization design of choice. The design used by OREDI is specified in [6] where a decoupling of the parts of the multi-agent

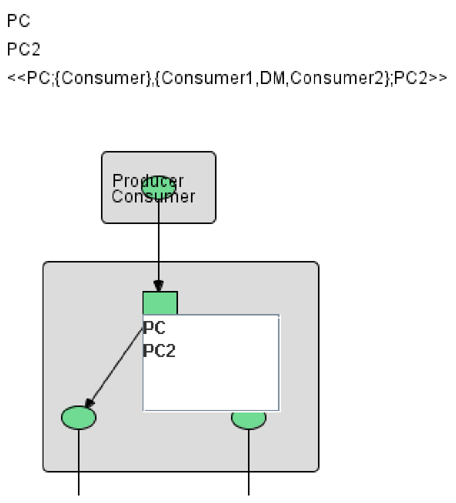


Fig. 5: Suggestion list for a transition.

system that specify the organizational structure from those that act as members of the organization is proposed. Such a design is consistent with the implicit detachment of organization structure and implementation in SONAR. In [6] there is a separation between Organization Position Agent (OPA) and Organization Member Agent (OMA). OPA-s derive from the structure of the organization and are owned by the organization. OMA-s are external agents that actually do the work. They carry out actions and make decisions. However, they must use the OPA-s as a gateway to the organization. OMA-s take part in the organizational processes only through their OPA-s. Fig. 6 displays an example.

OMA-s can be any kind of agents, including agents from different multi-agent platforms. The decisive point is that an interface to the OPA-s is offered. OMA-s can also occupy multiple positions. In the example in Fig. 6 the same agent occupies the “coordinator” and “firm A”. OPA-s are connected through *formal channels* which correspond to the delegation relationships of SONAR organization nets. Also, informal channels may emerge between OMA-s due to interactions that were not foreseen by the formal specification.

## 5 Deploying SONAR Formal Organizations as OPA/OMA Networks

OREDI supports the deployment of SONAR formal organizations as OPA/OMA networks. After modeling a formal organization net it is exported in an XML format. The XML file generated from the formal organization net is parsed and the deployment process begins. The generation of the XML and its subsequent

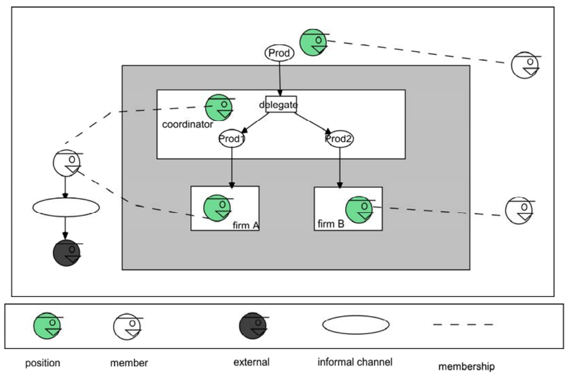


Fig. 6: A multi-agent design derived from an organization net ([6]).

parsing was conceived to provide a platform and application independent solution to deploying SONAR formal organizations. As such, the generated XML file contains all relevant formal information of the models and can be used for deployment in any multi-agent platform. We opted for an implementation of the deployment steps in *Mulan* as it supports building multi-agent systems with reference nets. This allowed using Petri nets both as a modeling as well as a programming technology thus easing and streamlining the gap between modeling and development.

Deploying a SONAR formal organization as OPA/OMA networks involves the generation of OPA-s and the assignment of OMA-s to OPA-s. After these two phases team processes such as team formation, team plan formation, and team plan execution can follow as specified in [6] and [7]. In this work, only the generation of OPA-s and the assignment of OMA-s is handled.

Organizational positions of the formal organization net are deployed as OPA-s. The generated OPA-s know the identity of their neighbor OPA-s and communicate with them through a set of encrypted messages. The assignment of OMA-s to OPA-s is made in a market based fashion with OPA-s making open position announcements and interested OMA-s competing for the employment for the open positions. The communication between OPA-s and OMA-s is also encrypted with a public key mechanism.

## 5.1 Deployment of OPA-s in Mulan

OREDI deploys formal organization as *Mulan* OPA-s. *Mulan* is a FIPA [8] compliant architecture. At first, a *Mulan* platform is generated where one or more agent organizations can be embedded. The position agents generated for each position in the SONAR organization net are placed inside the created platform. Additionally, suitable *Mulan* protocols handle agent conversations. *Mulan* agents can use protocols proactively or reactively as a response to specific messages.

The decision which protocol to use for a specific received message is made in the knowledge base where a mapping between message templates and protocols is consulted.

OPA-s have the same information of their corresponding positions in the SONAR organization net specified in the XML file. This information includes the position's relative place in the organization (knowledge about neighbor positions), the roles they are implementing/delegating and their tasks. Generating OPA-s out of an organization net specification is accomplished in agent-oriented fashion by an initial agent. The initial agent is responsible for the generation of the OPA-s and their initialization with informations extracted from the formal organization net specification. The initial agent is called the *organization agent* as it has a global view on all positions.

The information needed from an OPA includes which other OPA-s are its neighbors. This requires the identity of the neighbors. At least in *Mulan*, the identity of agents and their location can only be known after the creation of these agents. This means that information about the neighbor positions has to be provided for a OPA only after, not during its creation. Thus, the information about the place of a position agent in an organization is conveyed through a conversation with the organization agent. During the conversation, in order to make sure that the messages come from the right parties they are signed with a public key mechanism which requires that parties know their respective public keys. In Fig. 7, an AUML sequence diagram displays the conversation between OPA-s and the organization agent during which the organization agent communicates to the positions all relevant informations extracted from the organization net specification. In FIPA terminology, conversations between agents are called *protocols*. *Mulan protocols* describe the behavior of agents during conversations. The AUML diagram in Fig. 7 also serves as an overall sketch of the used *Mulan* protocols.

The conversation displayed in Fig. 7 is based on the assumption that the OPA-s already know the identity of their organization agent. However, the organization agent does not know the identities of its OPA-s. OPA-s send a message with their identifiers to their organization agent requesting their local structure which should include all the relevant information extracted from the respective positions in the SONAR organization net such as the neighbors, the implementing and delegating roles, the tasks, etc. After receiving the requests for the local structure and the identifiers from all the position agents, the organization agent proceeds and sends the respective local structure to each position agent. The conversation partners know their respective public keys so all the messages of the conversation are signed with the private keys of the sending parties. However, the aspect of authentication has been left out from Fig. 7 for simplicity.

## 5.2 Assignment of OMA-s to OPA-s

After the generation of the OPA-s and the communication of the local structures to them, the assignment of OMA-s to OPA-s is started. The approach for the assignment process is leaned on [7]. As the organization agent represents some

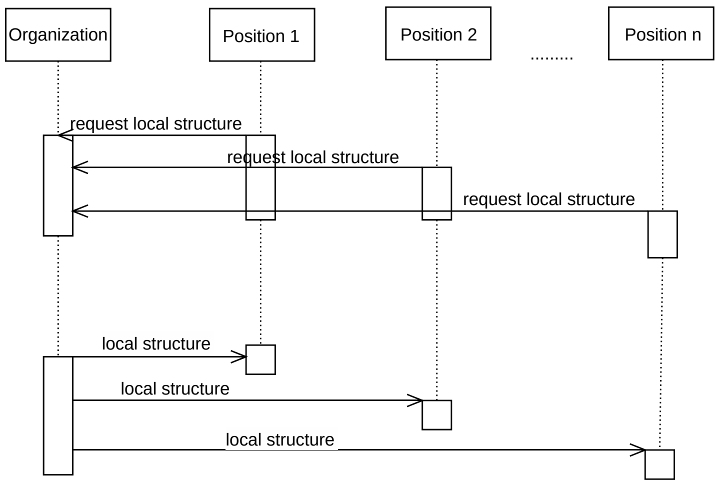


Fig. 7: The organization agent communicating to the generated positions the informations extracted from the respective positions in the SONAR organization net.

kind of a service provider and logical platform to the OPA-s and the potential OMA-s, it assumes at this point the management of the assignment of OMA-s to OPA-s. If an OPA has an open position either because its OMA resigned or it has been fired, the OPA sends a request to the organization agent to start the procedure for the occupation of the open position. The organization agent publishes a job description for the open position to a central registry component named DF[4] (**D**irectory **F**acilitator). A DF is a mandatory component of an agent platform in FIPA that provides a yellow pages directory service to agents. Agents can advertise their services through the DF. In *Mulan* the DF is also an agent with which the organization agent can communicate.

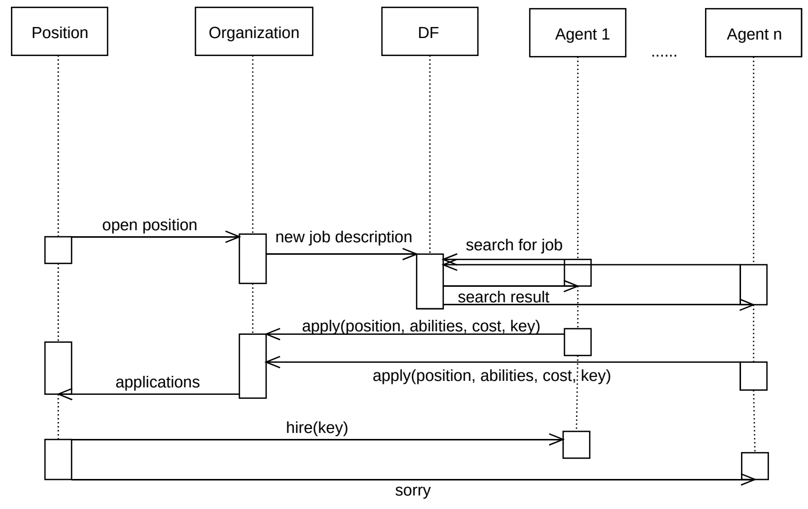
The external agents that are interested in occupying open positions in the organization can search through the DF and apply to the organization agent for a specific open position. The initial assignment of OPA-s with OMA-s is a special case where all OPA-s have open positions. In Fig. 8a, an AUML sequence diagram displays the procedure of occupying a vacant position during the initialization process, while in Fig. 8b, the diagram displays the case when the resignation of a member triggers the start of the procedure to assign a new member for the position. The organization agent sends a description for a new job to the DF. The job description contains the identifier of the vacant OPA, the requirements that applicants have to fulfill, and a time period during which applications for the job are accepted. Agents that find that job description interesting after a search in the DF, apply to the organization agent. Their application includes the description of the job for which they are applying, their public key, their personal abilities as a response to the requirements specified in the job description, and the costs for their service. The applications are received from the organization agent.

After the application period for a vacant position expires, the organization agent sends all received applications to the respective position agent, the OPA selects the new member and lets it know the public key for the authentication during their future communication as well as the fact that it has been hired. In the case of resignation from a member, the member is dismissed only after finishing its ongoing activities on behalf of the organization. Even if a new member may have been hired since the resignation request of the old member, the old member is dismissed only after finishing all its ongoing activities. This means that a position can have more than one member agents for limited time periods. An alternative way for an agent to get the list of open positions within an organization is to send a message to the organization agent itself with a request for the open positions that the organization has. In Fig. 9 this case is displayed as an AUML diagram.

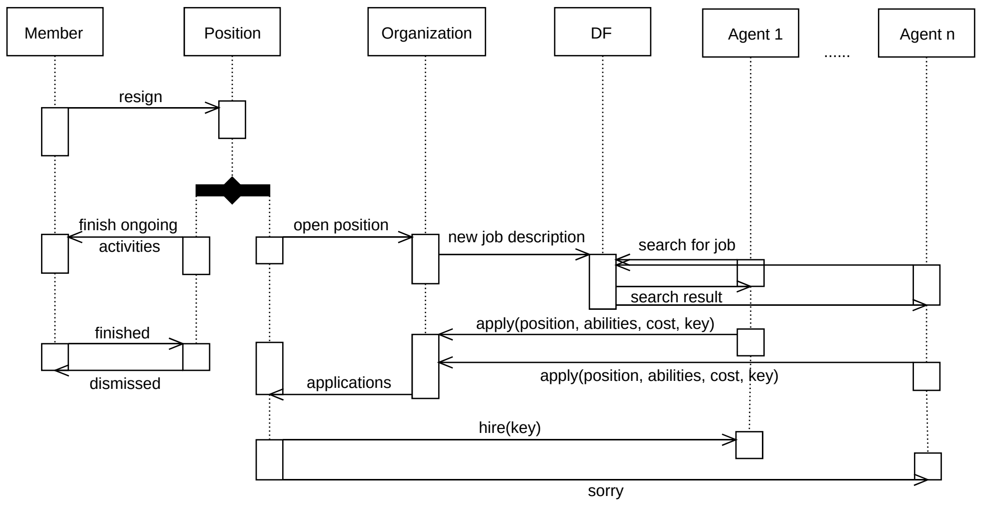
For handling the communication for both the generation of the OPA-s and the assignment of OMA-s to OPA-s an ontology was developed. The ontology was developed with Protégé[5] and was employed as a domain specific shared

[4] See FIPA Agent Management Specification.

[5] Freely available from http://protege.stanford.edu



(a) During initialization.



(b) After the resignation of the member.

Fig. 8: The assignment of an agent as a member to a position agent.

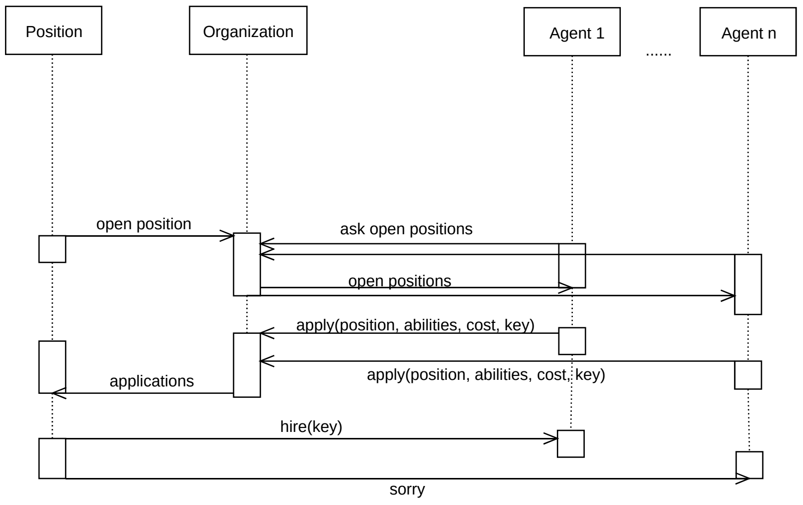


Fig. 9: Agents searching the organization directly for open positions.

vocabulary throughout the conversations between the agents involved in the two phases.

## 6 Conclusion and Outlook

OREDI, the tool presented in this paper, is a Petri net based software tool for modeling SONAR formal organizations as well as for deploying these models to agent organizations. By providing a tool that carries out specification as well as deployment of formal organizations a close link between these two phases of system development has been provided. With OREDI users can build SONAR organization and R/D nets through a combination of graphical interfaces, a set of interaction constraints and context based suggestions without being required to possess active knowledge of the formal rules underlying the models' specifications. The deployment of formal organization models is based on the decoupling of the elements of the multi-agent system that specify the organizational structure from those that act as members of the organization. Positions in the formal organization are deployed as *Mulan* agents (OPA-s) and are embedded in a Mulan *platform*. OREDI provides the specification and implementation of the assignment of OMA-s to OPA-s. Provided the necessary capabilities, any agent can be assigned to an OPA as its OMA and take over the execution of the necessary tasks.

In the future extensions of OREDI both the modeling and the deployment phases will be subject to further development. A special focus should be laid on the modularization of the OREDI models as it can allow the distribution of the modeling process. Formal organizations can be partitioned into modules which can be developed and maintained in separate files. Modules can be linked to each other by delegation relationships. Linking modules can be achieved by adding extra graphical components to OREDI that represent the link to other modules of the organization. Besides, the deployment of formal organization nets as agent organizations can also be extended to include the team formation, team plan formation and team plan execution phases as the corresponding theoretical foundations have already been provided in [6].

## References

1. Köhler, M.: Formalising Multi-Agent Organisations. Proc. of Concurrency, Specification, and Programming CS&P'2006 (2006)
2. Ferber, J., Gutknecht, O. and Michel, F.: From Agents to Organizations: An Organizational View of Multi-agent Systems. AOSE, 214-230 (2003)
3. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
4. Köhler, M., Moldt, D. and Rölke, H.: Modelling the Structure and Behaviour of Petri Net Agents. Proc. of the 22nd Conf. on Application and Theory of Petri Nets 2001, 224–241 (2001)
5. Kummer, O., Wienberg, F. and Duvigneau, M.: Renew — The Reference Net Workshop. Tool Demonstrations — 22nd Int. Conf. on Application and Theory of Petri Nets (2001)
6. Köhler, M. and Wester-Ebbinghaus, M.: Petri Net-Based Specification and Deployment of Organizational Models. 67–81 (2007)
7. Köhler, M.: Koordinierte Selbstorganisation und selbstorganisierte Koordination. Universität Hamburg (2007)
8. FIPA: Foundation for Intelligent Physical Agents. (2003)

# Extended abstract: Service oriented email for organisation modeling and process execution

Petteri Kaskenpalo

AUT University

**Introduction** We describe motivation for and outline technical aspects of a demonstrator system, that enables end users to collaboratively model organisations and processes, and execute these on top of the modeled organisation's overlay networks. While work to date has focused on technical aspects, we are also interested in user level issues relating to the individuals publishing their work interfaces as *services*, and enabling them to link services from other users into complex organisation-wide collaborative activities. Organisational structures are built and maintained dynamically by participants in a peer-to-peer manner.

We have chosen to implement the system on top of email in order to keep the users working in a familiar environment, to emphasise the asynchronous nature of the service interfaces, and to solve email related problems by providing tools for automating and managing email exchanges. This also enables capturing of tacit knowledge embedded in the exchanges. The prototype system models activities with Event-Driven Process Chains (EPC), organisations with HR-XML Consortium formats, and services with WSDL-like definitions. All functionality is implemented in a context-aware agent application, which automates the background message exchange and emails the user when their input is required.

**Motivation** Email is the most used communication tool in business today and its use has been extended far beyond its original purpose of simple information exchange [1].One reason for this success is the asynchronous and distributed nature of the email message exchange, which by definition retains control over the communication, contents and time of interaction with the end-users.

Examples of email uses include information management, task and time management, multi-party activity co-ordination, distributed decision making, negotiations, and voting as a discussion facilitator to name a few. Considering the relatively long development record of messaging tools, it is striking how its very success has led to its many problems; difficulties in dealing with overflow, unsolicited messages, and message linking, archiving and recovery. Inefficiencies can be considerable as people do not have enough time to manage message flows.

We consider *Service Oriented Email* to address organisational workflow challenges in a structured and controlled manner, and at the same provide an organisational approach for solving email related problems. Our research advances the notion of semantic email introduced by McDowell et al. [2], and proposes a scalable distributed semantic email platform for automating complex multiparty update, query, resolution and process activities without exposing individuals beyond customary expectations for email privacy and control.

**Architecture** Agent nodes and the email infrastructure form the system. The agent accesses the user's email server and retrieves relevant messages for processing and communicates with other agents or users. The agent's graphical tool provides for the modeling of EPCs, associated data items, viewing of organisation structures, and making links between the activities and their performers. Standard email client is sufficient in participating in the activity execution.

In terms of the Service-Oriented Computing research planes [3], our *Service Foundations (SF)* layer include the basic messaging, process execution control, and security primitives. The SF layer helps in service binding by evaluating the availability of resources and event timings by qualitative simulation graphs [4]. We also use this technique in evaluating the availability of the required data items, and their security attributes. We divide the *Composition* layer into *Platform Semantics* layer and *User Semantics* layer. The platform semantics layer defines composite services building on the SF primitives. These include *process definitions* for organisational structure management, *shared notice board* service, and composite security services such as *group key generation and distribution* and *voting* protocols. These are locked from modifications by the end user. The User Semantics layer builds on these composite services and SF primitives. The semantics layers implement their process step logic by embedding Java code in the EPC definitions. *Scripting for the Java$^{TM}$Platform* is used for evaluating the embedded code dynamically at runtime in an isolated and controlled execution environment. Only the data items that are accessible at each process step are made available. The Management and Monitoring layer maintains the status of processes that the agent is involved in, and participation in organisational structure management tasks.

We are further defining the required services by implementing test scenarios. Some of these include the *360 Review* personal development process, which involves a number of stages and stakeholders, and provides a number of test scenarios for information confidentiality and disclosure. The Austrian government has introduced an *eLaw-process*, which reveals requirements for document integrity and non-repudiation as well as voting, electronic signing, and version control. The bank *loan approval process* highlights the principles and requirements for separation of duty, and supervisory review and control.

## References

1. Edward J. Lusk. Email: Its decision support systems inroads–an update. *Decision Support Systems*, 42(1):328–332, October 2006.
2. Luke McDowell, Oren Etzioni, Alon Halevy, and Henry Levy. Semantic email. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 244–254, New York, NY, USA, 2004. ACM Press.
3. Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17:223–255, 2008.
4. Ricki G. Ingalls, Douglas J. Morrice, and Andrew B. Whinston. The implementation of temporal intervals in qualitative simulation graphs. *ACM Trans. Model. Comput. Simul.*, 10(3):215–240, 2000.

# Reorganisation and Self-organisation in Multi-Agent Systems

Gauthier Picard[1], Jomi Fred Hübner[1]*, Olivier Boissier[1], and Marie-Pierre Gleizes[2]

[1] SMA/G2I/ENSM.SE, 158 Cours Fauriel
42023 Saint-Etienne Cedex, France
{picard,hubner,boissier}@emse.fr
[2] IRIT, Université de Toulouse, 118 route de Narbonne
31062 Toulouse Cedex 9, France
gleizes@irit.fr

**Abstract.** In the last years, social and organisational aspects of agency have become a major issue in multi-agent systems' research. The conducted works may be structured along two main points of view: an agent-centred point of view (ACPV) and an organisation-centred point of view (OCPV). In both approaches the central notion of multi-agent organisation dynamic is considered. In ACPV, this notion leads to a kind of informal, bottom-up, emergent phenomena that we regroup under the general term of *self-organisation*. In OCPV, this notion gives birth to a huge set of works related to the *reorganisation* of the formal, top-down, pre-existent organisations that are installed in the MAS. In this paper, we propose to position these two approaches to build a comprehensive picture of organisation dynamic in multi-agent systems.

## 1 Introduction

Our aim in this paper is to study and propose a comprehensive view of how one could make multi-agent organisations adapted to dynamics, openness and large-scale environment. In the multi-agent domain, the SASO and the COIN communities are the two that are mainly considering such topics.

SASO (*Self-Adaptive and Self-Organizing systems*) community[3] studies organisation from the point of view of emergent phenomena in complex systems. In multi-agent systems context, we can characterise this point of view as *agent-centred* (ACPV). In fact, designers of such systems first focus on parts of the system-to-be, namely the agents. By designing proper local behaviours and peer-to-peer interactions, the global function of the system is the result of complex interactions and dynamics within the agent society. However, such an engineering approach often inject unpredictability or uncheckability, since the global behaviour is more than the juxtaposition of agents' behaviours. COIN (*Coordination, Organisation, Institutions and Norms in agent systems*) community[4] aims at engineering effective coordination or regulatory mechanisms as a

* Supported by French ANR Project ForTrust ANR-06-SETI-006.

[3] see the SASO Conferences at http://www.saso-conference.org/

[4] see the COIN International Workshop Series at http://www.pcs.usp.br/˜coin/

key problem for the design of open complex multi-agent systems. Contrary to the SASO approach, COIN focuses on an *organisation-centred* point of view (OCPV), in which the designer designs the entire organisation and coordination patterns on the one hand, and the agents' local behaviours on the other hand. At runtime, the agents may consider the constraints imposed by the defined organisation as compulsory or possible guidelines for the coordination of their local behaviours. Systems designed using OCPV can therefore ensure some invariants stemming from the organisation specification.

As we can see, *organisation* is at the intersection of both approaches. This paper mainly aims at clarifying the differences and the common points between these two views focusing on the dynamic dimension. Since there is no universally accepted definition of MAS reorganisation, in this paper, we use *reorganisation* to denote the adaptation of the organisation as promoted in the OCPV, top-down approaches. We use the term of *self-organisation* to denote the ACPV, bottom up approaches where, de facto, an adaptation and modification of the emergent organisation is installed. We propose different possibilities of convergence and complementarity between these processes. The remainder of the paper is structured as follows. In Section 2, we first provide a comprehensive view of the concept of organisation in MAS. Section 2.4 dresses up a comparison of the two points of views with respect to the adaptability and checkability properties. Section 3 analyses the adaptation of organisations in ACPV (self-organisation) and OCPV (re-organisation), so as to propose in Section 4 definitions of self-organisation and reorganisation concepts. Finally, Section 5 concludes this paper with some perspectives for future works and roadmap.

## 2 Comprehensive View of Organisations in MAS

There is still no unanimously accepted definition of what is called "organisation" in MAS. Its meaning often varies between two basic views [30]: (i) a collective entity with an identity that is represented by (but not identical to) a group of agents exhibiting relatively highly formalised social structures [39], (ii) a stable pattern/structure of joint activity that may constrain or affect the actions and interactions of agents towards some purpose [6]. As we can see, organisation refers, in a general sense, to a *cooperation pattern* that can be more or less formalised. As in Sociology [2], it may concern the expression of a division of tasks, a distribution of roles, an authority system, a communication system, or also a contribution-retribution system. According to [18], this range of topics may also be extended to knowledge, culture, memory or history. This is what Parunak et al. express in different terms when they propose a definition of organisation, relatively to self-organisation, at three levels [35]: (i) an order (measure) on organisations, i.e. mapping from the set of organisations to the set of real numbers; (ii) a process in a single system in which the previous measure increases with time (from less organised to more organised); (iii) the structure resulting from the previous evolution.

Both views are generally not mutually exclusive and have led to different approaches in the domain. As in [3], we focus on a few features in order to build a comprehensive view of them. First, we will take into account the "definition process" of the agents' organisation (Sections 2.1) and then consider its "representation" within the agents' minds (Section 2.2). As what happens with every classification attempt, the one proposed here has its limits and must be considered as an analysis grid of the different works and not as a definitive view on multi-agent organisations in MAS. The two dimensions of this grid are continuous, and it is completely possible to identify approaches that are at the boundary of two categories.

## 2.1 Agent-Centred View vs Organisation-Centred View

The first axis of the grid is an extension of the *agent-centred* and *organisation-centred* points of view initially proposed in [31].

The *agent-centred* point of view takes the agents as the "engine" for the organisation. Organisations only exist as observable emergent phenomena which state a unified bottom-up and objective global view of the pattern of cooperation between agents (see first row in Fig. 1-a-b). For instance (case (a)), in an ant colony [14], no organisational behaviour constraints are explicitly and directly defined inside the ants. The organisation is the result of the collective emergent behaviour due to how agents act their individual behaviours and interact in a common shared and dynamic environment. A similar point of view may be considered in the different reactive self-organisation approaches that exist in the literature [38]. In a more cognitive way (case (b)), the studies on coalition formation define mechanisms (within agents, e.g. social reasoning [40]), to build patterns of cooperation in a bottom-up process. In this view, the pattern of cooperation both structures and helps the agents in their collaborative activities.

The *organisation-centred* point of view sees the opposite direction: the organisation exists as an explicit entity of the system (see second row in Fig. 1-c-d). It stresses the importance of a supra-individual dimension [18] and the use of primitives that are different from the agents' ones. The pattern of cooperation is settled by designers (or by agents themselves) and is installed in a top-down manner in order to constrain or define the agent's behaviours. Note that the observer of the system can obtain a description of the system's organisation. For instance, in a school we have documents that state how it is organised. Of course, besides the explicit description of the organisation, the beholder can also observe the real school's organisation which is, possibly, different from the formal one.

## 2.2 Organisation Awareness vs Organisation Unawareness

From an agent architecture perspective, we can further refine these two points of view by considering an orthogonal axis regarding the agents' capabilities to represent and reason about its organisation.

In the first column of Fig. 1, the agents don't know anything about the organisation. In case (a) the agents don't represent the organisation, although the *observer* can see an emergent organisation. In some sense, they are not aware that they are part of an organisation. In case (c), the organisation exists as a specified and formalised schema, made by a designer but agents don't know anything about it and do not reason about it. They simply comply to it as if the organisational constraints were hard coded inside them (e.g. the MAS resulting from some AOSE, Agent-Oriented Software Engineering) methodologies where the agent's code is generated from an organisational specification [27, 1]).

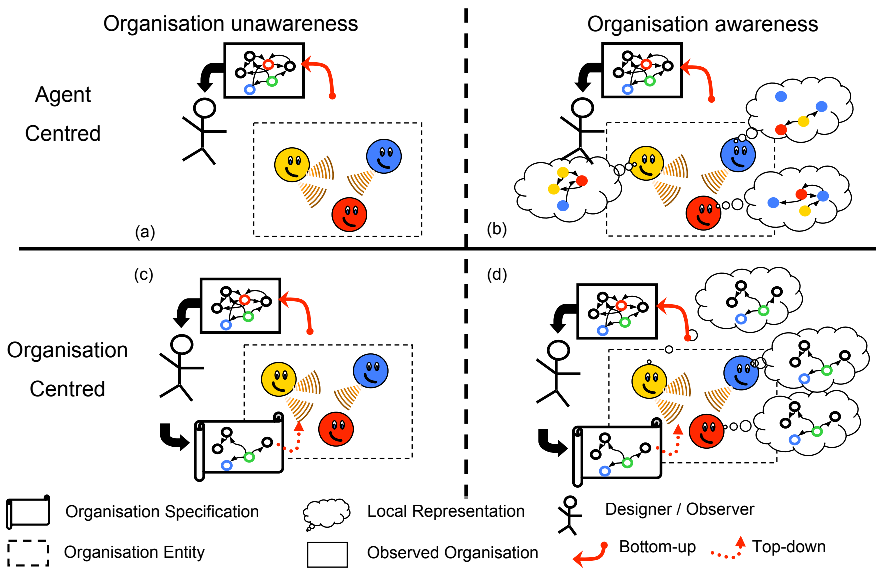


**Fig. 1.** Comprehensive view on organisations in MAS: (a) Emergent Organisation MAS; (b) Coalition Oriented MAS; (c) Agent Oriented Software Engineering; (d) Organisation Oriented MAS. The Designer/Observer may be the Developer/User (exogenous case) or a set of agents (endogenous case).

In the second column, we consider the cases where agents have some *representation of the organisation* in which they are executing. In case (b), each agent has an internal and local representation of cooperation patterns which it follows when deciding what to do (e.g. social networks for coalition formations [40]). This local representation is obtained either by perception, communication or explicit reasoning (e.g. social reasoning as in [40]) since, in an agent-centred view, there isn't, *a*-priori, any explicit global representation of the organisation which is available to the agents. In case (d), agents have an explicit representation of the organisation which has been defined (organisation-centred view). The agents are able to reason about it and to use it in order to initiate cooperation with other agents in the system.

In the literature, some agents' organisation approaches fit to a specific case shown in Fig. 1, others are based on multiple cases. For instance, proposals concerning reorganisation approaches for formal organisations may combine cases (b) and (d) in the sense that agents are using their internal mechanisms to adapt the organisation that was imposed on the system. The bottom-up or top-down manipulation of the organisation may be realised either *endogenously* (i.e. realised by the agents belonging to the organisation themselves) or *exogenously* (i.e. by an external designer, a human or agents outside of the organisation).

### 2.3 Applying this Comprehensive View to MAS

To be clearer, we can position some concrete existing systems and approaches within the grid.

1. *Emergent Organisation MAS (top-left)*: Here agents are unable to reason on the organisation since it is not modelled. Agents self-organise in a peer-to-peer fashion or using indirect communication via the environment. As example, we can cite swarms-based systems, adaptive multi-agent systems (AMAS) [20] and more generally all the works coming from the SASO community.
2. *Coalition Oriented MAS (top-right)*: Here agents are still unable to reason about the global organisation but can build inner models of the interactions/dependence relations with their neighbourhood with respect to predefined social rules and patterns. Common examples of such approaches are multi-agent coalitions [40] or more simply systems based on the contract net protocol [41].
3. *Agent-oriented software engineering (bottom-left)*: This category of approaches consider organisation at the design-time. Organisations are specified before encoding the agents. Agents can reason on the organisation at run-time but cannot be able of modifying it. Examples of such approaches are mainly found in the AOSE community which proposes several multi-agent oriented methodologies that focus on the organisational model such as MASE [11] and INGENIAS [36].
4. *Organisation Oriented MAS (bottom-right)*: These approaches are influenced by both AOSE and social reasoning, in the sense that organisation are used by designer to specify the system-to-be and by the agents that can perform organisational acts and possibly modify the organisation. Example of such approaches are AGR [16], TAEMS [32], STEAM [44], $\mathcal{M}$OISE$^+$ [25] or ISLANDER[15].

### 2.4 Checkable vs Adaptive Organisations

Previous sections expounded the two dimensions of our comprehensive view grid, and some examples. Let us now come back to the main motivations of the existence of the two proposed points of view. On the one hand, SASO community aims at providing *adaptive* systems inspired by natural, biological and physical systems that display real adaptation and autonomy capabilities. However such properties also raise some design problems: how can we ensure that the system will converge towards a specified state and not continuously adapt and change? On the other hand, AOSE community, and to a lesser extent COIN community, firstly aimed at providing engineering tools to design systems using organisational concepts rather than classical object- or agent-oriented concepts. Such concepts and models enable a designer to *check* whether the system behaves as specified by the organisation specification. Thus, the specified cooperation patterns ensure certain properties on the system such as sanction triggering using formal commitments [17], or global states using Object-Z framework [23]. Nevertheless, such an *a priori* model of the organisation represents a limitation in terms of adaptation: how can we model all the possible organisation changes at design-time? Even if some works proposed reorganisation approaches to tackle this problem, it is not surprising that these two views often collide since they focus on two opposite extrema of the same spectrum from full adaptation to full checkability.

As we can see, these two points of view focus on two different extrema of the same spectrum, from complete adaptation to complete checking. Between these, midterms can be reached: systems in which partial checking and verification can be processed, and systems in which total checking is feasible under certain hypothesis. We will further aim at identifying complementarity in order to dress up perspectives for unifying self- and reorganisation approaches.

## 3 Where are Self- and Re- in the Adaptation of Organisations?

In this section we analyse several aspects regarding the two main approaches related to the organisation modification process –reorganisation and self-organisation– through the following questions: *what*, *when*, *why*, *who* and *how*.

### 3.1 What is Changed?

Changing the organisation may imply changes within the system at different levels and at different extents. These changes strongly depend on the chosen view (ACPV or OCVP) and on the organisational capabilities of agents (being or not aware of the organisation).

Considering the *emergence-based MAS*, the observed organisation of the agents may be changed along different dimensions: spatial configuration, neighbourhoods, differentiation/specialisation of the agents. In many approaches, the spatial configuration of the system strongly constrains the capabilities and the potential that the system can exhibit. Therefore, system adaptation expresses as changing this spatial configuration such that it can behave in a more adapted way to the new environmental pressure. For instance, in a self-constructed mechanical system [4], mechanical parts (agents) change by themselves their attachment to other parts as to optimise a trajectory function, which may change at runtime, without being aware of the global organisation (i.e. the whole mechanism configuration). The same kind of adaptation is also found in collective robotics [10] where the spatial position of all the agents is strongly related to the plans that they have to execute. The observed organisation may also be changed by changing the partners with which the agents of the system interact or the way they interact which each other. Several other self-organising mechanisms are also presented in [12, 22]. In other systems, adapting a system is visible as changes in the tasks and/or goals that agents achieve. For instance, in bio-inspired approaches the self-differentiation installs a bottom-up kind of implicit role allocation [14]. To conclude, such approaches often propose systems able to *explore the space of possible organisations*.

In the *coalition-oriented MAS*, the topology of the organisation is expressed in terms of local social configurations: agents belong to neighbourhoods, coalitions (agents know each other and cooperate together) expressed in terms of power relations, dependence relations or social commitments. For instance in [7], agents change their interactions depending on a *trust* evaluation, which is calculated using past experiences and opinions from other trusted agents. This represents a regulatory local mechanism that leads the system to a social order. Coalitions are also built and deleted using such mechanisms, in market places [9]. In distributed problem solvers like [34] coalition is defined/changed from the election depending on the current state of the neighbourhood (e.g. the most constrained agent is elected to propose a solution) or the advance in the solving process.

Considering the *organisation-oriented MAS* approach, since there exists some explicit specification, the change of organisation may be considered at two levels: (i) change of the definition of the organisation itself and (ii) change of the allocation of roles to the agents, i.e. the way the multi-agent organisation itself is built. An example of the former is the work of Hübner et al. [26], in this work the agents are able to evaluate their organisation, identify that the problem is caused by the current definition (or specification) of the organisation, and then decide to change the specification. This kind of change usually implies changes in the role allocation, since the very set of available roles can be changed. Examples of the latter case of reorganisation include, for instance, the work of Kamboj et al. [28] where the reorganisation consists in the spawning and composition of agents in the system in order to reassign the roles the agents play. The reasons for changing the organisation consist in the wish to have a structure adapted to the environment and the task structure that has to be processed by the agents. Another example is the work of Glaser et al. [21] where the organisation is changed by the entrance of new agents in the system playing a particular role. The new agent is accepted only when it increases the utility of the overall system. Organisational structure generation has also been proposed as arising from local [19], global [8], and hybrid perspectives.

Concerning the *methodological viewpoint* (bottom-left), the designer may change the *model* of the system at several levels. At a low level, the agentification (i.e. the way model entities are specified as agents) may be changed, and therefore the organisation too. At an upper level, the organisation model may be changed by adding new roles, groups, tasks, etc.

## 3.2 When and Who Changes the Organisation?

Depending on the approach, the organisation can be changed at different times during the system life cycle. It can be initiated by different actors. The decision when to start the process can be either static or dynamic. In the first case the process is started according to a predefined criterion fixed within the definition of the organisation. In the second case, the reorganisation process is a consequence of the functioning of the system. It means that if agents do not meet one or several criteria (goal, performance etc.) the organisation is changed. In [13], Dignum defines the "when" to reorganise as linked to the utility of the organisation (interaction success, role success and structure success) and to the utility of the agent (different for each agent, depends on its goals, resource production and consumption).

The process is static when the adaptation of the system is performed by the designer, during the development of the system. Only designers (seen as Oracles) can detect, by using appropriate model checking, fast prototyping, or simulation tools like in IODA [29], which exhibit global undesired behaviours. Agents are not actors of the organisation adaptation process, but are only an implementation of the organisational model specified by the designer.

The process is dynamic when the organisation is modified at runtime. This modification can be performed by an external entity (designer, other systems, etc.) that acts on living agents, or autonomously by agents themselves. For instance, agents within a self-organising system change indirectly the organisation as a reaction to an *environmental change* detected at the agent level. So, in the example of self-constructed mechanics, mechanical components can change their weights (length, pressure, etc.) when they receive an external positive or negative feedback (by propagation), concerning the distance to the objective trajectory [4]. A coalition reacts also to changes detected at the coalition level, by *social pressure* (e.g. a coalition received a negative opinion concerning its leader), and therefore changes the organisation (e.g. by changing its leader) [33].

## 3.3 Why and How does the Organisation Change?

While we can identify different kinds of *changing objects* according to the organisation model being used, we can also identify some types of *changing processes* that will be detailed in the remainder of this section: (*i*) predefined changes, (*ii*) controlled changes and (*iii*) emergent changes. The general reason to trigger the change is that the organisation does not help in achieving the social purpose. In other words, the current organisation constrains the agents' behaviours to those which do not fit the behaviours that draw the social purpose. Such situations may happen, for instance, when the environment has changed, the MAS purpose has changed, the performance requirements are not satisfied, the agents are not capable of well playing their roles, a new task request arrives and the current organisation is not appropriate, etc.

**Generic Organisation Process.** Generally, in order to modify an organisation, we can identify a generic organisation process (or adaptation process) which will be differently implemented depending on the chosen adaptation approach. This process is usually composed of two main phases –monitoring and reparation. This last one, depending on the MAS type, can be decomposed into design, selection and execution phases [42] or selection and execution phases only. This adaptation process can be part of a more general environment-system life-cycle: perception, adaptation process, action, perception, and so on. Inherent problems of this process and the chosen approach are detailed in the remainder of this section.

1. The *monitoring* phase aims at detecting inadequacy problems between the system, the organisation or the agents, and the environment. Whatever is the entity responsible for the monitoring, and therefore whatever is the abstraction level of this detection (macro or micro), it is advisable to define the set of situations of non adaptation.
2. Once a need to adapt is detected, the *reparation* phase performs a process in order to find back an function as optimal as possible, at runtime.
   (a) For this, the *design* phase aims at defining and developing a set of possible alternatives for the current organisation, in a top-down or a bottom-up manner.
   (b) The *selection* phase determines one alternative to modify the organisation. The main problem is therefore the definition of *evaluation criteria* for evaluating the different alternatives.

(c) The *execution* corresponds to the implementation of the previously chosen alternative.

**Predefined Change of Organisation.** In this case we consider that changes are already planned and expressed by the designer to be performed at a precise moment [5]. For instance, a soccer team has previously accorded to change its formation at the 30 minutes of the match [43]. In this approach, the execution of the adaptation process is quite straightforward. Monitoring is performed by agents themselves or an external entity (e.g. the coach of the soccer team which consults a timer to know when to trigger the team change). The design phase, which determines all these trigger conditions, is not performed at runtime, but at design time. Selection and execution phase are immediate since trigger conditions are coupled with predefined actions, performed on the fly.

**Controlled Change of Organisation.** In this case, the system does not know when the organisation will change, but knows what are the conditions to trigger a change process, that will be implemented following a known procedure (e.g. a team has an expert that controls the reorganisation process). The main difference with the previous approach is that the designer des not know a priori when and how the organisation has to be changed. However he is able to define strategies for monitoring and repairing the organisation. These strategies can be used by agents to control and drive the organisation process. This process can be performed in two ways: either (*i*) an *endogenous* approach where a particular agent (centralisation) or agents themselves (coordinated decentralisation) will manage the reorganisation; or (*ii*) an *exogenous* approach where the user of the system, or an external system, controls the process. During the execution of an instance of the specified system, the entity responsible for the change (designer or agent) can detect the organisation is not adapted because of *inadequate performances*, and can therefore modify the model and the specification for improving the performances by *programming* a more adapted organisation.

The monitoring phase identifies a situation where the current *organisation is not adapted* and does not satisfy the needs of the MAS. The main problem this phase is *how to identify whether the social purpose is not being achieved because the current organisation does not allow it*. Many other reasons may cause the unaccomplishment of the MAS purpose (e.g. the social purpose is impossible to be achieved). In some cases to change the organisation is not helpful. Even in the case we know the problem can be solved by the reorganisation process, the new problem is to identify *which part of the organisation is causing the problem* in order to set the correct reorganisation level. The part of the organisation that is responsible for the problem can be either its specification (e.g. set of possible roles) or the current instantiation of that specification (e.g. who plays which role). The reparation phase requires then to execute the design, selection and execution phases. The design phase intends to develop a set of possible alternatives for the current organisation. The design of this set of alternatives can be based on (*i*) a search in a library of predefined organisations or (*ii*) their creation on demand. In the first case, the problem is to identify which predefined organisation is appropriate for the failure caught by the monitoring phase. In the second case, we have to deal with yet another problem: the hugeness of the search space for new organisational specifications (this search space is defined by the organisational model). During the execution, means to change the current organisation must be defined without causing any failure. For example, how an agent will deal with the fact that the role it is playing was removed in the new organisation? What it will do with the commitments adopted under this extinguished role?

An example of controlled change of organisation is $\mathcal{M}$OISE$^+$ [26]. This work considers the organisational structure and functioning. They explicitly focus on controlling the reorganisation process for which they consider the four phases discussed above. In their view reorganisation is a cooperative process itself which is performed in an endogenous and decentralised way. This process may itself be the subject of a dedicated organisation composed of a hierarchy of roles specialised in the management of the reorganisation. Another example is [24], where a centralised reorganisation process is used, based on TAEMS (Task Analysis, Environment Modeling, and Simulation) modelling language and a diagnosis expert subsystem in charge of detecting deficiencies in the organisation and assisting in the creation of a solution. Its monitoring phase identifies failures when the system does not behave as expected by its functional model. Examples of controlled self-organisation can be found in some agent-based problem solvers, like [34] or [37], in which predefined roles are taken by agents depending on the state of the solving process. In a nominal situation, agents try to find a value by checking constraints shared with neighbours, but without knowing their values. If an agent detects that its neighbouring agents (sharing constraints) are blocked or over-constrained, it will take the *mediator* role [34] or will launch an *election process* [37] so as to force them to share their value as to set a new value minimising conflicts, for instance. Therefore, agents can play two different roles, predefined by the designer, at different times of the solving process.

**Emergent Change of Organisation.** In this last case, as for the controlled change of the organisation, the time to trigger the process is not predefined by the designer. The reasons to change are equivalent to the previous cases: the system behaviour is not adequate in its environment. The main characteristic of the process is that it is not led by an entity external from the system. Difference with other approaches is that the designer does not have global knowledge of strategies to monitor and repair the organisation. All the knowledge he has is local knowledge that is manipulated at the agent level, at a local level. Thus, the organisation which is defined comes from the interactions between the agents from their local perception and actions.

The monitoring phase is performed endogenously, by one or more agents. The designer equips them with capabilities to detect at local level that the organisation is no more globally adapted. For this, agents are able to know *they are not adapted*. Next, the agent that detects the problem will perform the reparation phase. It consists in the selection of one or more actions among the set of all possible actions, followed by the execution of this chosen action. In [38], the agent chooses and performs the action it judges as being the most appropriate with respect to a local evaluation criterion called cooperativeness. This action can be defined and implemented, or learned at runtime [14]. This phase is also realised during runtime. An agent *reacts* to change its position within the organisation/topology in order to adapt the system as a whole, or to change its own behavioural specialisation [14]. An agent is able to autonomously decide the action that will change the organisation, by removing itself from the system for instance. An agent can also decide to act cooperatively, i.e. by taking into account its neighbours' states [38]. For instance, in a self-constructed mechanics, a component receiving a negative *feedback* from the environment must change its weights in order to change its function and its interactions with its neighbourhood as cooperatively as possible and therefore it reduces the negative feedback. Coalitions modify their organisation when *relations between agents are not adapted*. Such changes are the result of a *reasoning process* based on social concepts such as powers and dependencies. For instance, in a coalition of surveillance drones, agents change their leader after the current one has displayed some lack of computation or communication capabilities, in order to maintain the group adapted to the collective mission [33].

## 4 Discussion and Definitions

| | | **Organisation Unawareness** | **Organisation Awareness** |
|---|---|---|---|
| **What?** | **ACPV** | Topology<br>Weights, Influence<br>Differentiation<br>Neighbourhood | Dependencies<br>Commitments<br>Powers |
| | **OCPV** | Design model<br>Agentification | Organisation specification<br>Role assignment |
| **When?** | **ACPV** | At runtime | At runtime |
| | **OCPV** | At design time | At runtime |
| **Why?** | **ACPV** | Agents are not adapted | Relations between agents are not adapted |
| | **OCPV** | Performances are not adequate | Organisation is not adapted |
| **Who?** | **ACPV** | Agents (environmental pressure) | Agents (social pressure) |
| | **OCPV** | Designer | Designer & Agents |
| **How?** | **ACPV** | Reacting<br>indirectly on the organisation<br>directly on the environment | Reasoning<br>indirectly on the organisation<br>directly on cooperation patterns |
| | **OCPV** | Programming<br>directly the organisation<br>directly the environment | Organising<br>directly the organisation<br>directly the cooperation patterns |

**Table 1.** Aspects of organisation adaptation

Table 1 provides a synthetic view of the different points we analysed above. We can thus propose definitions of reorganisation and self-organisation concepts, in the context of Figure 1 and Table 1.

**Definition 1.** Reorganisation *is a process, endogenous or exogenous, concerning systems in which organisation is explicitly manipulated through specifications, constraints* *or other means, in order to ensure an adequate global behaviour, when the organisation is not adapted. Agents being aware of the organisation state and structure, they are capable of manipulating primitives to modify their social environment. This process can be both initiated by an external entity or by agents themselves, by reasoning directly on the organisation (roles, organisational specification) and the cooperation patterns (dependencies, commitments, powers).*

This process thus appears on the right side of the grid, and mainly concerns organisation-oriented systems and, to a lesser extent, coalition-based systems.

**Definition 2.** Self-organisation *is an endogenous and bottom-up process concerning systems in which only local information and representations are manipulated by agents unaware of the organisation as a whole, in order to adapt the system to the environmental pressure by modifying indirectly the organisation, therefore by changing directly the system configuration (topology, neighbourhoods, influences, differentiation), or the environment of the system, by local interactions and propagation, by avoiding predefined model biases.*

This process appears on the top row of the grid, and therefore concerns emergent organisations and, to a lesser extent, coalition-based-systems. So, we can identify a continuum between self-organising systems and reorganising ones, via coalition-based systems. Reorganisation and self-organisation are also two different implementations of the same generic process of adaptation of organisations: detection and reparation. In a self-organising system, this process is decentralised, implicit and endogenous, giving the responsibility to agents and often initiated by an environmental change. In a reorganising system, this process can be decentralised or not, but always explicit and directly performed by entities (designer or agents) manipulating organisational primitives.

## 5 Conclusion

In this paper we presented a comprehensive view of the organisational aspects in multi-agents systems from agent-centred and organisation-centred points of view. We also underlined the main differences between the reorganisation and self-organisation processes by analysing the reasons and scope of organisation changes (what, when, who, why, how). Following this analysis, we proposed definitions of reorganisation and self-organisation, with respect to the organisation-centred and agent-centred points of view. However, these two points of view are not incompatible if we consider them at different moments of the life time: for instance, emergence at the beginning, capitalisation and injection of the capitalised organisation in the functioning of the system, change of this organisation (reorganisation), and self-organisation once the set of known possible organisations is no more sufficient.

Once we have dressed up a comprehensive view of the organisation adaptation context, in future works we will aim at defining a formal framework capturing the notions of both views as to make them cooperate. We can imagine using self-organising mechanisms at the organisation model level to explore the space of possible organisations, and to propose more adequate organisations and norms. From the opposite viewpoint, we can imagine using organisational specification to set boundaries for the emergent behaviours of self-organising systems, by defining, for instance, regimented constraints that agents cannot violate and enforced constraints that agents may violate to explore new organisational configurations.

## References

1. F. Bergenti, M.P. Gleizes, and F. Zambonelli. *Methodologies and Software Engineering for Agent Systems*. Kluwer, 2004.
2. P. Bernoux. *La sociologie des organisations*. Seuil, 3ème edition, October 1985.
3. O. Boissier, J. F. Hübner, and J. S. Sichman. Organization oriented programming from closed to open organizations. In Gregory O'Hare, Michael O'Grady, Oguz Dikenelli, and Alessandro Ricci, editors, *Engineering Societies in the Agents World VII (ESAW 06)*, volume 4457 of *LNCS*, pages 86–105. Springer-Verlag, 2007.
4. D. Capera, M.-P. Gleizes, and P. Glize. Mechanism Type Synthesis based on Self-Assembling Agents. *Journal of Applied Artificial Intelligence*, 18(9-10):921–936, 2004.
5. T. Carron and O. Boissier. Towards a temporal organizational structure language for dynamic multi-agent systems. In *Pre-Proceeding of the 10th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001)*, 2001.
6. C. Castelfranchi. Modeling social action for AI agents. *Artificial Intelligence*, (103):157–182, 1998.
7. C. Castelfranchi. Engineering Social Order. In *ESAW '00: Proceedings of the First International Workshop on Engineering Societies in the Agent World*, pages 1–18. Springer-Verlag, 2000.
8. D. D. Corkill and V. R. Lesser. The use of meta-level control for coordination in distributed problem solving network. In Alan Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI'83)*, pages 748–756, Los Altos, CA, 1983. William Kaufmann.
9. D. Cornforth, M. Kirley, and T. Bossomaier. Agent Heterogeneity and Coalition Formation: Investigating Market-Based Cooperative Problem Solving. *Autonomous Agents and Multiagent Systems, International Joint Conference on*, 2:556–563, 2004.
10. M. M. de Weerdt and B. J. Clement. Introduction to planning in multiagent systems. *Multiagent and Grid Systems An International Journal*, 5(4), 2009.
11. S.A. DeLoach. *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook Series : Multiagent Systems, Artificial Societies, and Simulated Organizations*, volume 11, chapter The MaSE Methodology. Kluwer Academic Publishing (available via Springer), 2004.
12. G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos. Self-Organisation and Emergence in Multi-Agent Systems: An Overview. *Informatica*, 30(1):45–54, 2006.
13. V. Dignum, F. Dignum, and L. Sonenberg. Towards dynamic organization of agent societies. In G. Vouros, editor, *Workshop on Coordination in Emergent Agent Societies*, pages 70–78, 2004.
14. A. Drogoul, B. Corbara, and S. Lalande. MANTA: New experimental results on the emergence of (artificial) ant societies. In Nigel Gilbert and Rosaria Conte, editors, *Artificial Societies: the Computer Simulation of Social Life*, pages 119–221. UCL Press, London, 1995.
15. M. Esteva, J.A. Rodriguez-Aguiar, C. Sierra, P. Garcia, and J.L Arcos. On the formal specification of electronic institutions. In Frank Dignum and Carles Sierra, editors, *Proceedings of the Agent-mediated Electronic Commerce*, LNAI 1191, pages 126–147, Berlin, 2001. Springer.

16. Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agents systems. In Yves Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press, 1998.
17. N. Fornara and M Colombetti. Specifying and enforcing norms in artificial institutions. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 3*, pages 1481–1484, 2008.
18. L. Gasser. Organizations in multi-agent systems. In *Pre-Proceeding of the 10th European Worshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001)*, Annecy, 2001.
19. L. Gasser and T. Ishida. A dynamic organization architecture for adaptive problem solving. In *Proceedings Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 185–90. The MIT Press & AAAI Press, 1991.
20. J.-P. Georgé, B. Edmonds, and P. Glize. Making Self-Organizing Adaptive Multi-Agent Systems Work - Towards the engineering of emergent multi-agent systems (chapter 8). In *Methodologies and Software Engineering for Agent Systems*, pages 319–338. Kluwer, 2004.
21. N. Glaser and P. Morignot. The reorganization of societies of autonomous agents. In Magnus Boman and Walter Van de Velde, editors, *Multi-Agent Rationality*, LNAI 1237, pages 98–111, Berlin, 1997. Springer.
22. S. Hassas, G. Di Marzo-Serugendo, A. Karageorgos, and C. Castelfranchi. On self-organising mechanisms from social, business and economic domains. *Informatica*, 30(1):63–71, 2006.
23. V. Hilaire, P. Gruer, A. Koukam, and O. Simonin. Formal driven prototyping approach for multiagent systems. *International Journal of Agent-Oriented Software Engineering*, 2(2):246–266, 2008.
24. Bryan Horling, Brett Benyo, and Victor Lesser. Using self-diagnosis to adapt organizational structures. In *Proceedings of the 5th International Conference on Autonomous Agentes (Agents' 01)*, 2001.
25. J.F. Hübner, J.S. Sichman, and O. Boissier. $\mathcal{M}$OISE$^+$: Towards a structural, functional, and deontic model for MAS organization. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2002)*, pages 501–502. ACM Press, 2002.
26. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the $\mathcal{M}$OISE$^+$ for a cooperative framework of MAS reorganisation. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, volume 3171 of *LNAI*, pages 506–515, Berlin, 2004. Springer.
27. C. Iglesias, M. Garrijo, and J. Gonzalez. A survey of agent-oriented methodologies. In *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories*, pages 317–330, Heidelberg, 1999. Springer-Verlag.
28. S. Kamboj and K.S. Decker. Organizational self-design in semi-dynamic environments. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.
29. Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations : From theory to implementation. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 383–387. IOS Press, 2008.
30. O. Boissier L. Coutinho, J.S. Sichman. Modeling dimensions for multi-agent systems organizations. In V. Dignum, F. Dignum, B. Edmonds, and E. Matson, editors, *Agent Organizations: Models and Simulations (AOMS), Workshop held at IJCAI 07*, 2007.
31. C. Lemaître and C.B. Excelente. Multi-agent organization approach. In Francisco J. Garijo and Christian Lemaître, editors, *Proceedings of II Iberoamerican Workshop on DAI and MAS*, 1998.

32. V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q.. Zhang. Evolution of the gpgp/taems domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004. Kluwer Academic Publishers.
33. M. T. Long, R. R. Murphy, and J. Hicinbothom. Social roles for taskability in robot teams. In *International Conference on Intelligent Robots and Systems (IROS'07)*, pages 2338–2344. IEEE, 2007.
34. R. Mailler and V. Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 438–445. IEEE Computer Society, 2004.
35. H. V. D. Parunak and S. A. Brueckner. Engineering swarming systems. In *Methodologies and Software Engineering for Agent Systems*, pages 341–376. Kluwer, 2004.
36. Juan Pavón and Jorge J. Gómez-Sanz. Agent oriented software engineering with ingenias. In Vladimír Marík, Jörg P. Müller, and Michal Pechoucek, editors, *CEEMAS*, volume 2691 of *Lecture Notes in Computer Science*, pages 394–403. Springer, 2003.
37. G. Picard, M.-P. Gleizes, and P. Glize. Distributed Frequency Assignment Using Cooperative Self-Organization. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on*, pages 183–192. ACM Press, 2007.
38. G. Picard and P. Glize. Model and Analysis of Local Decision Based on Cooperative Self-Organization for Problem Solving . *Multiagent and Grid Systems*, 2(3):253–265, septembre 2006.
39. W. R. Scott. *Organizations: rational, natural and open systems*. Prentice Hall, 4 edition, 1998.
40. J.S. Sichman, R. Conte, Y. Demazeau, and C. Castelfranchi. A social reasoning mechanism based on dependence networks. In Tony Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 188–192, 1994.
41. R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transaction on Computers*, 29(12):1104–1113, 1980.
42. Y. So and E.H. Durfee. An organizational self-design model for organizational change. In *Proceedings of AAAI93 Workshop on AI and Theories of Groups and Organizations*, 1993.
43. P. Stone and M.M. Veloso. Task decomposition and dynamic role assignment for real-time strategic teamwork. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop Agent Theories, Architectures, and Languages (ATAL-98)*, LNCS 1555, pages 293–308, Berlin, 1999. Springer.
44. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Reseearch*, 7:83–124, 1997.

# Modeling an Open and Controlled System Unit as a Modular Component of Systems of Systems

Matthias Wester-Ebbinghaus, Daniel Moldt

University of Hamburg, Department of Informatics
Vogt-Kölln-Straße 30, D–22527 Hamburg
{wester,moldt}@informatik.uni-hamburg.de

**Abstract.** Modern software systems are frequently characterized as systems of systems. Agent-orientation as a software engineering paradigm exhibits a high degree of qualification for addressing many of the accompanying challenges. However, systems of systems demand for means of hierarchical/recursive decomposition that are not inherently rooted in the agent-oriented paradigm. We present a model of a system unit that both embeds system actors and is itself embedded as a collective system actor in surrounding system units. We supply generic concepts and mechanisms for combining the system unit's control structures and its openness. This allows the system unit to be applied at arbitrary levels of a system of systems in a recursive fashion.

## 1 Motivation

Today's software systems are frequently characterized as inherently distributed and heterogeneous *systems of systems* [1] whose parts potentially exhibit a great deal of operational and managerial independence [2–4]. Agent-orientation has emerged as a software engineering paradigm that is in many respects ideally suited to deal with these kinds of systems. It advocates flexible, high-level interactions between system parts that exhibit a great deal of local freedom and initiative (agents). Organization-oriented stances have proven as effective means to combine local agent autonomy and heterogeneity with controllability and predictability at the system level. Various approaches have developed where agents are integrated (taking on roles / occupying positions) into an organizational framework that constrains and imprints organizational behavior [5].

However, it is the system of systems aspect that (at least in the first instance) poses problems for agent-orientation. A system of systems perspective inherently demands a hierarchic treatment of different levels of abstractions, where level of abstraction relates to the granularity of system parts studied at each level. Agent-orientation on the other hand features a rather flat decomposition, namely "decomposing problems in terms of autonomous agents" [6]. As analyzed in [7] the *component view* of objects (objects may be composed of other objects in a recursive fashion) was lost with the transition from object- to agent-orientation.

But it should be noted that multi-agent system (MAS) research has already made substantial efforts to overcome this gap (see the related work section for more details). There exist approaches that view collectivities of agents (holons, teams, organizations, etc.) as first class abstractions that can collectively act on their own. We interpret these efforts as a gradual transition from classical agent-oriented concepts to rather organization-oriented or systemic ones that might ultimately result in a new paradigm of its own.

However, the treatment of recursive nesting of collectivities deserves further investigation. Existing approaches mainly focus on how to implement collective agency, for example via federative mechanisms (mediators represent a collectivity to the outside) or by providing the possibility to let collectivities play roles. We consider these as *technical* means to realize collective agency. To provide *conceptual* frameworks/"thinking models" is equally important. By conceptual, we mean qualitative guidelines and distinctions of what analytical components a collectivity has to exhibit and how they have to be arranged in order to allow a systematic transition from individual to collective agency. Research in this context has to rely on system and organization theoretical results as these disciplines have a long tradition in dealing with nested collectivities of more or less loosely coupled (individual or corporate) actors.

In this paper, we approach this research challenge by proposing an intentionally basic and generic model for collective agency. It has relevance to any system of systems engineering approach. Nevertheless, we stress the actor metaphor and thus build a particularly close link to agent-orientation. We present a model of a *system unit* that is to be applied recursively at arbitrary levels in a system of systems. A system unit is an environment for system actors and can at the same time act as a system actor in surrounding system units. In Section 2, we begin with a very abstract model that focusses on the analytical components of a system unit in terms of how it relates to its embedded system actors, manages its own configuration and relates to surrounding system units.[1] In Section 3 we model operational aspects of the former abstract model and thus derive a technical understanding of how the system unit actually functions. We discuss related work in Section 4. Section 5 summarizes our results.

As our modeling technique, we choose Petri nets [9] and especially the high-level Petri net formalism of reference nets [10]. Petri nets offer a restricted set of modeling elements, from which expressive models can be build. Reference nets additionally offer the *nets in nets* concept that allows for nested net hierarchies. Given systems of systems according to Figure 1 as our object of study, we consider nets in nets as a an inherently fitting modeling technique.

## 2 Open and Controlled System Unit: Analytical Model

We address software systems that are composed of various subsystems. At the heart of our approach lies the concept of a *system unit* that is illustrated in Fig-

[1]The model that we present in Section 2 has initially been developed in [8] for the AAMAS'2008 conference. Here, we summarize the results and present some further issues that were not covered back then (this basically concerns bridging issues in Section 2.3).

ure 1 in UML notation. A system unit is an environment for system actors. At the same time, a system unit can be a system actor of its own that has surrounding system units as environments. By recursively applying this understanding, we arrive at a hierarchy of system units, each of which is both an embedding environment and environmentally embedded.

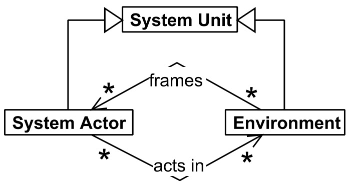


**Fig. 1.** Basic System Unit Concept

We propagate three *conceptual directions* that are to be investigated from the viewpoint of every system unit: “looking downwards/inwards” at embedded system units (internal system actors), “looking upwards/outwards” at embedding system units (surrounding environments) and “looking at ones own level” at local configurations that imprint the other two directions.

## 2.1 Open and Controlled System Unit as a Structure in Threes

We derive our model of an open and controlled system unit that is applied at *all* system levels in Figure 2. It features three different types of internal system actors that participate in three different types of system activities. Each system activity is composed of three different kinds of basic operations (addition of system units, usage/modification of system actors, removal of system actors).

The model from Figure 2 has an underlying *reference net* semantics whose operational peculiarities will be elaborated on in Section 3. In this Section, we focus on the analytical characteristics of the system unit model and are satisfied with the rather abstract model from Figure 2. It basically can be interpreted as a Petri net, but we apply some modeling short forms and assume additional semantics. (1) Activity participation of system actors is modelled by arcs. Arcs connecting to the outer circle can be associated with any type of system actor. (2) An arc labelled with a + describes a *necessary* activity participation of at least one (but possibly multiple) representatives of the associated type of system actors. An unlabeled arc describes a *contingent* participation (depending on particular instances of the associated type of system activities). (3) An arc with triangles as arrow tips indicates that the associated type of system actors enables and controls the corresponding type of system activities.

In the following, we describe the analytical aspects behind this model in more detail and demonstrate how the three types of system activities from Figure 2 exactly match with the three conceptual directions mentioned at the beginning of this section.

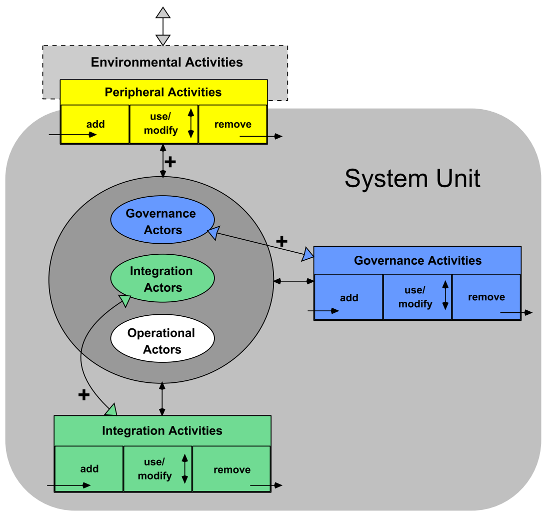


**Fig. 2.** Open and Controlled System Unit as a Structure in Threes

## 2.2 System Control

We distinguish the analytical aspects of *operation, integration* and *governance* for each system unit and classify system actors accordingly. As the three aspects are intrinsically interwoven and interdependent we do not require the three sets of system actors to be disjoint. In Section 3, we will present a concrete operational model of the interplay between the three types of actors. In this section, we focus on the intended conceptual relationships between operation, integration and governance as illustrated in Figure 3 (a) in UML notation.

Governance and integration together embody the system's control. The combination of integration and operation on the other hand represent the system's (day to day) business. The governance actors develop overall system goals and strategies and derive high-level rules that the system's business has to follow. The integration actors interpret and apply these rules to the operation of the system. They develop subgoals, plans and performance standards according to which they enable, coordinate and regulate the system's operation. Within this context, the operational actors carry out the system's primary activities and provide (low-level) operational feedback. In addition, the integration actors provide (high-level) strategic feedback to the governance actors.

With these explanations, we obtain a clearer picture of the governance and integration activities from Figure 2 and can relate them to the conceptual directions mentioned at the beginning of Section 2. Governance activities target at negotiating and transmitting system rules as well as for receiving feedback of how these rules work out. We interpret this as the direction of "looking at ones own level" as the system unit's overall configuration is under question. Integration

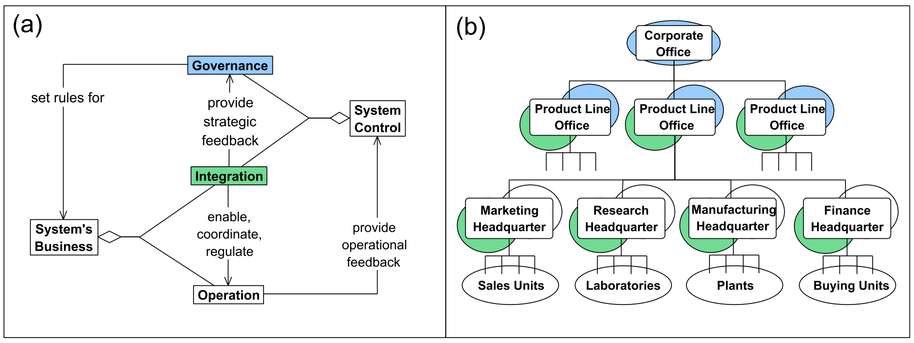


**Fig. 3.** (a) System Control Concepts, (b) Applied to the M-Form (Adapted From [11])

activities target at running the system's business in conformance to the system rules while monitoring the performance. We interpret this as the direction of "looking downwards/inwards" as here the promotion and application of the system rules to the system unit's internal functioning is addressed. Consequently, integration sits between governance and operation. While the governance actors *set* and are responsible for system rules, they do not *implement* them. This separation of concerns targets at attenuating oscillation between high-level control decisions and their effects on the system's operation.

Figure 3 (b) shows an exemplification for the case of the classical multi-divisional form (M-Form) from organizational literature (cf. [11]). We can identify the corporate office as one of the purest forms of governance actors. It is a distinguishing feature of the M-Form to free selected officials from the "tyranny" of daily operational decisions and thus allowing them to concentrate on positioning the firm, determining the proper mix of product lines and allocating resources among divisions. Strategic decisions and rules made by the general corporate office apply for the whole firm and constitute the core of its organizing principles. One of the most important and determinant directives imposed by the governance is which actors (integration) are responsible for implementing which kinds of system rules towards which other actors (operation) in which contexts. This basically determines the overall structure of the system unit.

The product line offices fulfil two analytical roles. Firstly, they are incorporated in specific governance decisions. But secondly and more important, they oversee and integrate their respective functional divisions (marketing, research, manufacturing, finance and possibly more) according to the firm's global directives. The headquarters of the functional divisions again fulfill two analytical roles. Towards their respective product line offices they act simply as operational actors, fulfilling specific functional purposes. But in addition, they oversee further units themselves and thus also act as integrators. Finally, the low-level work units of the M-Form are clearly operational actors.

So even in a strict hierarchy like the M-Form, we encounter actors with more than one analytical role. Things get even more complicated in the case of structures with additional horizontal ties.

## 2.3 Openness and Boundary Management

So far we have addressed the two conceptual directions of "looking at ones own level" and "looking inwards/downwards". Here, we look at the third direction, namely "looking upwards/upwards". Just as we have done with the other two directions, we relate the third one to one of the three type of system activities from Figure 2. In this case, we focus on peripheral activities. A system unit according to our model is a *Janus-faced* entity, embedding system actors and acting as a system actor itself in surrounding system units. Peripheral activities allow a system unit in focus to relate to system activities of surrounding system units. How these activities look like is imposed by the surrounding system units.

However, the conception of peripheral activities from Figure 2 is a simplification. As one might expect, things (at least potentially) get more complicated at the borders of a system. First of all, it is quite unlikely for most cases that a system unit exhibits control structures that only address internal functioning. Instead, boundary management is subject to careful control considerations. Consequently, in addition to having to conform to guidelines supplied by surrounding system units, peripheral activities are typically (co-) controlled by control actors of a system unit in focus as illustrated in Figure 4 (a).

This immediately leads to the distinction that was already made earlier, namely whether an activity has a governance / strategic character or an integration / daily business character. Depending on this distinction, it is governance or integration actors that enable and control a peripheral activity as illustrated in Figure 4 (b).

An further issue arises orthogonal to the one just mentioned. Analytical interpretations of actions often differ from system level to system level. For example, governance aspects from the perspective of a system unit in focus may appear as operational aspects from the perspective of a surrounding system unit. Figure 4 (c) illustrates this circumstance that arises when we apply our model of a system unit in a recursive fashion: Peripheral activities from a system unit in focus may potentially correspond / map to all kinds of activities of surrounding system units (the dashed lines model this mapping).[2]

Figure 4 (d) displays an example of different analytical interpretations. Negotiations between multiple enterprises to form a strategic alliance have a governance character from the perspective of the enterprises. From the perspective of the surrounding market however, these negotiations appear as ordinary

[2]We do not necessarily *require* that our model of a system unit is applied at all levels of a system of systems in a recursive fashion (especially in systems with a high degree of legacy structures it might not be possible in the first instance). Instead, one might apply our model selectively where it seems appropriate. In these cases, the interface to environmental system parts of course looks different from Figure 4 (c).

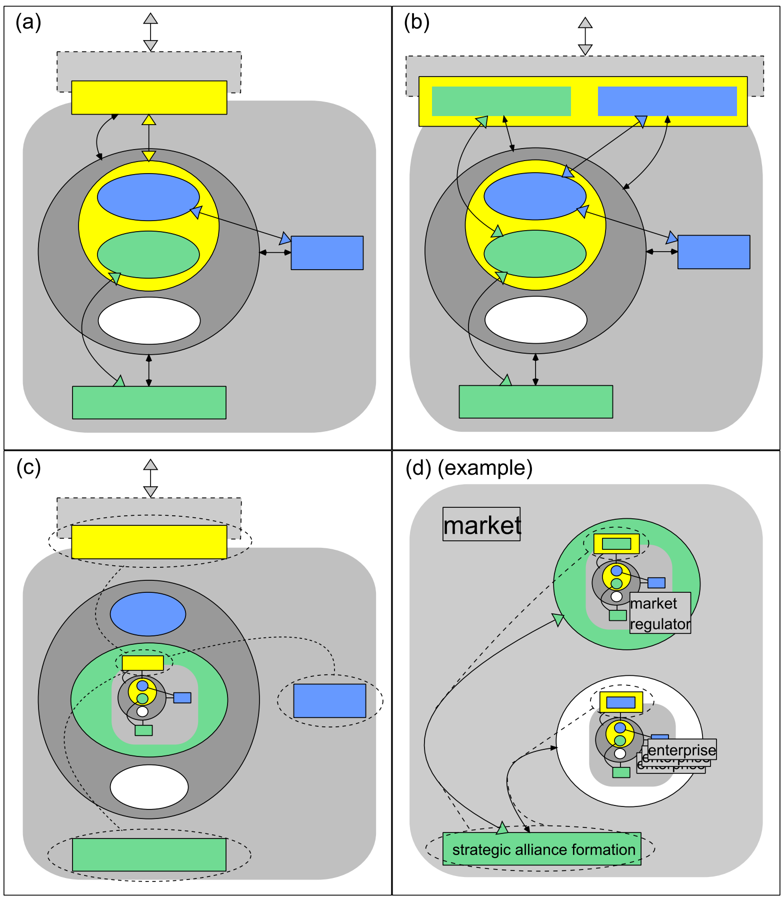


**Fig. 4.** Open and Controlled System Unit: Bridging

business-to-business interactions between operational actors that just have to be monitored by a market regulator as an integration unit.

## 3 Open and Controlled System Unit: Operational Model

The system unit model from the previous section was quite abstract and many additional (textual) explanations were necessary to clarify its intended functioning. In this section, we propose a concrete operationalization that provides (computer scientists) a technical understanding of how an open and controlled system unit according to our understanding functions. This lays the groundwork to actually realize/implement systems according to our philosophy.

Due to space limitations, we will not be able to cover all aspects of our operational model. For instance, we will not address the addition or removal of system actors. Instead, we focus on how existing system actors invoke and participate in system activities.

As our modeling technique, we choose reference nets [10]. They provide an elegant way to transition from rather abstract to more and more detailed models and in addition allow an effective way to model dynamic multi-level composition of components. We give a short introduction into reference net modeling.

## 3.1 Reference Net Modeling

We assume that the reader is at least vaguely familiar with Petri nets in general as well as “ordinary” colored Petri nets (see e.g. [9]). We will give a brief overview of the specific extensions that reference nets [10] exhibit. Reference nets implement the *nets within nets* concept [12] where a surrounding net - the *system net* - can have nets as tokens - the *object nets*. As hierarchies of net within net relationships are allowed, the denomination of system or object net depends on the beholder’s viewpoint. Reference semantics is applied, thus net tokens are *references* to *net instances*. Figure 5 shows a simple example. We have an assembly line as a system net. Boxes as object nets are moved along the line in order to be filled with items according to some description.

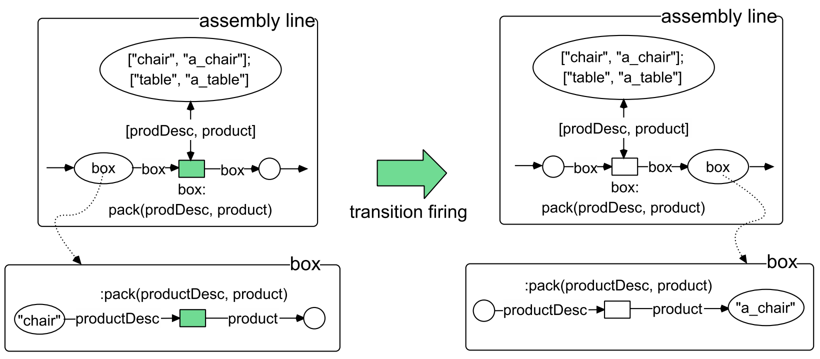


**Fig. 5.** Operational Semantics of Reference Nets

To facilitate communication between net instances, *synchronous channels* permit a fusion of two transitions at a time for the duration of one firing occurrence. A channel is identified by its name and its arguments. Channels are directed, exactly one of the two transitions (the one with a *downlink*) indicates the net instance in which the counterpart of the channel (a transition with the corresponding *uplink*) is located. However, information flow between the fused transitions is bi-directional and achieved by *unification*. In our example, we have the channel :pack() which is directed from the assembly line (downlink) down to the box (uplink). Information flow is bi-directional. The :pack() channel has two parameters and a unifying binding for corresponding parameter positions on both sides of the channel has to be found. The product description (productDesc) is supplied by the box while the product itself (product) is supplied by the assembly line (it is looked up from a mapping between product descriptions and products).

As for most other net formalisms there exist tools for the simulation of reference nets. The Renew tool [13] additionally allows for (mostly) arbitrary JAVA inscriptions to transitions. This allows for a powerful approach of "implementation through specification". In order to transform an abstract net model into a specific prototype it is in most cases sufficient to add combined channel/JAVA inscriptions to transitions and to add certain auxiliary net elements.

## 3.2 Embedding

We begin by operationalizing the open Janus-faced character of our system unit model from Section 2.

**Abstract Operational Model** A system unit according to our understanding embeds internal system actors and can at the same time act as a system actor at higher system levels. Figure 6 shows a simple reference net operationalization of this understanding. According to the terminology introduced in the previous subsection, the figure features a system unit as a system net and system actors as well as system activities as embedded object nets.

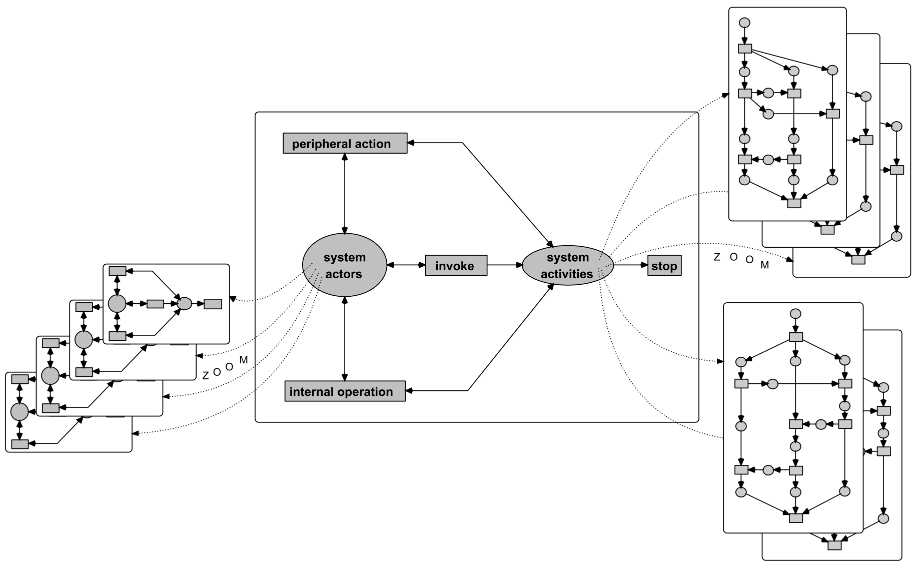


**Fig. 6.** Open System Unit: Operational Model

Consequently, we can identify the two concepts of system actors and system activities from the analytical model of Figure 2. The figure illustrates the recursive character of our model as system actor nets are of the same structure as the system unit in focus. For the case of activities, we apply a modeling approach in conformance with the UML [14]: *An activity models part of a system's behavior by describing how elementary behavioral elements (i.e.* actions*) are combined to more complex behaviors by means of control and data flows* (see below for a more detailed example of activity modeling).

Actions have to carried out by system actors. Consequently, actor and activity nets have to be synchronized for action execution (transitions internal operation and peripheral action). In addition, system actors can invoke new system activities (transition invoke).

By peripheral actions, system actors act *on behalf* of a system unit in focus and thus allow it to appear as a system actor in surrounding system units. Thus, depending on action parameters, the peripheral action transition of a system unit in focus can be synchronized with any of the three action transitions (for invocation, internal actions and peripheral actions) of surrounding system units.

Figure 6 lacks operationalization details in order to make it fully executable. We will not cover the details comprehensively but focus on the two action transitions for internal operations and peripheral actions in the following paragraph.

**Operational Details** At this point, the model from Figure 6 lacks most of the analytical distinctions from Section 2. But we can distinguish internal activities that consist only of internal actions from peripheral activities that also include peripheral actions. Operationalization of internal as well as peripheral actions deserves further investigation, so Figure 7 provides additional details.

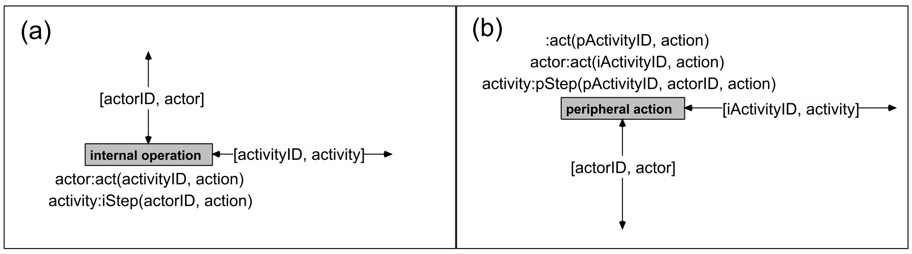

(a)
[actorID, actor]
internal operation
[activityID, activity]
actor:act(activityID, action)
activity:iStep(actorID, action)
(b)
:act(pActivityID, action)
actor:act(iActivityID, action)
activity:pStep(pActivityID, actorID, action)
peripheral action
[iActivityID, activity]
[actorID, actor]

**Fig. 7.** Operationalization Details for Internal and Peripheral Actions

For each internal action, the system unit net synchronizes an actor net with an activity net. Synchronization takes place via the channels :act() and :iStep(). The underlying semantics is of course that an actor carries out an action in order to realize an activity step. As there exist typically multiple actors and activities at the same time, we have to make sure that the right actors and activities come together for an action execution. For this reason, actors and activities denote each other mutually via identifiers. If these match, synchronization is possible.

In addition, the action parameters of both channels have to unify. Note that the action parameter is actually a complex data structure itself that includes an action type, an action description and action data. Which of these parts are supplied by actor or activity net (or both) for unification depends on the action's context.

The peripheral action type allows a system unit to appear as a system actor of its own that is embedded in a surrounding system unit. For a system unit in focus, a peripheral action is carried out by an internal system actor, via an :act() channel between the system unit in focus and the internal actor. But from the perspective of a surrounding system unit, the action is carried out by the system unit in focus as a whole, via an :act() channel between the system unit in focus and the surrounding system unit. Thus, we arrive at a technical understanding of *collective/corporate* action. As an additional aspect, peripheral actions are not only associated with an internal system activity (via the identifier iActivityID) but also with a system activity of the surrounding system unit (via the identifier pActivityID). So the channel pStep:() has three arguments instead of the two ones for the :iStep() channel.

**Example** Figure 8 shows a simple sample scenario with three system levels. We have two system units (market and prod firm) according to our model. For the system units that represent the "leaves" in this scenario we make no further assumptions other than that they offer :act() channels as interfaces.

Activities are composed of net components for invocation and stopping (diamonds), internal actions (downward-pointing triangles) and peripheral actions (upwards-pointing triangles). In this paper, we will not deepen on how to model activities. We just assume a close connection to UML (AUML) interaction diagrams (c.f. [15]). Transitions model actions and places connect actions in order to form *life lines* for activity *roles* and *message exchanges* between activity roles. Whenever a place connects two transitions whose associated actions are to be carried out by the same actor in the context of the activity, the transitions are drawn vertically on the life line of the corresponding role. Whenever a place connects transitions whose associated actions are to be carried out by different actors, the place models a message exchange between the corresponding roles which is drawn horizontally. In this respect, one might model activities as *distributed workflow nets / inter-organizational workflows* [16, 17].

The system unit prod firm embeds internal system actors and acts itself as a system actor on the market. The market activity deliver-receive (including the roles prod-firm and receiver) is an internal activity as it only consists of internal actions. The activity produce-ship (including the roles producer and shipper) of the producer firm on the other hand is a peripheral one as it includes peripheral actions. [3] In the figure, we have just denoted the :iStep()/:pStep() channels for activity steps and left out the associated parameters. We have highlighted the involved transitions of two actions respectively. For once, we have the production

[3]One might apply a constraint that only allows peripheral actions in peripheral activities. Additional internal actions that are required would have to be sourced out to supplementary internal activities.

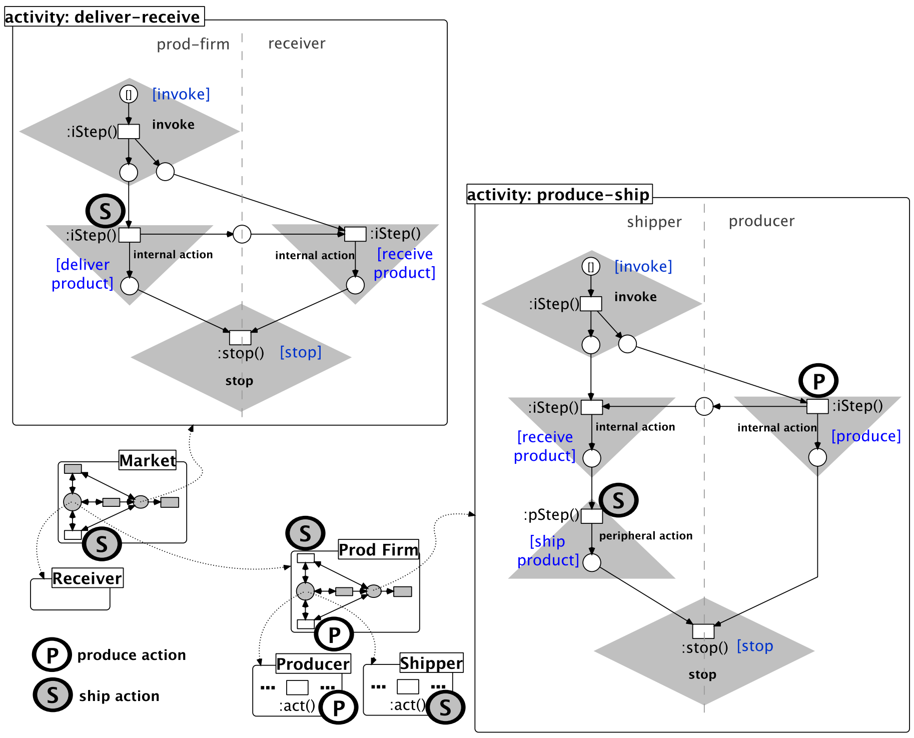


**Fig. 8.** Sample Scenario

action that is internal to the producer firm. Then we have the ship action that is a peripheral action from the perspective of the producer firm and an internal action from the perspective of the market. It draws a connection between the two associated activities.

## 3.3 System Control

So far we have imposed no restrictions on which system actors may initiate which kinds of system activities and which additional system actors may participate in those activities. In this section, we extend the Janus-faced model from the previous section in order to better account for the analytical distinctions from Section 2.

**Abstract Operational Model** Figure 9 shows the model of an open system unit with a two-level control structure. We make the same distinctions concerning system actors (operation, integration, governance) and system activities (integration, governance) that were already made in Section 2. We illustrate this by indicating a coloring of the places for system actors and activities using the same colors as in Figure 2. Besides coloring the places for system actors and activities, we have colored transitions according to the interpretation, which kinds of system actors carry out the corresponding actions. We also explicitly model the concept of *system rules* that were already referred to in Section 2. This supports developing an operational model for the analytical descriptions from Section 2.

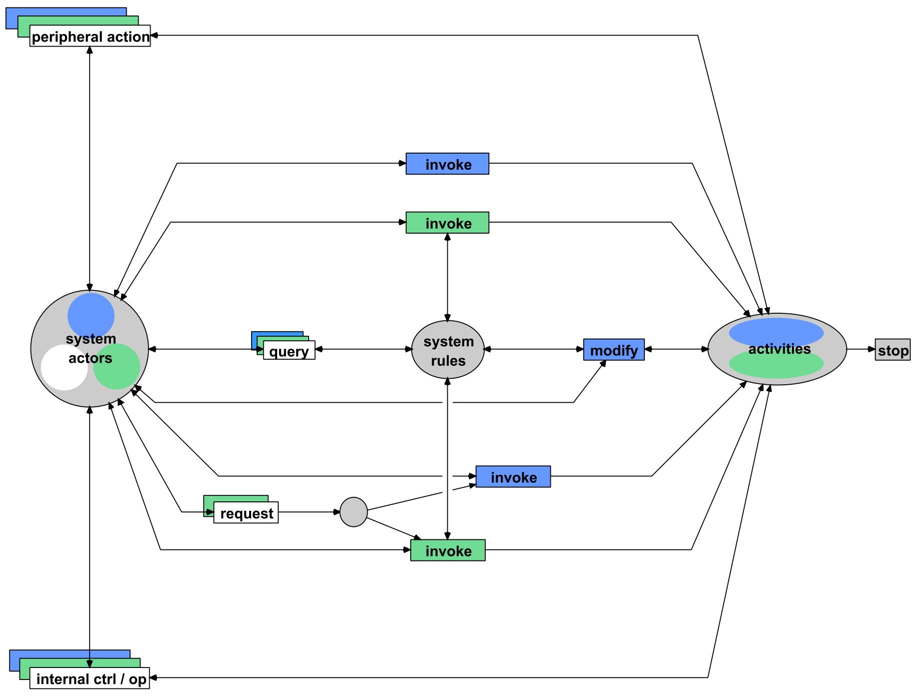


**Fig. 9.** Open and Controlled System Unit: Operational Model

In order to deal with the topic of system rules comprehensively one would have to deal with several subtle aspects. In particular, there exist various kinds of rules with quite different characteristics concerning their genesis, their effects and possibilities of being enforced (c.f. e.g. Scott's characterization [18] of *regulative, normative* and *cultural-cognitive* rules in institutions/organizations). For our current purpose, we take a less sophisticated approach and apply a rather plain handling and usage of system rules. We refer to rules that are explicitly formulated (so to say "written down") and can be consulted.[4] Basically, we require a consistent, unambiguous ruleset that can be consulted to check whether

[4]These are the kind of rules we deal with explicitly. We do not prohibit that there exist further kinds of rules, perhaps less formal ones that only take effect when interpreted and backed up by certain system actors. But for the moment we do not address them systematically.

the invocation of a given system activity is admissible. This simple approach still allows for quite expressive system specifications. In particular, it is three main types of rules that can be formulated.

1. Who is allowed to invoke and individually shape (within given boundaries) which kinds of activities?
2. Who is allowed/required to participate into which kinds of activities?
3. What form/characteristics do certain kinds of activities have to take on?

The first two points might give rise to sophisticated role/position/(group) models that are for example quite common in MAS engineering. (Organization-oriented) MAS specifications often take on a form where rights and responsibilities of agents are formulated depending on which roles/positions (in which groups) they occupy. The first and the third point give rise to more or less standardization concerning system activities. Constraints on form/characteristics of activities might for example be a requirement that they have to be well-formed workflows according to [19] or a declaration of interaction *landmarks* that an activity has to pass like applied in [20]. System rules that grant system actors a high level of freedom in shaping activities individually lead to rather *organic* system behavior. A high level of standardization on the other hand leads to rather *bureaucratic* system behavior. At one extreme, we might for example only have a set of fixed activity patterns from which to choose.

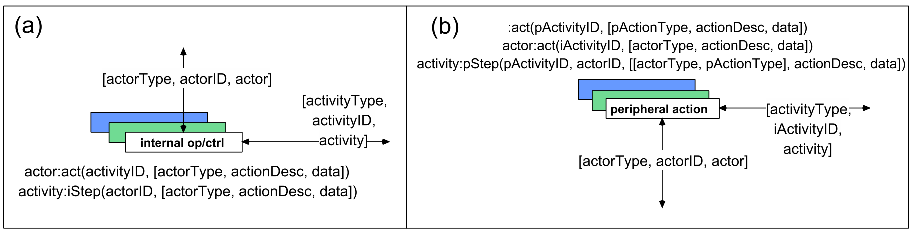
As can be seen in Figure 9, only integration and governance actors may invoke system activities, either pro-actively or reactively in response to a request by other system actors. The model exhibits separation of concerns with respect to *setting* and *applying* system rules as it was already propagated in Section 2. System rules are set and modified by governance actors in the course of governance activities. Any access to system rules other than that is restricted to be read-only. Integration actors may invoke integration activities only if they are in adherence to system rules. Governance actors on the other hand are allowed to invoke governance activities without reference to system rules.[5] Consequently, we have the desired effect of integration as an intermediary between governance and operation of a system unit.

We still deal with an open system model and recursive system of systems structures. Both controlling and operational system actors can carry out peripheral actions. We obtain a modular comprehension of overall system control. Each system unit is *autonomous* and may establish its own control mechanisms and principles. However, (most) system units are not *autarch*, they depend on being able/allowed to participate in activities of surrounding system units. Consequently, control aspects have to be coordinated/negotiated among system units. We encounter a mechanism of *mutual constitution* at each system level: Each

[5]However, for many systems such a bypass around system rules might never be acceptable. In these cases, the model would have to be extended so that governance and integration activity invocation are treated the same way. This basically means that different governance activities offer different levels of authority when it comes to rule amendment.

system unit as a whole *imprints* its embedded system actors and is at the same time *reconstructed* by them.

**Operational Details** It is worth to have a second look at the operational details of internal as well as peripheral actions in Figure 10 now that more analytical distinctions have been introduced.



(a)
[actorType, actorID, actor]
[activityType, activityID, activity]
internal op/ctrl
actor:act(activityID, [actorType, actionDesc, data])
activity:iStep(actorID, [actorType, actionDesc, data])
(b)
:act(pActivityID, [pActionType, actionDesc, data])
actor:act(iActivityID, [actorType, actionDesc, data])
activity:pStep(pActivityID, actorID, [[actorType, pActionType], actionDesc, data])
peripheral action
[activityType, iActivityID, activity]
[actorType, actorID, actor]

**Fig. 10.** Operationalization Details for Internal and Peripheral Actions

System actors are no longer only associated with an identifier but in addition with an explicit actor type (although we pointed out in Section 2 that an actor might have multiple types). The same holds for system activities.

In Section 3.2 we had already mentioned that the action parameter was actually a complex one. Here, we make use of this fact. In our reference net notation, an action is a tuple [actionType, actionDesc, data] consisting of an action type, an action description and additional data. The action type may take on the values "operation", "integration" or "governance" and in the case for internal actions in Figure 10 (a), we see that it has to match the type of the actor that carries out the action. For the case of peripheral actions in Figure 10 (b), things get a bit more complicated. In Section 2, we described that the attachment of analytical interpretations to actions differs from system level to system level. So the action type of a peripheral action is itself a tuple, consisting of its analytical interpretation from the system unit in focus (which has to match the type of the internal system actor that carries out the action) and its analytical interpretation at the level of the surrounding system unit.

According to our modeling philosophy, the system rules again are represented by a reference net and Figure 11 shows the channel interface. The channels :query(), :approve() and :modify() are used for ruleset queries, for requesting the approval of system activity invocations by integration actors and for rule amendments by governance actors respectively. We leave out further operationalization details as especially activity invocation encompasses several technical aspects that we do not want to cover in this paper.

Although we demand the system rules to be represented as a reference net, this is no heavy restriction on how to actually formulate rules and apply reasoning mechanisms on them. For example, it is typically preferred to formulate rules in a declarative way. With reference nets it is easily possible to use the ruleset net just as a wrapper for any kind of (JAVA-supported) declarative rule engine (e.g. Prolog or JESS).

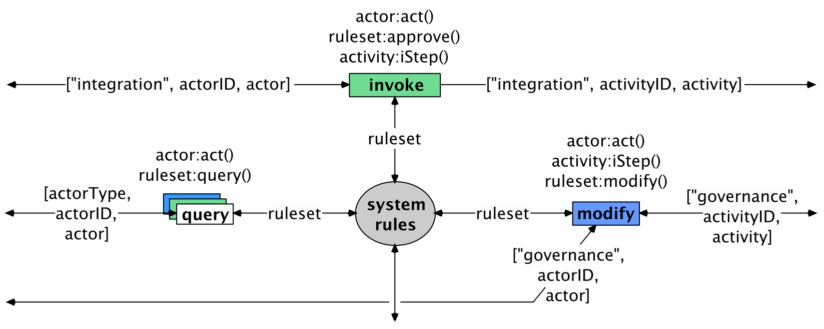


**Fig. 11.** Operationalization Details for System Rules

## 4 Related Work

As our modelling approach rests on quite basic ideas and provides concepts as well as mechanisms that are neutral to specific technologies, the range of related work is very wide. In some way, any modelling/engineering approach for complex systems might be considered related and we have provided several references throughout the paper. In this section, we restrict ourselves to address two fields of related research explicitly. The first one is general system and organization theory as it provided much inspirational input. The second one is MAS research as it basically was the starting point of our research and we view our results as a conceptual extension to classical agent-orientation.

**General System and Organization Theory** General system and organization theory have a long tradition in studying openness and control of systems.

Concerning open systems, we have first of all adopted the common view from [18] (among others): An open systems (like for instance an organization) relies on services and resources of other systems but at the same time constructs and continuously reconstructs its boundaries across which it relates to its environment. Because of the importance of the environment for the studying of open systems, it is not enough to characterize it as simply "everything else". Instead, our approach is consistent with the systemic perspective that the environment does not only consist of other systems (that in turn view the first system as a part of their respective environments) but is itself a system (c.f. e.g. [21, 22]). Each open system has an environment exactly because of its participation in a greater whole and its characteristics are a function of its fit into this greater context. For sufficiently complex systems, we obtain a hierarchy of systems embedding other systems. Simon [23] points out that hierarchy in this sense of clustering is a fundamental feature of *all* complex systems. Each particular system is localized both in terms of its embedded systems and in terms of the systems by whom it itself is embedded. As in our case, this perspective has frequently been applied in a recursive manner, with similar systems at each level (c.f. the concept a *Holon* [24] or the *Viable System Model* [25]).

In the case of system control, we have basically drawn on inspiration from cybernetics [26] as a sub-field of general systems theory. It propagates the analytical separation of a controlled part and a controlling part of a system. The variant to carry out a further separation of the controlling part in two that we have applied is quite popular (c.f. [27, 28]). Ashby [29] characterizes this distinction as separating the handling of disturbances "in degree" (applying existing decision rules to the operation of a system) from the handling of disturbances "in kind" (determining whether it is necessary to redefine the rules) and it is closely related to what Argyris [30] labels *double-loop learning*. According to Scott [18], this principle provides a very powerful and general model of control that is widely employed in organizational settings.

The rationale behind the deep rooting of our approach in system and organization theory is to learn from the adaptivity, robustness, scalability and reflexiveness of their respective objects of study (especially social systems) and to translate their building principles in effective information technology. Thus, we have not only combined and integrated the inspirations concerning openness and system control into our model of a system unit. We have also proposed an operational model that provides a technical understanding of the theoretical concepts.

**Multi-Agent System Engineering** We have already stressed the relation to MAS research as the starting point of our research in the introduction of this paper. MAS technology already provides numerous technical realization means for recursive nesting and collective agency in agent-based systems.

For example, organization-oriented approaches often advocate the concept of groups as a further decomposition means. While groups are mostly just considered as contexts for individual agent behavior, several approaches (e.g. holons [31], JACKTeams [32], socially-constructed agents [7], meta model for MAS with organizations as specialized agents [33]) go further and regard the grouping concept as a potential basis for a recursive decomposition means, resembling the object-oriented one. Our approach can be considered as orthogonal to these efforts. We provide a conceptual model for collective agency. It consists of both an analytical and operational part, but does not prescribe any particular realization means.

In addition, we have intentionally chosen a restricted set of basic concepts (operation, integration, governance) that we relate horizontally as well as vertically in a generic way. To account for these concepts while using current agent-oriented technology (modelling approaches, methodologies, middleware) is straightforward. For example, in Section 3.3 we have already drawn the connection between our model and common MAS engineering approaches that rely on specifications in terms of roles, positions and groups. Even our concept of utilizing system rules/specifications that are "written down" and can be consulted at run time is very common in MAS engineering, for example in the context of organization-oriented middleware approaches [5].

To put it differently, while we see our model not necessarily coupled to agent-orientation, we ascribe agent-oriented technology the potential of providing an ideal vehicle for the deployment of systems of systems according to our approach. In the opposite direction, we see our model as contributing to agent-oriented efforts that seek to close the gap between classical agent-oriented thinking (flat decomposition) and the need for hierarchic decomposition mechanisms that still respect the actor perspective.

## 5 Conclusion

In this paper, we have presented a modular approach to comprehend systems of systems by means of composing modular system units. Each system unit may be regarded under a platform perspective, where it offers technical and strategic frames for its inhabitants. Furthermore the same unit may be regarded under a corporate agency perspective, where it collectively acts as a holistic entity in the context of a higher-level system unit. Specifically, we have set collective agency in relation to the control structures at each level. This provides a conceptual basis to systematically study and implement different modes of coupling, both horizontally and vertically.

In [8] we have applied this model to study different levels of what we termed a reference architecture for *multi-organization system*. We have qualitatively distinguished departments, organizations, organizational fields and the society as iteratively embedded specific types of system units according to the model from Figure 2. In [34] we have pursued this research by studying the fit between the different levels of the reference architecture and different engineering approaches from MAS field.

In this paper, we chose another focus instead of applying our model in an analytical way. We have provided a concrete operational basis for the formerly abstract concepts. The underlying purpose is somewhat twofold. The main purpose is still to provide a *reference model* for the comprehension of and thinking about complex systems. Here, we have supplied a technical understanding of our former abstract concepts. This helps to get a clearer picture of how systems of systems according to our thinking model function. In addition, such a technical understanding is necessary in order to get a grasp of how to actually build software systems according to our approach. As for concrete implementation means, we assume no strong restrictions. But we have pointed out our belief that agent-oriented technology has the potential to provide an ideally suited deployment vehicle.

Besides such conceptual guidelines for comprehending and building systems of systems, our reference net-based notation opens up another possibility. Reference nets allow for an “implementation through specification” approach as explained in Subsection 3.1. For example, this approach has been proven successful in the case of the conceptual multi-agent system model Mulan [35] that has been refined/extended into a full-fledged FIPA-compliant multi-agent framework (Capa [36]) based on reference nets. For the time being, our operational model presented in this paper has not yet reached such a mature state. It is fully functional as a prototype and can be considered as a proof of concepts. But for specific usage, a reference model like ours has to be complemented by modeling languages, tools and methodologies that really make it applicable in the first place. For the time being, we have basically just a developed a reference model. In future work, we plan to extend it into a comprehensive software engineering approach.

## References

1. Maier, M.: Architecturing principles for systems-of-systems. Systems Engineering **1**(4) (1999) 267–284
2. Lankes, J., Matthes, F., Wittenburg, A.: Softwarekartographie: Systematische Darstellung von Anwendungslandschaften. Wirtschaftsinformatik 2005 (2005)
3. Northrop, L.: Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon (2006)
4. Hess, A., Humm, B., Voss, M., Engels, G.: Structuring software cities - a multidimensional approach. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). (2007) 122–129
5. Boissier, O., Hübner, J., Sichman, J.S.: Organization oriented programming: From closed to open organizations. In O'Hare, G., Ricci, A., O'Grady, M., Dikenelli, O., eds.: Engineering Societies in the Agents World VII. Volume 4457 of LNCS., Springer, Heidelberg (2007) 86–105
6. Jennings, N.: On agent-based software engineering. Artificial Intelligence **177**(2) (2000) 277–296
7. Boella, G., van der Torre, L.: Organizations as socially-constructed agents in the agent-oriented paradigm. In Gleizes, M.P., Omicini, A., Zambonelli, F., eds.: Engineering Societies in the Agents World V. Volume 3451 of Lecture Notes in Computer Science., Springer Verlag (2005) 1–13
8. Wester-Ebbinghaus, M., Moldt, D.: Structure in threes: Modelling organization-oriented software architectures built upon multi-agent systems. In: Proceedings of the 7th International Conference an Autonomous Agents and Multi-Agent Systems (AAMAS'2008). (2008) 1307–1311
9. Girault, C., Valk, R.: Petri nets for systems engineering: a guide to modelling, verification and applications. Springer Verlag (2003)
10. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
11. Chandler, A.: Strategy and Structure: Chapters in the History of the American Industrial Enterprise, publisher=Cambridge, MA: MIT Press, year=1962
12. Valk, R.: Petri nets as token objects: An introduction to elementary object nets. In Desel, J., Silva, M., eds.: Application and Theory of Petri Nets. Volume 2001. Springer Verlag (1998)
13. Kummer, O., Wienberg, F., Duvigneau, M.: Renew – the Reference Net Workshop. Available at: http://www.renew.de/ (2006) Release 2.1.
14. Bock, C.: UML 2 activity and action models. Journal of Object Technology **2**(5) (2003) 43–53
15. Cabac, L., Moldt, D., Rölke, H.: A proposal for structuring Petri net-based agent interaction protocols. In v. d. Aalst, W., Best, E., eds.: International Conference on Application and Theory of Petri Nets 2003. Volume 2679 of Lecture Notes in Computer Science., Springer-Verlag (2003) 102–120

16. Köhler-Bußmeier, M., Moldt, D., Wester-Ebbinghaus, M.: A formal model for organisational structures behind process-aware information systems. Special issue of ToPNoC on Concurrency in Process-Aware Information Systems, to appear March 2009 (2009)
17. van der Aalst, W.: Interorganizational workflows. Systems Analysis - Modelling - Simulation **34**(3) (1999) 335–367
18. Scott, W.R.: Organizations: Rational, Natural and Open Systems. Prentice Hall (2003)
19. van der Aalst, W.: Verification of workflow nets. In: Application and Theory of Petri Nets 1997. Volume 1248 of Lecture Notes in Computer Science., Springer Verlag (1997) 407–426
20. Vasquez-Salceda, J., Dignum, V., Dignum, F.: Organizing multiagent systems. Autonomous Agents and Multi-Agent Systems **11** (2005) 307–360
21. Parsons, T.: Structure and Process in Modern Societies. Glencoe, IL.: Free Press (1960)
22. Luhmann, N.: Soziale Systeme. Frankfurt a. M.: Suhrkamp (1984)
23. Simon, H.: The architecture of complexity. In: Proceedings of the American Philosophical Society. Volume 106. (1962) 467–482
24. Koestler, A.: The Ghost in the Machine. Henry Regnery Co. (1967)
25. Beer, S.: The Heart of the Enterprise. New York: Wiley and Sons (1979)
26. Wiener, N.: Cybernetics. New York: Wiley and Sons (1948)
27. Swinth, R.: Organizational Systems for Management: Designing, Planning and Implementation. Columbus, OH.: Grid (1974)
28. Herring, C.: Viable software: The intelligent control paradigm for adaptable and adaptive architectures. Dissertation, Uiversity of Queensland, Department of Information Technology and Electrical Engineering (2002)
29. Ashby, R.: Design for a Brain. Ney York: Wiley and Sons (1960)
30. Argyris, C., Schön, D.: Organizational Learning: A Theory of Action Perspective. Reading, Mass.: Addison Wesley (1978)
31. Fischer, K., Schillo, M., Siekmann, J.: Holonic multiagent systems: A foundation for the organization of multiagent systems. In: Holonic and Multi-Agent Systems for Manufacturing, First International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS). Volume 2744 of Lecture Notes in Computer Science., Springer Verlag (2003) 71–80
32. AOS-Group: Jack intelligent agents team manual. Available at: http://www.aosgrp.com/ documentation/jack/JACK_Teams_Manual_WEB/index.html (2009)
33. Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems **18**(2) (2008)
34. Wester-Ebbinghaus, M., Köhler-Bußmeier, M., Moldt, D.: From multi-agent to multi-organization systems: Utilizing middleware approaches. In Artikis, A., Picard, G., Vercouter, L., eds.: International Workshop Engineering Societies in the Agents World (ESAW 08). (2008)
35. Köhler, M., Moldt, D., Rölke, H.: Modelling the structure and behaviour of Petri net agents. In Colom, J., Koutny, M., eds.: Application and Theory of Petri Nets 2001. Volume 2075 of LNCS., Springer Verlag (2001) 224–241
36. Duvigneau, M., Moldt, D., Rölke, H.: Concurrent architecture for a multi-agent platform. In Giunchiglia, F., Odell, J., Weiß, G., eds.: Agent-Oriented Software Engineering III. Third International Workshop, AOSE 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions. Volume 2585 of Lecture Notes in Computer Science. Springer-Verlag, Heidelberg (2003) 59–72

# From AGR to MASQ: Understanding organizations from a multi-agent point of view (Invited Talk)

Jacques Ferber

ferber@lirmm.fr
LIRMM – University of Montpellier II, 161 rue Ada,
34592 Cedex 5, Montpellier, France

Multi-agent systems (MAS) have taken their main ideas from human and animal societies. One of the last concept that has unfolded in the MAS domain is the concept of organiza-tion, from which OCMAS or (Organization Centered Multi-Agent Systems) have been built. The AGR family (Ferber & Gutknecht, 1998; Ferber, Gutknecht & Michel, 2004) has been one of the first to give a simple and detailed account of what an OCMAS should be, and has belle the conceptual basis for the MadKit platform (MadKit 2004). The Moise+ has also shown its importance in the field by providing a general framework for groups, roles, goal-driven agents and norms (Hbner, Sichman & Boissier, 2002). In this talk, we will present several models (AGR, AGRE, MOISE+, etc..) and compare their way of addressing the complexity of organizations in an operational and tractable way from a computer science point of view. We will also explore the work on institutions using Searle theory (Searle, 1995) and show how this can help the understanding of organizations (Dastani et al. 2009). Then, we will present an integrated model of MAS, using the MASQ generic model (Ferber et al. 2009, Stratulat et al. 2009), which constitute an abstraction of the various aspects of a MAS, generalizing the AGR and AGRE approach and incorporating the institutional work of Searle. This model is based on a 4-quadrant conceptual framework, where the analysis and design of a system is performed along two axes: an interior/exterior dimension and an individual/collective dimension. We will give a conceptual definition of this approach and we will show that it is possible of applying it to practical problems in the computer science fields. We will give some ideas about its use as a methodological tool and also how this model could be used to represent human organizations, through the various diagrams and notations that are proposed.

## References

Dastani M, Tinnemeier N. A.M, J.-J. Ch. Meyer, A Programming Language for Normative Multi-Agent Systems in Virginia Dignum (Ed), Multi-agent Systems: Semantics and Dynamics of Organiza-tional Models. IGI.

Ferber, J. & Gutknecht, O. (1998). A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. Proceedings of the 3rd International Conference on Multi Agent Systems (ICMAS?98) (pp. 128-135). IEEE Computer Society.

Ferber, J. (1999). Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Addison-Wesley.

Ferber, J., Gutknecht, O., & Michel, F. (2004). From Agents to Organizations: An Organizational View of Multi-Agent Systems. In P. Giorgini, J. Mller, J.Odell (Eds.), Agent-Oriented Software Engineering (AOSE) IV (pp. 214-230). LNCS 2935, Springer.

Ferber, J., Michel, F., & Baez, J. (2005). AGRE: Integrating Environments with Organizations. In D. Weyns, V. D. Parunak, F. Michel (Eds). Environments for Multi-Agent Systems (pp. 48-56). LNAI 3374, Springer.

Ferber J., Stratulat T., and Tranier J. (2009) Towards an Integral Approach of Organizations: the MASQ approach, inMulti-Agent Systems in Virginia Dignum (Ed), Multi-agent Systems: Semantics and Dynamics of Organizational Models. IGI.

FIPA (2005). The Foundation of Intelligent Physical Agents. www.fipa.org

Hbner, J., Sichman, J. & Boissier, O. (2002). A model for the structural, functional, and deontic speci- fication of organizations in multiagent systems. In Bittencourt, G. & Ramalho, G. L. (Eds.), Advances in Artificial Intelligence, 16th Brazilian Symposium on AI, SBIA'02, LNAI 2507 (pp. 118-128). Berlin, Springer.

MadKit (2004). A Multi-Agent Development Kit. www.madkit.net

Searle, J.R. (1995). The Construction of Social Reality. Free Press.

Stratulat T., Ferber J., Tranier J. MASQ - Towards an Integral Approach of Agent-Based Interaction. AAMAS 2009.