Semantic Mappings between Service, Component and Agent Models

Nour Alhouda Aboud
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Nour-alhouda.Aboud@univ-pau.fr

Eric Cariou
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Eric.Cariou@univ-pau.fr

Eric Gouardères
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Eric.Gouarderes@univ-pau.fr

Philippe Aniorté
LIUPPA / Université de Pau
2 Allée du parc Montaury
64460 ANGLET, France
aniorte@iutbayonne.univ-pau.fr

ABSTRACT

Regarding the design and the development of distributed applications, service, component and agent oriented approaches provide their own interests and characteristics. Most of current applications are designed according to a single approach but it would be interesting to use these approaches simultaneously to provide a more efficient paradigm for developing distributed applications. Our goal is to integrate these three approaches by focusing on the concepts of service and interaction as key points, notably regarding cooperation between agents and components. The service is the business abstraction of a component or agent and it represents a pivot to support its interactions. We presented previously our service, component and agent models for this purpose. In this paper, we establish the semantic mapping rules between the concepts of each of these domains to be able to move on from one model to another.

Categories and Subject Descriptors

D.2 [**Software Architectures**]: Languages

General Terms

Design

Keywords

Component, agent, service, semantic mappings.

CBSE'12, June 26–28, 2012, Bertinoro, Italy.

1. INTRODUCTION

Regarding the design and the development of distributed applications, service, component and agent oriented approaches provide their own interests and characteristics. Services provide the interests of abstraction and large-scale interoperability. Abstraction in the sense that a service specifies a functional element without specifying how this element is implemented (by a component or an agent for example). Components are a robust approach based on the composition and the reusing of clearly defined elements through their interfaces. Agents[1] are elements with dynamic goal-directed behaviors and using high-level interactions with the other agents forming an application. Figure 1 represents the main advantages and complementary aspects of these approaches. Unfortunately, these specific advantages of each one are not shared between all of these three approaches. For example, [4, 9] highlight the lack of reusability on the agent side and [3] highlights the lack of reasoning ability and dynamicity on the component side.

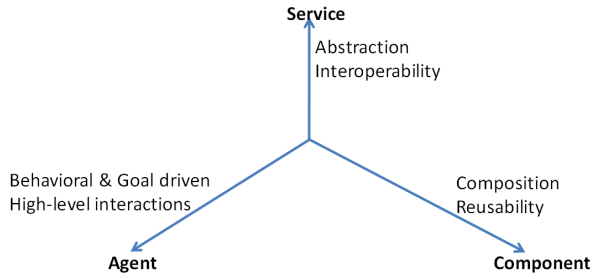

Figure 1: Relations between services / agents / components

However, it would be interesting to benefit from all these characteristics within a same application. Then, the aim

[1]In this paper, when we talk about agents, we refer systematically to the Organizational MultiAgent Systems (OMAS).

of our work is to integrate these three approaches. We focus on the concepts of service and interaction as key points, notably regarding cooperation between agents and components. The service is the business abstraction of a component or of an agent and it supports interoperability between them by considering that a service can be implemented either by an agent or by a component. To serve this purpose, we presented earlier in [2] our service, component and agent models. In this paper, we define the semantic mappings between the concepts of each of these models to move on from one domain to another. With a Model Driven Engineering (MDE) approach, these models are DSMLs (*Domain Specific Modeling Languages*) and the semantic mappings will be implemented by model transformations.

The rest of this paper is organized as follows. Section 2 reminds our design process and its three models of service, component and agent, and section 3 defines the semantic mappings between these three models. Then, we conclude.

2. REMINDER OF SERVICE, COMPONENT AND AGENT MODELS

In order to well understand the contribution of this paper, we remind firstly in this section elements we already presented in [2].

2.1 Design process

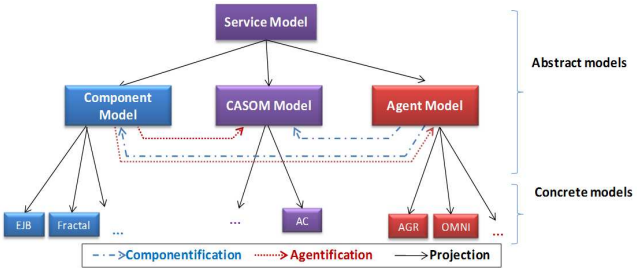

Figure 2: Integration process of service, component and agent approaches

Our general proposition for the integration of service, component and agent approaches is based on a design process described in figure 2. The process contains four main models (four DSMLs in a MDE approach). At the most abstract level, we find our service model that describes an application only through interacting services. Therefore, it focuses on the business specification of an application. At a lower level, the same application is specified by defining the concrete elements implementing these services: we use either only components, only agents or a mix of them. For this purpose, we use respectively the models of component, agent or CASOM (for *Component Agent Service Oriented Model*) which incorporates the concepts of agent and component. Our component and agent models as well as CASOM are general and abstract. It is possible to project their specification onto a concrete technological model, like EJB[2] or Fractal [5] for components, AGR [7] or OMNI [11] for agents or AC [8] for a mixed component/agent approach. The process can be used in several ways: by a top-down approach, as mentioned above where we define at the abstract level the interacting services of an application, then we project these services towards components, agents or a mix of them before projecting this new specification onto a concrete technological platform. We can carry-out a reverse approach (a bottom-up one), that is processing reverse engineering. Finally, we have a middle-out (horizontal) approach where we transform existing specifications of agents into components and vice versa via respectively the *componentification* and *agentification* actions.

In the current state of our work, the three models of service, component and agent are defined and have been presented in [2]. In this paper, we propose the semantic rules of transition between the concepts of these three models, that is, the rules corresponding to the arrows between these three models in our process.

There are already many general or abstract models of component or agent. Nevertheless, we decided to define our own models since none of the existing models satisfactorily integrates explicitly both aspects of interaction and service. These aspects are the keys in our approach to integrate components and agents as explained in the introduction. Similarly, we defined our service model as none of the existing models completely satisfies our need of abstraction, namely not to specify the elements performing the services. [1, 2] describe the existing models of service, component and agent we studied in order to define our three models.

2.2 The three models

Our three models of service, component and agent are shown in figure 3. They are 3 different models (shown as UML class diagrams) but we decided to superimpose them in one figure. Besides the interest of this view in saving space compared to 3 separate diagrams, it shows visually the common and shared concepts between our three models, such as service and interaction. This allows to give a first idea of the general relationships between these models. Complementary OCL constraints (not presented here due to lack of place) fully specify these diagrams. In the following subsections, we summarize the content of our 3 models (for more details, see [2]). The terms in italics correspond to the concepts of the models.

2.2.1 Service model

A *service* is composed of *operations* through *service points*. A *service point* regroups operations in a *required* or a *provided* mode. Then, a *service* is the logical assembly of sets of operations; some sets are provided and others are required (this is done as services are defined in WSDL for instance). A service can be *composite*, that is structurally composed of other services, or it can be a leaf *primitive*. Different services interact between each other via *interactions* associated with their service points. A service plays a given *role* in an interaction. There are two main types of *basic interaction*: *function calls* (RPC / RMI) between a required service point and a provided one or a *delegation* between a service point of a composite service and another service point of the same type of one of its internal services. If an interaction is complex, then we use a *protocol* to define it.

Finally, the concept of *application* corresponds to a global application or system. Since an application is formed of interacting services, it is viewed as a special kind of composite service.

[2]http://www.oracle.com/technetwork/java/javaee/ejb/

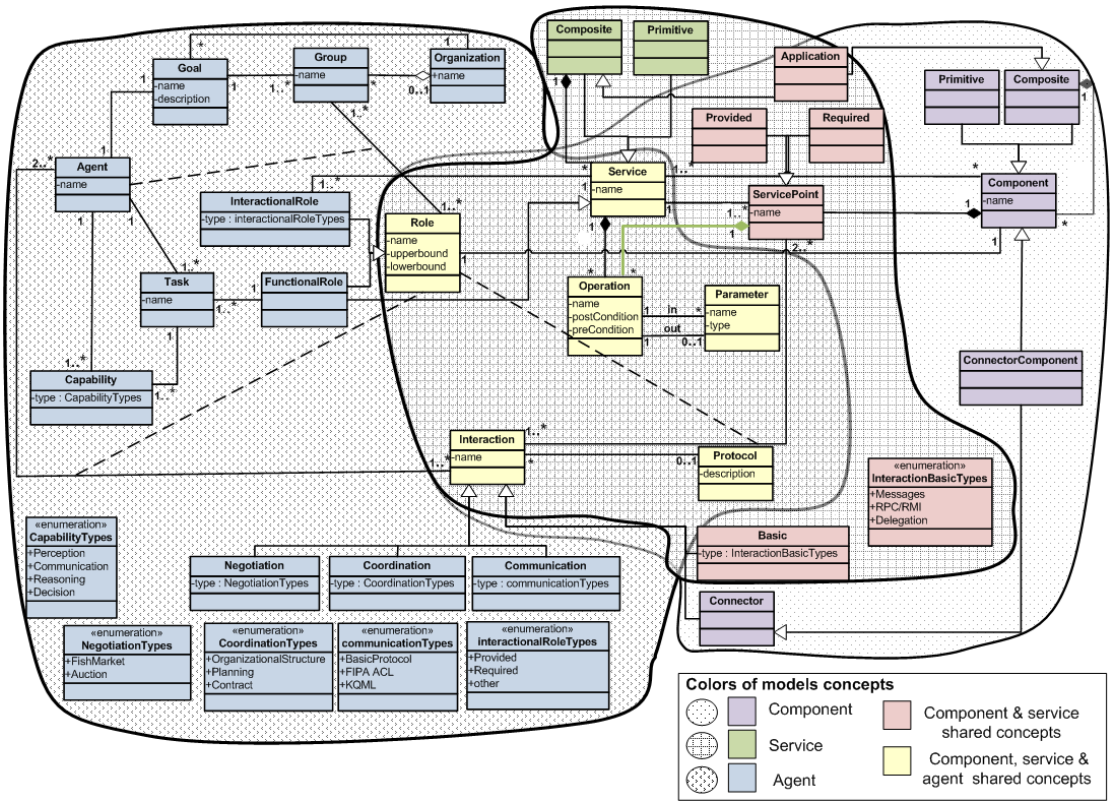

Figure 3: Unified view of the three models (service, component and agent)

2.2.2 Component model

A *component* is a reusable entity with well specified interaction points named *service points*. Each service point is associated with a *service*[3] in a *required* or a *provided* mode. A service is composed of *operations*. A component can be *primitive* or *composite* when it is structurally composed of other components. Different components interact with each other via *interactions* associated with their service points. A component plays a given *role* in an interaction. There are two main types of basic interaction: *functions calls* (RMI / RPC) between a required and a provided service points of different components or *delegation* between a service point of a composite component and another service point of the same type of one of its internal components. If a more complex interaction is required between the service points of different components, it is realized by a *connector* which implements a *protocol*. A *connector component* is a special kind of component dedicated to communication between components [6]. Then, it is also a connector and it implements a protocol.

Finally, the concept of *application* corresponds to a global application or system. Since an application is formed of interacting components, it is viewed as a special kind of composite component.

[3]In component approaches, another vocabulary is generaly used: a port for a service point and an interface for a service. We used these terms to name the common concepts between our three models in the same way.

2.2.3 Agent model

In our work, the notion of agent is related to the organization centered multi-agent systems. In the context of "service-oriented computing" which focuses on the notions of interaction and service, an organization provides ways to build systems of collaborative services [10]. An *organization* is associated with a functional *goal* which can be decomposed onto several sub-goals. These sub-goals are associated with *groups* that present the structural entities of an organization and define the actors and their roles within it. In this context, an *agent* is an autonomous rational entity (associated with a goal) that plays *roles* in a group. A *functional role* allows the specification of an agent's tasks or behaviors through a *service*. A service is a set of *operations*. An *interactional role* defines the responsibilities assumed by an agent in an interaction with other agents. Particularly, it allows to provide or to require services. To play a given role, an agent must own certain *capabilities* like perception, communication and reasoning.

Generally, the agents interact with each other through their interactional roles by using high-level interactions (*communication languages* (such as FIPA-ACL), *negotiation*, *coordination*) based on *protocols*.

3. SEMANTIC MAPPINGS

In this section, we establish the semantic mappings between the concepts of each domain basing on our three previously presented models. Specifically, these mappings allow the transformation or the translation of a specification for

Table 1: Service / component / agent mappings

Service side	*Component side*	*Agent side*
Primitive service	**Primitive component**	**Agent** with a **goal** by default
Composite service with internal services	**Composite component** with internal components	**Group** containing an agent for each internal service/component* with a **goal** by default
Application	**Application**	**Organization** with a **goal** by default
Provided (resp. required) **service point** of a service and its set of operations and playing a **role**	**Service** of a component with the same set of operations	**Functional role** with the same set of operations
	Provided (resp. required) **service point** of a component with the same set of operations and playing a **role**	**Interactional role** with the type of the service point (provided or required)
Interaction associated with a **protocol** between service points	**Connector** associated with a **protocol** between the equivalent service points	**Interaction** associated with an equivalent **protocol** between the equivalent interactional roles
Basic interaction of a given type between service points	**Basic interaction** of the same type between the equivalent service points	**Communication** of basic protocol type† between the equivalent interactional roles

* Only if these internal components or services are primitive. See section 3.3 for the explanations.
† Except for the delegation of the service and component models. See section 3.3 for the explanations.

one model to another one. Table 1 gives the main mappings between the concepts of the three models. Due to lack of place, we are not able to present some examples to illustrate our mappings but they are available online[4].

3.1 Services and service points

The service and component models are very close. We can consider the component model as a direct projection of the service one by adding the elements implementing the services, that is, the components. In the same way, an agent is an element implementing services. However, there are some differences between the three models for the management of the services.

The particularity of our service model is in not defining explicitly the elements supporting the implementation of the services (components or agents in the other models). In the service model, a service contains several sets of operations, each set being associated with a provided or required service point. Here, the service acts as a logical link between these sets of operations. On the component side, this notion of global and logical service disappears since the component becomes its concrete implementation. Then, each set of operations of a service associated with a service point in the service model is mapped onto the same set of operations associated with the service point of its equivalent component. On the component side, we name "service" this set of operations associated with a service point. A component is then associated with multiple services – each service is a set of operations – which correspond to the sets of operations of the service points on the service model side. Therefore, the concept of service, even if consistently associated with operations, is not exactly the same in both models.

Similarly to a component, an agent is an element realizing the implementation of services but it does not directly manage services or service points. In an organizational multi-agent approach, all interactions between agents are done through roles: the interactional roles enable an agent to provide or to require services and these services are defined through functional roles. In the agent model, a functional role is defined as a specialization of the concept of service which is abstract. Then, a functional role on the agent side is mapped onto a service point with its associated operations on the service side or onto a service and its associated operations on the component side. Associated with a functional role, a provided (resp. required) interactional role on the agent side is mapped onto a provided (resp. required) service point on the service or component side.

3.2 Interactions

There exist two main kinds of interaction, the basic ones and the high-level ones. Concerning the basic interactions, there are direct mappings between the three models. The concept of basic interaction is the same in the service and in the component models. This concept is mapped onto the one of communication of a basic protocol type on the agent side (excepting in the case of a delegation because it does not exist in the agent model as explained in section 3.3).

The high-level interactions are defined through a protocol in the three models. However, this protocol is not managed in the same way depending on the model. On the service side, the protocol is used solely to define a high-level interaction. On the component side, a connector is implementing such a protocol. Actually, the connector plays the same kind of role as the component as being the concrete support for realizing an abstract concept: the component realizes a service and the connector realizes a protocol. On the agent side, there is a richer classification based on three main types of interactions (communication, coordination and negotiation). Then, we do not have the same level of details on the agent side compared to the service and component sides. When moving on from agent to service or component side, a non-basic interaction is systematically associated with a protocol (and a connector on the component side), but in the other direction, an automatic default choice must be made or the

[4]http://web.univ-pau.fr/%7Eecariou/casom/

designer must choose between one of the three types of interactions.

When an agent interacts with other agents via a protocol, it must own the required capabilities in terms of communication (language and basic protocols for example, FIPA ACL, FIPA Query, FIPA Request,...) and possibly reasoning and decision making (coordination, negotiation) in order to play the associated interactional roles through this protocol. Unfortunately, on the service or component side, there are no equivalent concepts for the capability one. Then, when moving on from the agent model to the service or component one, all these elements will be "removed", while in the other direction, the designer must add or modify these elements manually on the automatically default obtained specification.

3.3 Hierarchical structures

The base of our mapping rules is that a primitive service is mapped onto a primitive component, both being mapped onto an agent. With the same logic, a composite service with internal services is mapped onto a composite component with internal components mapping these internal services. However, we do not find an equivalent concept for composite on the agent side. Indeed, the organizational structure of the agent model offers only two levels: the group and the agent. A group is a logical structure containing only agents. This causes problems for hierarchical composition of more than 2 levels (for example, a composite service containing composite services[5]). One way to overcome this problem is to use shared agents between different groups to represent the structural levels of an organization. This approach is used in AGR model for instance [7]. If an agent member of a group of level n is also member of the group of level $n+1$, then it can be considered as the representing element of the group of the lower level and all the interactions between the two levels must be managed by this agent. Another problem deals with the delegation of external service points to internal ones for a composite service or component: such a delegation principle does not exist on agent side. When we transform a composite service or component to a group of agents, we consider only the service points of the internal services or components to create the functional roles of the agents (the same set of operations is associated with an internal service point and with its delegated external service point on the composite service or component). For the other direction, from agents to services or components, we must create external service points on the composite and the required delegations between internal and external service points.

The absence of the composite concept on the agent model makes the transformation of an application (on the service or component model) more complex. Intuitively, one can consider a mapping between the concept of application of the service or component model and the concept of organization of the agent one. The problem is that an organization is containing only groups and cannot directly contain agents. In this case, when moving on from the service or component side to the agent one, each primitive service or component directly contained in the application is mapped onto a single agent that is contained in a dedicated group. The composite services or components contained in the application are directly mapped onto groups and agents as explained in the previous paragraph. For the other direction, as an organization is always composed of groups, there is no problem to move on from the agent model to the service or component one.

[5]For this reason, regarding agents, the table 1 presents the mapping of a composite service or component containing only primitive ones. In this case, the mapping is direct and simple: it is a group composed of agents.

3.4 Specific elements of a domain

Some elements or concepts are specific to a given model and have no direct mapping, or no mapping at all, on the other models. The connector component is a specific element of the component model. Such an element is a component but associated with a protocol as it is also a connector. In the direction from the service or agent model to the component one, there is no mapping between elements of the service or agent model and the component connector element (except the mapping variant bellow). In the other direction, from the component side towards the service or agent one, the component connector concept is viewed as a regular component by applying the rules of table 1 and section 3.3 for a primitive or composite component according to the structure of the component connector. The only difference is that its associated protocol is ignored. However, in addition for the mapping to the agent side, the designer can manually add to the mapped agent(s) the required capabilities to implement this protocol.

On the agent side, the table 1 presents for several mapping rules a "goal by default" for an agent, a group or an organization. Indeed, each of these elements must have a goal but this concept of goal does not exist on the service or component side. When moving on from the service or component side to the agent one, it means that the added goal has a description field which is not specified. The designer has to set it afterwards. In the other direction, from agents to services or components, the goal associated with an agent, a group or an organization is ignored.

Finally, still on the agent side, there are no equivalent concepts for task (as for capability or goal) on the component or service model. Then, when moving on from the agent model to the service or component one, all these elements will be "removed", while in the other direction, the designer must add or modify these elements manually on the automatically default obtained specification.

3.5 Mapping variants

It is possible to consider variants of mappings instead of some direct mappings presented above. The first variant consists in not transforming systematically a composite (resp. primitive) element into a composite (resp. primitive) element of another model. If the internal elements in a composite (component or service) or in a group do not interact with each other, we can transform this composite element into a primitive element on another side[6]. This variant is notably interesting in the case where we do not want, on the service side, to specify a service as implicitly formed of provided or required sets of operations (via associated service

[6]If the internal elements are connected, this means that the designer has explicitly specified internal interactions between these elements and that "merging" these elements in just a single one would cause to lose these interactions and the associated explicit internal structure. Then, there would be information loss.

points) but explicitly dedicate each set of operations to a particular service (that is, a primitive service that contains only one service point), defining in this way a composite service containing primitive ones. However, regarding the component or agent side, it is more relevant to have a single primitive component or a single agent as each set of operations is directly associated with a service point of the component or with a functional role of the agent, then with a single service.

A second variant consists, on the component side, to prefer using a primitive component connector instead of a connector to map a complex interaction of the service or agent side.

4. CONCLUSION

Based on previously defined models of service, component and agent, we established semantic mappings rules between these models (including variants of mapping according to the developer needs). These mappings are part of our design process for integrating service, component and agent approaches to take advantage of the benefits of each approach in the specification and development of distributed applications. As far as we know, there are no other works that are interested in providing precise and implementable definitions of mappings between the three domains of service, component and agent (in the organizational field) as we do.

The component model can be seen as an implementation of the abstract service model, where a component implements a service and a connector a complex interaction. Thus, there are bidirectional mappings allowing to move on systematically and totally from a service specification to a component one and vice versa. Regarding the agent side, defining the mappings to the service and the component models is a bit more complicated. The main concepts of interaction and service have systematic bidirectional mappings but some secondary concepts exist only on the agent side without equivalent in the other two models. From the point of view of interactions, they are richer on the agent side compared to the service and component sides. In return, the definition of the composition is less natural on the agent side. These differences in interaction and composition are actually quite logical: we retrieve here the strengths and weaknesses of agent approaches regarding the other two approaches, as stated in the introduction.

The mappings we defined can be applied to our own service, component and agent models. However these models are relatively general by defining the main conceptual principles of each field. Our study of semantic mappings between these three fields can also be considered as a "theoretical" and general one, and not just dedicated to our models. It is possible, through an adequate adaptation, to easily reuse the principles of mappings we established for specific models of service, component or agent.

The perspectives of our work consist mainly of two steps. The first one is the definition of CASOM which is the mixed model containing both component and agent aspects. In addition to CASOM, we will define the relations between CASOM and the other three models (service, component and agent) by adapting the already established mappings. We will also provide a design guide explaining in which context it is more relevant to use agent or component features in an application specification. The second step is the implementation of these four models and the semantic mapping rules in a model driven engineering environment, in order to concretely specify applications through our models. We will use for this purpose the Eclipse/EMF platform[7]. Each model will be implemented as an Ecore meta-model. Concrete model transformations will enable moving on from one model to another.

5. REFERENCES

[1] N. A. Aboud, P. Aniorté, E. Cariou, and E. Gouardères. Towards a Component Agent Service Oriented Model (CASOM). Technical report, LIUPPA / Université de Pau, July 2010.

[2] N. A. Aboud, E. Cariou, E. Gouardères, and P. Aniorté. Service-oriented Integration of Component and Agent Models. In *Special session on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC) of ICSOFT 2011*. SciTePress Digital Library, 2011.

[3] F. Bergenti and M. N. Huhns. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering handbook*, chapter On the Use of Agents as Components of Software Systems, pages 19–32. Kluwer Academic Publishing, 2004.

[4] J.-P. Briot, T. Meurisse, and F. Peschanski. Architectural design of component-based agents: A behavior-based approach. In *4th International Workshop on Programming Multi-Agent Systems (ProMAS 2006)*, volume 4411 of *LNCS*, pages 71–90. Springer, 2007.

[5] E. Bruneton, T. Coupaye, and J.-B. Stefani. The Fractal Component Model, 2004. http://fractal.ow2.org/specification/index.html.

[6] E. Cariou, A. Beugnard, and J.-M. Jézéquel. An Architecture and a Process for Implementing Distributed Collaborations. In *6th IEEE International Enterprise Distributed Object Computing Conference (EDOC'02)*. IEEE Computer Society, 2002.

[7] J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations: an Organizational View of MultiAgent Systems. In *Agent-Oriented Software Engineering (AOSE) IV*, volume 2935 of *LNCS*. Springer, 2004.

[8] R. Krutisch, P. Meier, and M. Wirsing. The agent component approach, combining agents, and components. In *Multiagent System Technologies, First German Conference (MATES' 03)*, volume 2831 of *LNCS*, pages 1–12. Springer, 2003.

[9] S. N. Schiaffino and A. Amandi. User - interface agent interaction: personalization issues, Elsevier. *Int. J. Hum.-Comput. Stud.*, 60(1):129–148, 2004.

[10] M. P. Singh and M. N. Huhns. *Service-oriented computing - semantics, processes, agents*. Wiley, 2005.

[11] J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing Multiagent Systems. *Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers*, 11:307–360, November 2005.

[7] http://www.eclipse.org/modeling/emf/