ROLES IN AGENT-ORIENTED MODELING *

RALPH DEPKE, REIKO HECKEL, JOCHEN MALTE KÜSTER

Department of Computer Science, University of Paderborn,
Warburger Str. 100, D-33098 Paderborn, Germany
E-mail: {depke, reiko, jkuester}@upb.de

For the generic specification of protocols, goals, or workflows, many approaches to agent-oriented modeling provide a concept of *role*. Roles abstract from the concrete agents involved in an interaction. They provide means for the evolution of agents and serve as components of agent design. Despite the wide-spread usage of roles in agent-oriented modeling, a systematic analysis of the different aspects and properties of this concept is still missing. In this paper, we perform such an analysis and identify requirements for a general role concept. We develop such a role concept for a modeling approach based on the UML and graph transformation systems and exemplify its use for the specification (and application) of protocols. Finally, we provide a run-time semantics for roles based on concepts from the theory of graph transformation.

Keywords: Agents, Roles, Graph Transformation, Agent-Oriented Modeling, Software Engineering

1. Introduction

In order to support the general reuse of often occurring interaction patterns interaction between agents is often described in terms of protocols, which represent templates of coordinated behavior (see, e.g., [11]). Instead of making reference to the concrete agents (or agent classes) involved in an interaction, reusability requires to reduce the description of the participating agents to such features and properties that are relevant to the protocol. For this purpose, the concept of *role* has been introduced. Other applications of roles in agent-oriented modeling like the specification of goals, tasks, or responsibilities follow a similar motivation, i.e., to support reuse by replacing concrete agent classes by roles that provide just a minimal set of required features.

Agent-oriented modeling can be seen as an extension and specialization of object-oriented modeling to the particular aspects of agents and multi-agent systems. As modeling concepts, agents and objects have complementary properties: Agents act autonomously, driven by their goals and plans, thereby sensing and reacting to their environment and cooperating with other agents [12]. Objects encapsulate

---

data structures and operations and provide services to other objects. In this sense, Jennings, et al. [26] state that *"There is a fundamental mismatch between the concepts used by object-oriented developers ... and the agent view."* However, the view of objects as mere service providers has its origins in the paradigms of sequential OO programming, and is no longer adequate when considering, e.g., *active objects* in the Unified Modeling Language (UML) [19]. Active objects support *concurrency* and *reactivity*, that is, they have their own thread of control and communicate (mostly asynchronously) by messages.

What is missing in active objects, is the idea of goal-driven behavior or *proactivity* and the related concept of *autonomy*. Autonomy emphasizes the fact that an agent has control over its own operations: They are not called from outside like methods but are only invoked by the agent itself in order to reach a certain goal. From this point of view, an *agent* is to be seen as an *autonomous active object*. Another attribute, that is typically associated with agents, is *cooperation*. This requires pre-defined interaction patterns, i.e., protocols. Therefore, it is no surprise that the concept of role is wide-spread in approaches to agent-oriented modeling, especially when speaking about behavior.

We have introduced an approach to model the behavior of agents using the UML and graph transformation in [8, 6]. The Unified Modeling Language (UML), which is originally designed for the modeling of object-oriented systems, provides extension mechanisms which allow to derive new syntax elements for agents as stereotypes of classes and instances. In addition, graph transformation systems provide a convenient model for the concurrency, reactivity, and autonomy of agents.

Concurrency is potential parallelism of execution, which manifests itself in the existence of separate threads of control for different execution units. In object-oriented modeling the concept of active objects provides a separate thread of control for each object. Independently acting agents are also well described by assigning them an own thread of control. Agents perform operations independently but communication between the agents and their access to shared data (objects) establish causal dependencies of the operations. Semantically, there results a partial *causal order* of the operations performed in a multi-agent system. It abstracts from all possibly executed linear orders of operations extending the causal order. The theory of graph transformation systems provides a concurrent model of computation (see, e.g., [1] for a survey). The basic idea is that two transformation steps are concurrent (i.e., they can be executed in any order, or in parallel) if all items that are shared are in read-access only. Describing agent operations by graph transformation rules this understanding of concurrency becomes available at the modeling level.

Reactivity is the capability of an agent to perceive its environment and react to changes. This property can be considered as a prerequisite for purposeful autonomy of agents, and it is already part of the concept of active object. In our approach, agents perceive their environment by matching the left-hand sides of their graph transformation rules against the current state of the system, thus searching for the occurrence of a certain pattern. Then, agents *react* to an occurrence by the

application of the corresponding rule.

Autonomy is a property of agents that is modeled by the nondeterminism of its behavior if the system is observed externally. Different to objects, agents possess *autonomous operations* that are not automatically triggered by messages but may be invoked by the agents themselves when a corresponding situation pattern occurs in their environment. If several autonomous operations are applicable in a particular situation, the decision which operation to apply is internal to the agent. The inherent non-determinism of rule-based graph transformation provides a convenient model for autonomy.

In this paper, we are going to analyze the use of roles in agent-oriented modeling. Roles are an important concept used for different purposes like the modeling of organizational structure of multi-agent systems, the modeling of protocols, and as components of agent design. Due to its different applications, the notion of role has many different aspects, and most approaches to agent-oriented modeling use only a few of them. However, an appropriate description and formalization of an integrated role concept is missing so far. The first aim of this paper is to provide a systematic analysis of the essential properties and features of this notion. Based on the lessons learned from this analysis, the second aim is to develop a role concept for our own approach to agent-oriented modeling which integrates all the relevant properties.

The paper is organized as follows. In Section 2 we derive the essential properties of roles. We present two typical examples from which the benefit of roles becomes visible and we survey the literature for different properties and applications of roles. In Section 3 we present our concept of roles which, we believe, can be prototypical for many approaches. In Section 4, we describe the run-time semantics for our approach, and Section 5 concludes the paper.

2. Role Concepts in Agent-Oriented Modeling

Roles are used in different ways for the development of agent-based systems. In this section we are going to examine which concepts of roles are needed for agent-oriented modeling.

2.1. *Two Typical Applications of Roles*

Roles have been used for modeling communication, coordination and organizational structure in agent-based systems [25, 3, 20, 27, 26, 2, 15]. Different interaction patterns like auction protocols and the contract net protocol can be formulated in a generic way by the use of roles. These patterns are instantiated by attaching roles to the agents involved in the interaction. The behavior of the roles and the message flow specify a protocol which is transferred to the agents when the pattern is instantiated. Another typical field of application is the description of workflows by roles. Roles abstract from persons (or agents) involved in the workflow in order to isolate the functional aspect. Roles are used to define positions within an organization and thus it must be possible to define relationships between roles. Roles encapsulate
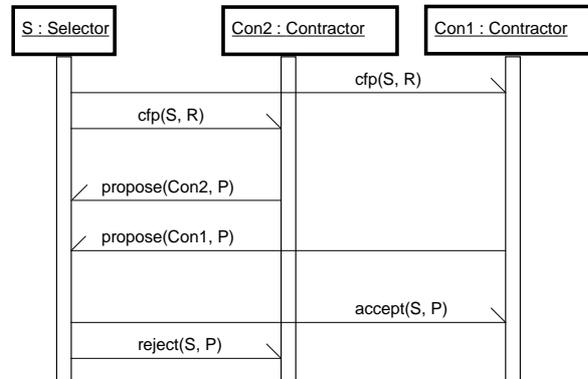
Fig. 1. Scenario of the contract net protocol

certain tasks, responsibilities and goals that an owner of a role has to fulfill. Thus, like for protocols a generic description of workflows makes use of roles. We will present two typical examples of protocols and workflows in order to demonstrate what properties a concept of role should support.

The contract net protocol is a typical interaction pattern that is used for agent communication [24]. In the contract net protocol two role types are involved, the selector role type and the contractor role type. Agents may use this pattern by playing the roles on which the protocol is based.

In Figure 1 a scenario of the protocol is shown in which the selector role solicits proposals from the contractor roles by issuing a call for proposals. Contractor roles receiving the call for proposals are able to generate proposals as it is done by con1 and con2. Alternatively, contractor roles may refuse to propose. Once the selector role receives replies from the contractor roles, it evaluates the proposals and makes its choice of which contractor role will perform the task. The contractor role of the selected proposal will be sent an acceptance message, the others will receive a notice of rejection.

A second example for the usage of roles is agent supported workflow management. Agents are a proper means to cope with a number of shortcomings in current workflow systems [14]. A concept of role is typically used to describe features that participants in a workflow should provide.

A workflow can be described as the automation of a (business) process, in whole or in part, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules. A workflow management system (WFMS) controls the execution of workflows based on their specifications. These systems are distributed in dynamically changing organizations and the workflow tasks are executed concurrently to a high degree. The members of the organizations act autonomously which makes their operations rather
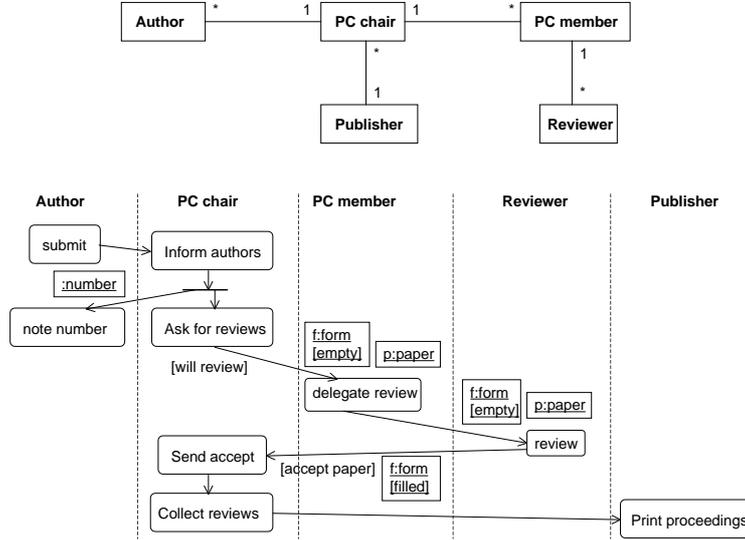
Fig. 2. Program committee role model and paper selection workflow (incomplete)

unpredictable. All these characteristics motivate the usage of agents in workflow management systems.

For further discussion we consider the example workflow of conference management taken from [27]: Managing the paper selection of a conference is a multi-step process involving several individuals and groups. After submission, authors need to be informed that their papers have been received and they are assigned a submission number. In the next step, the program committee (PC) has to handle the review of the papers by asking potential referees to review a number of the papers. After a while, the returned reviews are used to decide about the acceptance or rejection of the submissions. Authors need to be notified of these decisions and, in case of acceptance, must be asked to produce the camera ready version of their revised papers. Finally, the publisher prints the conference proceedings from the camera ready versions of the papers.

The corresponding workflow system consists of agents playing different roles like author, PC chair, PC member, reviewer and publisher, see Figure 2. The roles played by each agent reflect the ones played by the associated person in the conference organization. They require agents to interact both directly with each other and indirectly via an environment composed of papers and review forms.

In the top of Figure 2 a role model shows relationships and multiplicities of the roles involved in the workflow. Every role is assigned a distinct task. For example, the PC chair role is responsible for selecting papers for a conference and a reviewer

must assess a paper for which he is responsible. Thus, roles have structure, dynamics and behavior and in a concrete model of a workflow system agents are built from these components. In the bottom of Figure 2 an UML activity diagram shows a fragment of the workflow just described.

2.2. *Properties of Roles*

From the examples characteristic properties of roles have become visible. They can be either classified as structural or behavioral properties.

First, we will look at structural properties of agent roles. For agents there exist two levels of modeling, the type/class level and the instance level. An agent class is considered as a set of agents with similar features and behavior. In order to adopt this distinction to roles it is necessary to introduce role types and instances. Role typing allows to introduce general role relationships like aggregation or generalization which form role *abstractions*. Role instances are necessary for different reasons. For example, they provide state information which is necessary if roles are protocol participants that have to store the state of an interaction. Multiple role instances for a single agent are demanded if two or more roles of the same type can be played by a single agent. In this case the roles are typically distinguished by their different states. This characteristic role property is called *multiplicity*. At last, role instances are also used as physical components for building agents. An agent and its role instances are manipulated as one entity. This means that an actual role instance may not belong to different agents at the same time. There is always one unique agent that a role instance is mapped to. Thus, a role instance has a weak *identity*, i.e., it depends on the identity of its base agent. In this sense an agent and its roles are said to have the same identity. Also, the existence of roles depends on the agent playing the role. A role (instance) will be deleted when the agent is destroyed, i.e., its lifetime is *dependent* on that of the base agent. Roles can be dynamically attached to and retracted from an agent. This feature is especially important if a role shall migrate from one agent to another. If agents are functionally decomposed into components named roles these agent roles are demanded to be able to act in the same way like agents do themselves.

The behavior of roles is determined by various properties. Roles are used to form different interfaces for agents in order to restrict the *visibility* of features and to handle permissions for the access to the internal state and role services of agents. Roles make features of their base agents visible but they fade out other roles of the agent. The functional behavior is richer for agents than for objects and this aspect inherits to roles. Through their interface roles may provide services in the same way like objects. This property is usually called *reactivity*. Furthermore, with respect to their internal behavior roles may capture goals and handle responsibilities, i.e., they are *proactive*. Finally, agent roles execute tasks *autonomously* in order to reach a goal.

The conference workflow example in Figure 2 illustrates the different aspects of roles. The PC member role can be considered as a component that is instantiated

for every PC member (agent) involved in the review process. The review role could be *instantiated multiple times* for a single PC member agent if different papers are to be reviewed. In the top of Figure 2 a role diagram shows the structural *relationships* between different roles of the conference management organization. Behavior which is associated to the relationship between roles is typically described by some sort of protocol. In the example an UML activity diagram in the bottom of Figure 2 serves for the protocol description. With respect to dynamical role properties the role of a reviewer is typically attached and retracted from an agent who acts as a representative for a scientist in a workflow management system. The *dependency* of a role on its base agent can be seen from the PC chair role which depends on the PC member role, i.e., if an agent retracts the member role it looses the chair role automatically. With respect to their behavior the different roles in the workflow management example serve as templates for agents participating in the protocol, see Figure 2. The restriction of *visibility* and their transient lifetime in connection to their base agent make roles appropriate participants in protocols in which their base agent is involved. The roles restrict the visibility of agent features with respect to the interaction of the protocol. Every role in the workflow example has to execute a special task *autonomously*. The task can be considered either social if for example the PC chair is asking for reviews or individual if the publisher prints the proceedings. As a short summary, lets record that we showed with our example the usefulness of the different role properties.

In this section, role properties like *visibility*, *instantiation*, *dependency*, *identity*, *multiplicity* and *abstraction* have been identified. These properties are not only typical for agent roles but also for object roles as Kristensen has shown in [16]. Additionally, agent roles are considered as agent components and may also carry agent concepts like *reactivity*, *autonomy* and *proactivity*. These aspects make agent roles different from object roles. The application of agent roles like for example in protocol specifications is also specific as we will see in the next section.

2.3. *Role-based Approaches in the Literature*

Regarding the literature we will further examine why roles are important for agent-oriented modeling. Characteristic role aspects within the different role concepts shall be pointed out and identified with respect to the properties named in the previous section.

Wood et al. [25] introduce the Multi-agent Systems Engineering (MaSE) methodology where roles are introduced as more fine-grained building blocks of agent classes which capture agent goals during the design phase. A role serves as a description for the functions it is responsible to fulfill in order to reach an assigned goal. This concept demands role instances because agents are built from role components.

Dependency of roles from their base agent or role is the main characteristics of roles in the Cassiopeia methodology [3]. Roles have three main features: they comprise special behavior, they form the behavior of an agent and they take a position in a group of agents. Individual roles posses behavior that agents are

individually able to perform. Relationships between different roles are considered in order to extract dependencies that are relevant with respect to the collective achievement of the task. The modeler defines influence relationships that indicate those dependencies. Next, relational roles like *influencing agent* or *influenced agent* are derived from the dependencies. For relational roles the designer must also specify the behavior to control the individual roles. At runtime agents choose to interact with each other with respect to the relational roles of their individual roles. Next, all potential groups of agents have to be defined. Finally, new organizational roles are introduced for structuring already established roles into groups. For example, an agent playing an initiator role is able to form a group and thus to dynamically organize collaborations with other agents. Group forming roles are only useful for agents with appropriate relational roles like the *influencing agent* role. From this description characteristic properties of agent roles can be derived. First, roles are components from which agents can be built. Second, relationships between roles steer the social behavior of the agents. Finally, in this role concept there exists a *lifetime dependency* of role instances on the base agent or role.

Omicini [20] describes a methodology for developing multi-agent systems. The focus lies on the coordination viewpoint which means that in multiagent systems the engineering of social aspects is considered important. In the analysis activity a role model is introduced that consists of goals, tasks, roles and groups. Goals are modeled in terms of tasks to be achieved whereas tasks are expressed in terms of responsibilities, required competence and of all resources they depend on. Tasks are classified as either individual or social and are associated to roles and groups. Groups are characterized both by the responsibilities related to their social task and the social roles participating in the group. Interaction between roles is modeled by *protocols* and for groups interaction rules restrict the behavior of the social roles.

In a similar modeling proposal to the previous one Zambonelli et al. [27] focus on role models in order to express organizational structure and (behavioral) rules of multi-agent systems. Interactions and thus *protocols* are well-identified and localized in the definition of a role. They express kinds of social behavior of an agent within an organization of agents. Thus, the notion of a role gives an agent a well-defined position in an organization (stated in a role model) and a set of behaviors expressed by protocols for the roles.

According to Wooldridge et al. [26], a role has associated to it responsibilities, permissions, activities, and protocols which are defined by specific role schemata. Responsibilities comprise lifeness conditions like the execution of a prescribed sequence of protocols that specify the interaction of roles. The exchange of messages obeys predefined *protocol rules* and every agent playing a role has to observe the rules.

In agent modeling approaches based on UML notation typically an UML concept of roles is used. Bauer et al. [2] define protocol diagrams that describe agent interaction relying on collaboration roles. These roles can be used to fix the view on instances of the class when the instances are accessed via an association to the

class. Kendall [15] introduces role modeling for capturing patterns of interaction of agents. Roles comprise a position and a set of responsibilities, which are made up of services and tasks. Services are accessible through interfaces. But it is not clear which UML concept of roles underlies the role patterns. Altogether, concepts of roles that rely on UML emphasize an interface notion of roles. Roles are able to restrict the *visibility* to an agent's *services*. Note, that this notion of role is limited as we have shown in [5].

In the previous overview of agent-oriented modeling concepts with roles the aforementioned role properties could be found. This justifies the identification of role properties in Section 2.2. Additionally, we conclude that agent roles are used for three main purposes. First, roles express organizational structure within a multiagent system. Relationships between roles together with behavior rules are captured in role models. Moreover, roles are members of protocols which are considered as generic interaction patterns between agents. For this purpose roles restrict the visibility to a base agent's features and they contribute features that are necessary for protocol execution. Roles are attached dynamically to agents during the execution of the protocol and are retracted afterwards. At last, roles are used as components from which agents are built. Thus, roles can be instantiated and they carry similar concepts like agents, for example, autonomy or pro-activity. Next, we are going to develop an integrated concept of roles that captures many of the aforementioned role properties and that is useful for different purposes.

3. Agent-Oriented Modeling with Roles and Graph Transformation

After the discussion of the previous section we see the concept of roles as a fundamental one for agent-oriented modeling, for the following reasons: agent-based systems are distributed systems where different agents communicate with each other exchanging messages. This distributed system is of a dynamic nature, thus interactions are not fixed but constantly change. Roles enable the generic description of interactions by defining protocols. Furthermore, roles enable the dynamic changing of interaction configurations. Finally, roles can be used as agent-building blocks, i.e., they are components from which agents can be composed. In this section, we present our concept of roles that combines all these three important usages of roles.

In our approach, roles are modeled as stereotyped classes and we distinguish between roles on the instance and type level. We introduce a special *role-of* relationship relating roles to their base which exists both on the type level (as role-of association) and instance level (as *role-of* link).

Thereby, the specific properties of roles discussed in Section 2 are supported as follows. Distinguishing between roles at the type and the instance level allows the use of inheritance and other relationships between roles (*abstractivity*). Furthermore, roles as stereotyped classes enable their use as agent components.

Other important role properties are achieved by the *role-of* relationship. We require roles in a *role-of* relationship to be lifetime dependent on the base class (*dependency*). From a structural viewpoint, the role-of relationship is similar to
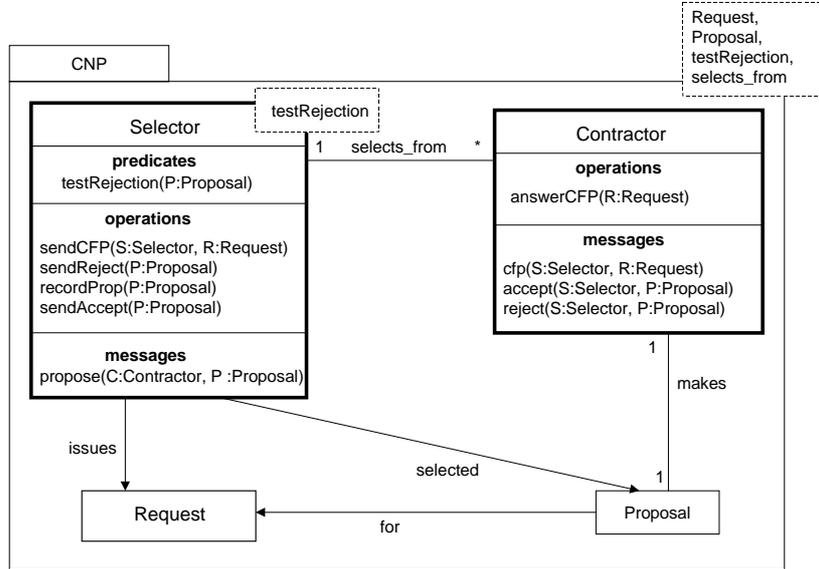
Fig. 3. The generic organizational model of the contract net protocol

composition in UML as role instances may be added to and removed from agent instances. Considering the behavior of roles, it provides implicit delegation because roles can access their parents' features as if they were their own which achieves the property of *visibility*. In addition, we allow the adornment of the *role-of* relationship with cardinalities in order to express the number of multiple role instances related to the base class which yield the property of *multiplicity*.

We use roles for several purposes in modeling agent-based systems.

Firstly, roles can be used for expressing the organizational structure of a multi-agent system. This can be achieved in our approach using class-like diagrams consisting of role classes and ordinary classes. In Figure 3, an organizational structure of the contract-net protocol is shown consisting of the two role classes Contractor and Selector and two ordinary object classes Request and Proposal. In this case, we use parameterized role classes as we see the contract-net protocol as a generic protocol that can be used in different contexts and combine the model in a parameterized package CNP. The parameters Request and Proposal describe the request and proposal messages sent within the protocol. The predicate testRejection describes the selection policy of the Selector and the association selects-from connects the Selector and its potential Contractor(s).
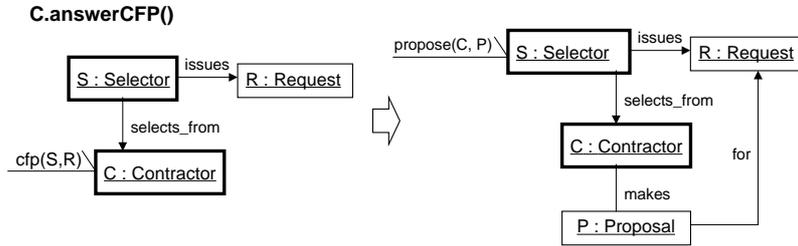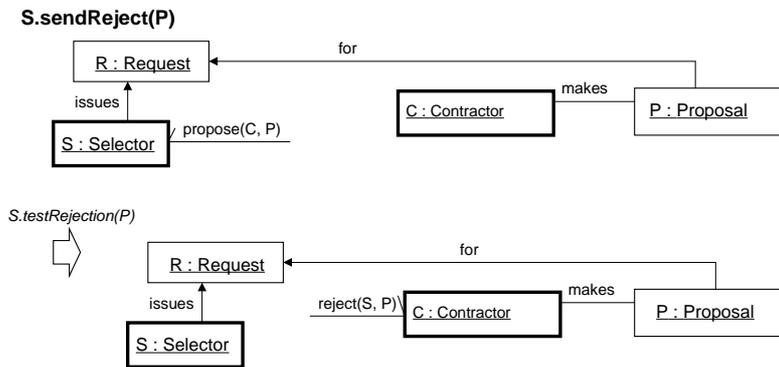
C.answerCFP()

Fig. 4. The generic answerCFP rule

S.sendReject(P)

Fig. 5. The generic sendReject rule
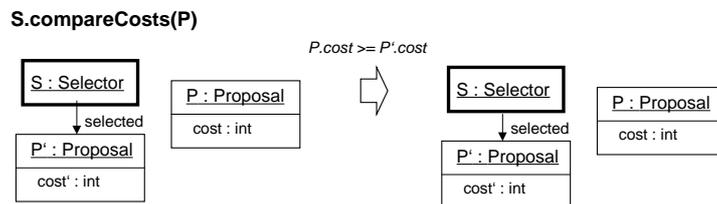
S.compareCosts(P)

Fig. 6. The compareCosts rule

Secondly, roles can be used for specifying interactions in a generic way. This is achieved by using a set of graph transformation rules for describing the operations of roles. With respect to the first usage of roles, the description of organizational structures, this application of roles can also be seen as the specification of interactions within an organizational structure.

In order to illustrate these concepts, let us recall the example of the contract-net protocol. In Figure 4 and Figure 5 graph transformation rules for the operations of the generic role classes are specified. A rule consists of a left- and right-hand side describing the precondition and the effect of the operation, respectively. The precondition describes when the rule is applicable. For example, the answerCFP rule is only applicable if the corresponding structure exists (i.e. there is a Selector instance connected to a Contractor instance by a selects-from link and to a Request instance by an issues link) and the cfp-message has been sent to the Contractor instance. The right-hand side of the rule describes the structure resulting from the operation. In the case of the answerCFP rule, a new Proposal instance is created and a propose message is sent to the Selector. Note that these rules are also generic, i.e., they depend on the parameters of the generic role classes. For example, the sendReject rule makes use of the testRejection predicate which is a parameter of the package.

The use of graph transformation for specifying operations supports two important aspects of agents, *autonomy* and *reactivity*. *Autonomy* is supported by the agent's ability to choose from different applicable operations, driven by its internally described goals. As a consequence, our model incorporates a certain degree of non-determinism which is to be resolved by the agent's internal decision mechanism. *Reactivity* is supported because the left hand side of a rule can be interpreted as a specific pattern the agent is able to react to. The concept of graph transformation can be seen as an orthogonal one compared to roles. For a more detailed treatment of graph transformation for agents the reader is referred to [6].

Thirdly, roles can be used as agent-building blocks in class diagrams. This is achieved by attaching a role class to an agent base class using the *role-of* relationship. In particular, in our approach, these agent-building blocks can be role classes that are obtained by binding the parameters of generic role classes described as an organizational model. As a consequence, we see our role concept as one that combines all possible important usages of roles, namely the modeling of organizational structures, the modeling of generic interactions and the usage of roles as agent-building blocks.

Concrete role classes can be defined using a parameterized generic role class and binding the parameters. For example, in Figure 7, the CNP::Selector is obtained by binding the parameters of the generic Selector role class of the package CNP. The concrete Selector role has as its base class the PBA and the concrete Contractor role the AccountAgent meaning that they are both lifetime dependent on their base classes. In addition, cardinalities are used to model that the AccountAgent takes part in several interactions as a contractor whereas the PBA may only take part

Fig. 7. Binding of template parameters and *role-of* relations in the class diagram

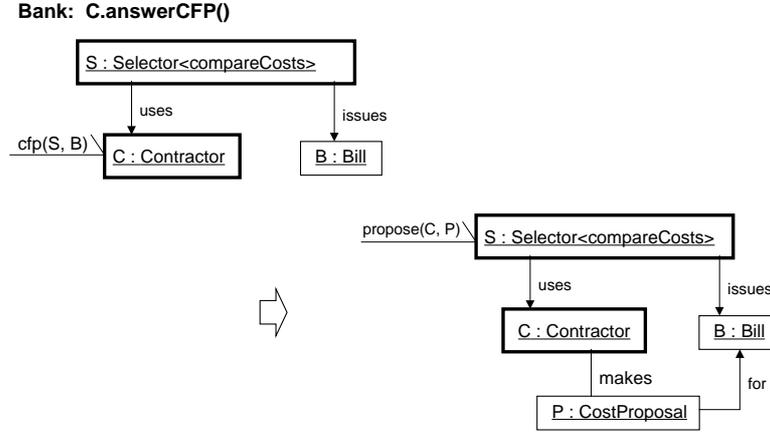**Fig. 8.** The expanded class diagram of the model

Fig. 9. The instantiated answerCFP rule

in one (modeled by multiplicity 1 at the role-of relationship to Selector). For the reader's convenience, an expanded class diagram representing the complete and instantiated structural model is shown in Figure 7. This expanded model could be derived automatically by a tool that implements the described transformation.

The behavior of the concrete role classes is determined by the rules for the generic classes as follows. Rules for the concrete role classes are obtained by replacing generic role names and association names with their concrete counterparts. For example, in Figure 9 the answerCFP rule is depicted obtained by renaming of types according to the binding described in the class diagram (e.g. the selected link is replaced with the uses link and the Proposal with the Bill). If a parameter operation like testRejection is used within a generic rule like CNP::sendReject the corresponding concrete rule Bank::sendReject is obtained by substituting within CNP::sendReject the formal by the actual parameter. In our example, this leads to an amalgamation of the rules CNP::sendReject and compareCosts, the result of which is depicted in Figure 10. The left side of the amalgamated rule is obtained by the union of the left sides of both individual rules (shown in the figure by different shadings). Similarly, the right side of the amalgamated rule is obtained by the union of the right sides of both individual rules. As the sendReject rule requires the applicability of the compareCost operation (expressed by the predicate testRejection which is bound to the compareCosts operation), in the amalgamated rule the predicate of the compareCost rule $P.cost \geq P'.cost$ has to hold.

4. The Dynamic Semantics of Roles

In the previous sections, class diagrams containing role classes have been used to model the structure of a sample agent-based application based on a generic spec-
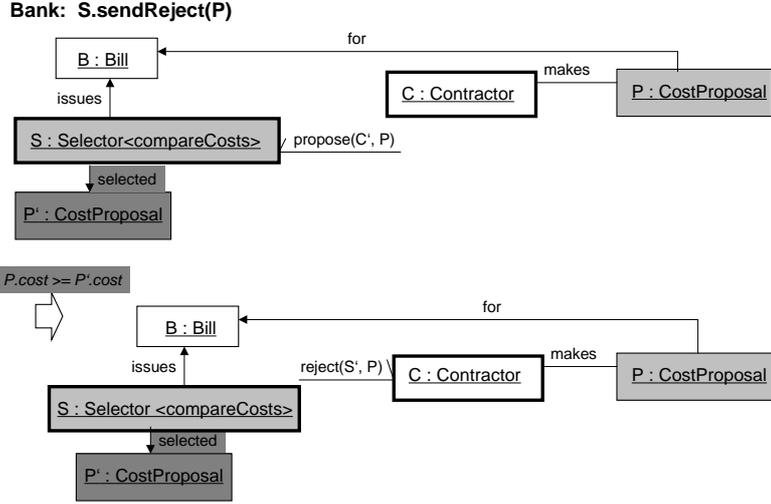
Bank: S.sendReject(P)

Fig. 10. The instantiated Bank::sendReject rule as amalgamation of CNP::sendReject and Bank::compareCosts

ification of the contract net protocol. The operations of the protocol roles have been specified by graph transformation rules and *role-of* relations have been used to associate these operations to the agents playing the respective roles. In this section, we make explicit the different levels of modeling inherent to this approach and we point out how the essential properties of roles, like *abstractivity, dependency, visibility,* and *multiplicity* are reflected in an object-oriented context at these levels.

Behavioral modeling techniques like graph transformation (or sequence diagrams, activity diagrams, etc.) often distinguish between three levels of modeling: the *class level*, the *instance level*, and the *specification level*. The class level, given by a class diagram, specifies the state space of the model, i.e., a collection of instance diagrams representing concrete system states. The specification level is given by graph transformation rules and describes transitions between states (resp. their representations as instance diagrams).

Both the instance and the specification level are associated to the class level by a *typing relation* which maps each agent or object to its respective class. However, these two typing relations differ w.r.t. their interpretation of *role-of* relations in class diagrams. At the specification-level, a role can access features of its base class, i.e., in order to support the interpretation of roles as interfaces structuring the *visibility* of features, the *role-of* relation provides implicit delegation of feature access. Since, moreover, roles can be used without base in transformation rules, this simplifies drastically the specification of operations since various redirections can be omitted. Finally, the possibility not to mention the base but to associate base class features directly with roles is essential for the use of roles for the generic

specification of protocols (cf. *abstractivity*).

With respect to the instance level, the *role-of* relation in the class diagram is interpreted like an ordinary association with multiplicity "1" at the base. That means, every role instance is connected to exactly one base instance by a *role-of* link. This allows to support multiple role instances of the same class at the same base instance, as required by the *multiplicity* property. Thus, it remains to clarify how the seemingly different interpretations of *role-of* on the specification and the instance level are related. This will also answer the question, how to deal with the life-time dependency between roles and their base when agents and roles are dynamically created and deleted.

The conceptual link between the specification level (i.e., the transformation rules) and the instance level (the states) is given by a notion of *transformation* explaining how a rule is applied to a given state yielding a derived state. Formally, we think of a transformation as an embedding of a transformation rule into a *context* given by the agents, objects, links, and attributes not directly affected by the application. Thus, in order to specify a notion of transformation, we have to describe the possible embeddings of rules into contexts.

The idea is to use meta-level *embedding rules* of the form

$$\frac{L \to R}{L' \to R'} \quad C$$

where $L \to R$ is the premise, $L' \to R'$ the conclusion, and C a side condition, typically a generic fragment of the class diagram. The set of transformations using a given rule $r = (L \to R)$ is then given by all transformations $G \to H$ derivable from r by means of the embedding rules, such that both G and H represent consistently typed instance diagrams over the class diagram. Conceptually, this use of embedding rules is inspired by the contextualization rules in SOS [21] (like the rule for restriction in CCS [18] stating under which conditions an action can be performed in the context of the restriction operator). Formally, our embedding rules are rules of a meta graph grammar [13] defined on the abstract syntax graphs of transformation rules and transformations. However, for the purpose of this paper, an intuitive understanding of these rules on the level of the concrete syntax of our modeling language shall be sufficient.

Before we turn to the embedding rules for roles and *role-of* relations, we specify the basic embedding policies of graph transformation. Although not directly related to roles, these policies provide two prototypical solutions for dealing with dependent items in the presence of deletion. In the sequel, this will be relevant for the handling of role dependency.

In Figure 11, the rule in the upper left states that, given any transformation, for every class C in the class diagram we can add an instance o:C to both the pre and the post state to obtain another transformation. The rule in the upper right states that a transformation can be extended by introducing, on both sides, links between instances, provided this is permitted by the class diagram. The lower left rule specifies the possibility of gluing two instances that are preserved during the
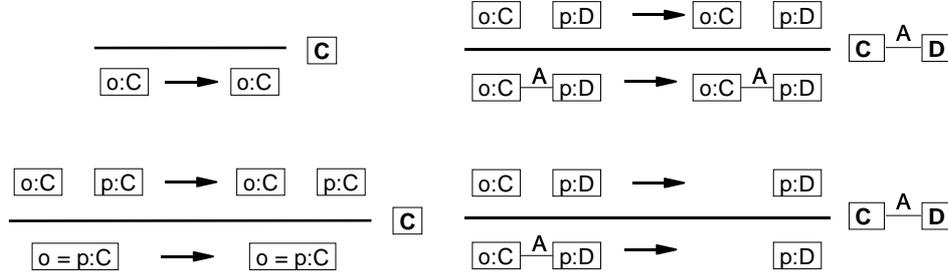
Fig. 11. Standard embedding rules of graph transformation: introduction of instances (upper left), gluing of instances (lower left), introduction of links (upper right), and implicit deletion of dangling links (lower right)

step, provided that they have the same type. A similar rule could be defined for the gluing of links. Jointly, these three rules specify a conservative embedding policy as it is typical for the DPO approach [10], where a rule application is given by an embedding of a rule into an otherwise constant context. This policy imposes a restriction on the applicability of a transformation rule in a given state: A rule deleting an instance must not be applied if this would leave a link dangling without source or target instance. This so-called *dangling condition* is the conservative way to ensure that the resulting structure is again a well-defined graph, i.e., that the dependency between links and their source and target instances is respected.

The second obvious strategy to ensure this dependency is the implicit deletion of dangling links. This destructive solution is typical of the so-called *single-pushout (SPO) approach* [17].[†]The embedding rule for the implicit deletion of links attached to deleted instances is given in Figure 11 in the lower right.

The embedding rules for *role-of* are shown in Figure 12. The rule in the upper left specifies the obvious introduction of *role-of* links between two instances that are preserved during the step. It is similar to the upper right rule in Figure 11 for ordinary links. The two different ways to deal with dependency are applied to the dependency of role instances from their base as follows. The conservative way, which does not require any particular embedding rule, is represented by the static constraint that each role must be the source of exactly one role-of link. As a consequence, it is forbidden to perform a transformation which leads to a violation of this constraints, e.g., by deleting the base without the role. The destructive solution is specified by the two rules in the bottom of Figure 12. Notice, that the

---

[†]Historically, the first of the algebraic approaches to graph transformation is the so-called *double-pushout (DPO) approach* introduced in [10], which owes its name to the basic algebraic construction used to define a direct derivation step: This is modeled indeed by two gluing diagrams (i.e., pushouts) in the category of graphs and total graph morphisms. More recently a second algebraic approach has been proposed in [17], which defines a basic derivation step as a *single* pushout in the category of graphs and *partial* graph morphisms: Hence the name of *single-pushout (SPO) approach*. See [4, 9] for an overview and a comparison of both algebraic approaches.
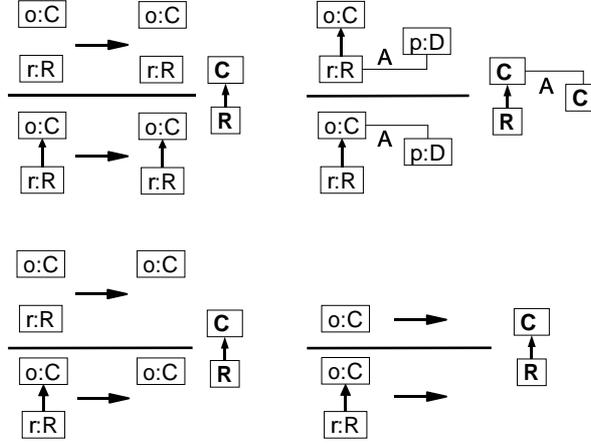
Fig. 12. Embedding rules for *role-of* relation: introduction and implicit deletion of *role-of* links (upper and lower left, resp.), implicit delegation of links (upper right), and implicit deletion of roles (lower right)

situation is asymmetric, i.e., the left rule specifying the implicit deletion of dangling *role-of* links is applied when the role instance (at the source of the arrow) is deleted. If instead the base instance (at the target) is deleted, the rule in the lower right applies, i.e., the role is deleted along with the *role-of* link.

Probably the most interesting rule is the one in the upper right. It shows the semantics of *role-of* as relation with implicit delegation: In a transformation rule, a feature of an instance o (like a link, in this case) may be accessed through its role r. That means, given a *role-of* relation in the class diagram, roles can be used instead of their base classes and all the features of the base carry over to the roles. However, on the instance level (i.e., at run-time) a *role-of* link is interpreted as composition. Therefore, by means of the rule in the upper right of Figure 12, the referred features have to be delegated back to the base instance where they belong according to the instance-level interpretation of the class diagram. (Recall that the same class diagram gives rise to two different notions of type consistency: one more liberal for the specification level (rules) which allows implicit delegation along *role-of* relations, and one more restrictive on the instance level (transformations) which treats *role-of* relations as purely structural.) Notice that, in contrast to the other rules which specify patterns for both the left- and the right-hand side, the premise and conclusion of the implicit delegation rule consist only of a single graph each. That means, this rule can be applied freely to instances on both sides of a transformation, regardless whether they are created, deleted, or preserved.

A sample transformation using the transformation rule answerCFP of Figure 9 is shown in Figure 13. In order to derive this transformation from the rule, it requires two applications of the embedding rule for instances (upper left of Figure 11), two
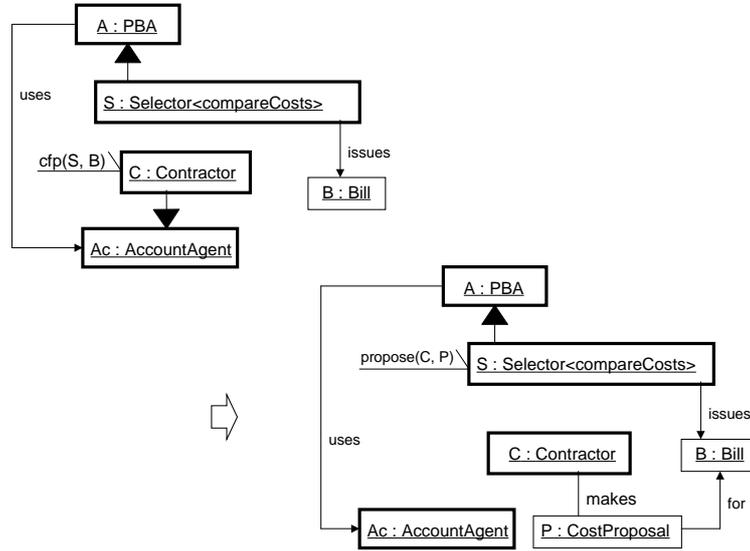
Fig. 13. A transformation using the rule Bank::answerCFP

applications of the rule for the introduction of *role-of* links (upper left in Figure 12), and finally two applications of the rule for link delegation (upper right in Figure 12). By the two applications of the rule for instances, two isolated agents :PBA and :AccountAgent are introduced on either side of the transformation that shall become the base of the respective roles :Selector and :Contractor. Thus, in the next two deduction steps, the two corresponding *role-of* links are added. Now, each role is associated to a single base instance, as required. However, the resulting pair of graphs is not yet a legal transformation because, according to the (instance-level interpretation of the) class diagram in Figure 8, the uses link has to connect the base agents instead of their roles. Thus, on each side of the transformation, two applications of the delegation rule are required, each moving one end of a uses link from a role instance to its base. The result is a pair of legal instance graphs and thus a valid transformation.

5. Conclusion

In this paper we have analyzed the use of roles in agent-oriented modeling. Roles are an important concept used for modeling participants of protocols where each agent participating in a protocol is assigned a certain role which it plays as long as the protocol is performed. Role models are also considered appropriate for modeling organizational structure of agents and roles are also interpreted as components of

agent design. We developed an integrated concept of roles that is appropriate for the mentioned purposes. Agent roles are notated as UML active class or object boxes which are connected to its base agent or role by a *role-of* relationship. The behavior of roles incorporates implicit delegation of features because a role has the ability to access its base agent's features directly. A structural condition demands that at the instance level the existence of an agent's role depends on that of the base agent, thus resembling the behavior of *composition* in UML. These aspects are reflected in the semantics of our modeling approach. There, agent dynamics is specified by graph transformation rules which represent patterns of communication or computation steps. The semantics of these transformation rules is described by specifying how a rule leads to an actual graph transformation on the complete system. A graph transformation is thought of an embedding of a transformation rule into a context. We used embedding rules on the meta level to specify the class of all legal embeddings for each transformation rule. Role aspects like the visibility and lifetime dependency constraints reflect in special embedding rules that modify graph transformations for rules containing roles.

This notion of role is compatible with our work reported in [8, 6]. There we have presented an agent-oriented modeling process based on UML notation and concepts of typed graph transformation systems. The agent-oriented modeling process is divided in a typical sequence of activities which is already well known from the modeling of object-oriented systems, i.e., requirements specification, analysis and design [23]. In requirements specification graph transformation rules are used to describe test cases, i.e., the pre- and postconditions of essential interactions of use cases. In the analysis stage the system's structure is captured in an agent class diagram. Mandatory functions of the agents and the resulting mandatory interactions (protocols) between agents are derived from the specified test cases and are expressed by graph transformation rules. In the analysis the rules have a universal semantics which is different from requirements specification. In the design model the analyzed functions and interactions are expressed by local operations that are defined by graph transformation rules.

In [7] a concept of roles is introduced to support the systematic transition between the different stages of the agent-oriented modeling process. In analysis it enhances the derivation of agent and object classes and protocols. In the design model the fine-grained modeling by roles eases the identification and coordination of local operations. In the future, the formal semantics of our role concept based on graph transformation and embedding rules are to be included in our agent-oriented process model.

References

1. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution*, pages 107 – 188. World

Scientific, 1999.

2. B. Bauer, J.P. Müller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Software Systems. *Int. Journal on Software Engineering and Knowledge Engineering*, 2001. Included in this special issue.

3. A. Collinot and A. Drogoul. Using the cassiopeia method to design a soccer robot team. *Journal of Applied Articial Intelligence*, 12(2-3):127–147, 1998.

4. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, Part I: Basic concepts and double pushout approach. In Rozenberg [22], pages 163–245.

5. R. Depke, G. Engels, and J. M. Küster. On the integration of roles in the UML. Technical Report No. 214, University of Paderborn, Dep. of Comp. Sci., August 2000.

6. R. Depke, R. Heckel, and J. M. Küster. Formal agent-oriented modeling with graph transformation. *Science of Computer Programming*. To appear.

7. R. Depke, R. Heckel, and J. M. Küster. Improving the agent-oriented modeling process with roles. In *Proc. Fifth Int. Conference on Autonomous Agents (AGENTS-2001)*, Montreal, Canada, June 2001. ACM Press. To appear.

8. R. Depke, R. Heckel, and J.M. Küster. Agent-oriented modeling with graph transformation. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June 2000*, volume 1957 of *LNCS*, pages 105–120. Springer-Verlag, Berlin, 2001.

9. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation, Part II: Single pushout approach and comparison with double pushout approach. In Rozenberg [22], pages 247–312.

10. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.

11. Foundation for Intelligent Physical Agents (FIPA). Agent communication language. In *FIPA 97 Specification, Version 2.0*. FIPA, 1997.

12. S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J.P. Müller, M.J. Wooldridge, and N.R. Jennings, editors, *Proc. ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III*, volume 1193 of *LNAI*, pages 21–36. Springer-Verlag, August 12–13 1997.

13. R. Heckel and A. Zündorf. How to specify a graph transformation approach: A meta model for FUJABA. In H. Ehrig and J. Padberg, editors, *Uniform Approaches to Graphical Process Specification Techniques, satellite workshop of ETAPS 2001, Genova, Italy*, 2001. To appear.

14. N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Journal of Applied Articial Intelligence*, 14(2):145–189, 2000.

15. E.A. Kendall. Agent Software Engineering with Role Modeling. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June 2000*, volume 1957 of *LNCS*, pages 163–170. Springer-Verlag, Berlin, 2001.

16. Bent B. Kristensen. Object Oriented Modeling with Roles. In *Proc. 2nd International Conference on Object-Oriented Information Systems (OOIS'95), Dublin, Ireland, 1995*, pages 57–71, London, 1996. Springer.

17. M. Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109:181–224, 1993.

18. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

19. Object Management Group. UML specification version 1.3, June 1999.

20. Andrea Omicini. SODA: Societies and infrastructures in the analysis and design of

agent-based systems. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June 2000*, volume 1957 of *LNCS*, pages 185–194. Springer-Verlag, Berlin, 2001.

21. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.

22. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations.* World Scientific, 1997.

23. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object–oriented Modelling and Design.* Prentice–Hall, 1991.

24. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In *IEEE Transaction on Computers*, number 12 in C-29, pages 1104–1113, 1980.

25. M. Wood and S. A. DeLoach. An Overview of the Multiagent Systems Engineering Methodology. In P. Ciancarini and M.J. Wooldridge, editors, *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June 2000*, volume 1957 of *LNCS*, pages 207–222. Springer-Verlag, Berlin, 2001.

26. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

27. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *Int. Journal on Software Engineering and Knowledge Engineering*, 2001. Included in this special issue.