

Improving the Agent-Oriented Modeling Process by Roles

Ralph Depke

Universität Paderborn, FB 17
33098 Paderborn, Germany

depke@upb.de

Reiko Heckel

Universität Paderborn, FB 17
33098 Paderborn, Germany

reiko@upb.de

Jochen M. Küster

Universität Paderborn, FB 17
33098 Paderborn, Germany

jkuester@upb.de

ABSTRACT

The agent-oriented modeling process is divided in a typical sequence of activities, i.e., *requirements specification*, *analysis*, and *design*. The *requirements* are specified by descriptions of the system's functionality and by exemplary scenarios of essential interactions. In *analysis* the system's structure is captured and mandatory behavior of agents is prescribed. The *design* model describes system behavior by means of local operations. The problem arises how the transition between these different stages of the modeling process can be performed. In this paper, we introduce a concept of roles in order to support the transition in a systematic way and thereby improving the agent-oriented modeling process.

Keywords

Agents, Roles, Modeling Process, Software Engineering

1. INTRODUCTION

In this paper, we discuss a concept of roles and how it can be used to improve the agent-oriented modeling process. Roles are a means to ease a systematic transition between different stages of the modeling process (e.g., *analysis* to *design*). First, we will summarize the agent properties that we capture during the modeling process. Before we start our discussion of roles and their application in our context we will review the agent-oriented modeling process that we have introduced in [5, 3].

For the purpose of this paper, the term *agent* is to be identified with *autonomous active object*. This incorporates properties like *concurrency*, *reactivity*, and *autonomy*, which are characteristic of agents [8]. In particular, the concept of autonomy goes beyond the notion of an object because a feature like method invocation is not usual for agents. Other relevant concepts like *cooperation* and *goal-driven behavior* are discussed elsewhere (see e.g., [3]).

Concurrency is potential parallelism of execution, which manifests itself in the existence of separate threads of con-

trol 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. *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 captured within the concept of active objects. *Autonomy* is a property of agents that manifests itself in 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.

We have introduced an approach to the modeling of agents using the Unified Modeling Language (UML) and graph transformation in [5, 3]. The UML provides several sublanguages that support the modeling of structure and behavior of object systems. By using extension mechanisms of UML we derive syntax elements like the *agent* as a stereotype of an actor in the use case diagram. Additionally, graph transformation systems provide a convenient model for the concurrency, reactivity, and autonomy of agents. The theory of graph transformation systems provides a *concurrent* model of computation which relies on a partial order of the transformation steps (see [1]). 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. Agent operations are given by graph transformation rules and in this way concurrency of operations can be described appropriately. 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. Agents possess *autonomous* operations which are modeled by graph transformation rules. If different rules have the same left hand side (lhs) the agent itself has to decide which one to apply in the situation when

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada
Copyright 2001 ACM 1-58113-326-X/01/0005 ...\$5.00.

the lhs occurs in the agent's environment.

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 [14]. The *requirements* are specified in a use case diagram where agents are considered as actors internal to the system. The narrative description of the function of each use case is supplemented by exemplary scenarios of essential interactions of the use case. Graph transformation rules are used to describe the preconditions and the postconditions of the interactions. In this way, test cases are specified that have to be demonstrated by the system. In the *analysis* stage the system's structure is captured in an agent class diagram that consists of agent and object classes and contains relations between the classes. Mandatory functions of the agents and the resulting mandatory interactions (protocols) between agents are derived from the specified test cases. Graph transformation rules and sequence diagrams express the results of the analysis. They have a universal semantics different from requirements specification. In the *design* model the analyzed functions and interactions are expressed by local operations whose order of execution is determined by state charts.

From this process model, the problem arises how the transition between the three stages can be performed. In the analysis stage agent and object classes and their relations have to be derived from the use case diagram and from the scenarios. From the exemplary scenarios mandatory behavior of agents has to be derived which is represented as protocols. The transition from analysis to design is also difficult. The protocols have to be mapped to local operations in a systematic way. The order of the operations in a singular protocol that an agent executes has to be fixed. Additionally, the different protocols that an agent is involved in at the same time have to be coordinated.

In object-oriented modeling, roles are used in order to encapsulate functionality which may change dynamically when the object evolves. The visibility and also the access to an object via a role is restricted to the attributes and methods of the role of an object and to the object's visible own features. At any time, an object may play one or several roles which can be of the same role class.

In the context of agent-based systems, roles are used for various purposes. Firstly, roles describe the organizational structure of multi-agent systems [18]. An organizational role model incorporates different roles, their relationships and behavioral rules. In a multi-agent system, the agents operate with respect to a given role model. Secondly, roles are used for the modeling of protocols (see, e.g., [7]). In this case, each agent participating in a protocol is assigned a certain role which it plays as long as the protocol is performed. 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. Roles are also used as components (so called agent-building blocks) from which agents are to be composed [16].

In this paper, we introduce a concept of roles in order to support the transition between different stages of the agent-oriented modeling process in a systematic way and thus to improve the process. We show how visually modeled roles can be used in order to describe the *development* of organizational structure and interaction protocols. At last, our

concept of roles also supports the usage of roles as agent-building blocks because we consider roles as instantiable units.

In Section 2, we survey the three main phases of system modeling, i.e., requirements specification, analysis, and design and explain how this general process model is specialized in our case. In Section 3, we explain how roles are used in object-oriented and agent-oriented modeling and we introduce a concept of roles that can be integrated in the agent-oriented modeling process. In Sections 4 and 5 we show how by the introduction of roles the transition between different stages of the modeling process can be better supported. At last, Section 6 concludes the paper.

2. MODELING AUTONOMOUS AGENTS WITH GRAPH TRANSFORMATION AND UML

In this section, we summarize and partially review the agent-oriented modeling process that we introduced before in [3]. We divide the modeling process in a typical sequence of activities which is already well known from the modeling of object-oriented systems [14]. First, the requirements are specified by informal descriptions of the system's functionality and by scenarios of important interactions. In the analysis model, the requirements specification is analysed and refined. Thereafter, in the design model the behavior that has been described globally in the analysis model is expressed by the local behavior of objects and agents. In [3], this approach is explained in more detail and a formalization based on the theory of graph transformation leads to a notion of consistency between requirements specification, analysis and design. In the following we will shortly review our approach. For a more detailed discussion the reader is referred to [3].

Requirements Specification. The functional and architectural requirements of a software product are specified by *use case diagrams* in an informal way. They provide an abstract view of the system by identifying the main actors using it and the main functions that the system provides to them. In the context of agent-based systems, UML use case diagrams are extended by a special kind of actor representing the agents *within the system* [10]. In this way, additional architectural requirements about the distribution of the system's functionality among different agents can be expressed.

The abstract narrative description given by use cases is illustrated by typical examples, called *scenarios*, of how the system behaves when a use case is performed. Scenarios are specified in two complementary ways. The overall effect of a scenario is described by a *graph transformation* on the instance level, i.e., a pair of instance diagrams modeling a *before-after* scenario of the use case. In order to specify the communication between actors participating in a scenario, UML *sequence diagrams* are used.

Analysis. In order to serve as a basis for future design decisions, the requirements are analysed and refined. Similar to object-oriented analysis, the refined model is structured into submodels [14], a *structural model*, a *dynamic model* and a *functional model*. As *structural model*, a class diagram specifies the types of objects and agents (presented as active classes), their attributes, associations, and messages. In the

case of agents, messages do not automatically result in the execution of methods since agents decide autonomously how and when to react to an incoming message. The autonomous operations of agents are specified in the design model.

The *functional model* specifies the overall effects of a use case on the state of the system. In the requirements specification, this has been done by a graph transformation on the instance level. Formally, this transformation can be seen as an individual test case which has to be realized by the implementation of the system. However, in order to have a complete view of the use case's overall effect, many such graph transformations would be needed. Thus, a mechanism is required to specify (rather than to enumerate) pairs of graphs. The theory of graph transformation suggests a rule-based approach to this problem. A *graph transformation rule* $L \rightarrow R$ consists of a pair of graphs L, R representing a rule's preconditions and postconditions. In the UML terminology, these are diagrams on the specification level, i.e., unlike graph transformations in the requirements specification, a rule has a mandatory interpretation: Whenever the precondition specified by L is satisfied in a given state G , which is witnessed by a subgraph isomorphism $o : L \rightarrow G$, called occurrence, at least that part of G is deleted which is matched by $L \setminus R$ and, to the resulting graph, a copy of $R \setminus L$ is added leading to the derived graph H . Notice that, during analysis, rules are considered as incomplete specifications of the transformations to be performed, i.e., additional (unspecified) changes are permitted. This (quite liberal) notion of *graph transition* [6] is strengthened in the design model by the notion of *graph transformation* which assumes a complete specification of the changes during a step.

The *dynamic model* complements the functional model by focusing on the communication required to execute a certain protocol. Like in the requirements specification, we use sequence diagrams to model the message flow between agents in the system. However, during analysis, we strengthen the semantics of these diagrams from an existential to a universal interpretation. Thus, a sequence diagram associated with a graph transformation rule provides a complete specification of the interactions to be performed when the precondition is met.

Design. The analysis phase is concerned with developing a model of what the system is supposed to do. The design model elaborates the analysis model concentrating on the question how the system will function. As a consequence, the focus of models is shifted from a global view on the system during analysis to a local view, thus providing the basis for an implementation. Like in analysis, we distinguish a structural model, given by a refined class diagram, a dynamic model, describing, by means of state diagrams for each class, the order in which operations of this class can be performed, and a functional model specifying the effect of operations using graph transformation rules. Models constructed during design are refinements of models of the requirements specification and analysis phase. In the *structural model*, the class diagram of the design phase refines the class diagram of the analysis adding, in particular, the signatures of the agent's autonomous operations for which an extra compartment is provided. Notice the difference with *methods* as specified in the method compartment of objects: agent's *operations* are *autonomous*, that is, they are never called by another object or agent but only executed under

control of the agent itself. As a consequence, we distinguish agent's messages and operations while in the case of objects, both notions are integrated in the notion of method. By a state diagram for each agent class, the *dynamic model* specifies the ordering of operations an agent of this class may perform. As agents do not automatically react to events of their environment but decide autonomously when and how to react, transitions are not labeled with an event and an action but only with the name of the operation. The notion of a protocol state machine [13] comes closest to our understanding of statecharts. In the *functional model*, the operations declared in the structural model are specified by graph transformation rules. Whereas the dynamic model is concerned with the order of operations, the functional model shows how operations change the state of the system.

3. ROLES IN OBJECT-ORIENTED AND AGENT-ORIENTED MODELING

We will first give a general definition of the notion of a role in object-oriented modeling and then we explain which properties a concept of roles should support. Roles are defined almost consistently throughout the literature [12, 9]. According to Kristensen et al., a role of an object is a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects. This definition is refined by a set of characteristic properties that concepts of roles should support [11, 9]:

- *visibility*: The visibility and also the access to an object is restricted to the attributes and methods of the role of an object and to the object's visible own features.
- *dependency*: The existence of the role depends on that of the object it belongs to. This is very similar to composition of objects from other objects.
- *identity*: An object and its roles have one identity. It is manipulated as one entity. As a consequence, this means that an actual role may not belong to different objects at the same time. There is always one unique object or role that a role is mapped to.
- *dynamicity*: Roles may be added and removed during the lifetime of an object. Thus one role is deleted and another one is created while the related person object survives.
- *multiplicity*: Several instances of a role may exist for an object at the same time.
- *abstractivity*: Roles can be related to each other by aggregation and generalisation.

These characteristics can be viewed as requirements for a role concept and each role concept can be evaluated against these requirements.

Recently, we have inspected whether UML supports an appropriate concept of roles and the result is negative [2]. We showed that the mechanism of a rolename falls short of fully supporting the requirements of roles for the following reasons. A rolename is only a sort of label and specifies behavior only if it is bound to an interface. Concerning multiplicity, rolenames do not provide the necessary means of type and instance level. Other than rolenames there exists

also the possibility to simulate roles with existing mechanisms such as inheritance or aggregation. We checked each of these mechanisms against the role requirements presented above (for an extended version of simulating roles the reader is referred to Kristensen et al. [12]). We refine our argumentation regarding rolenames and interface specifiers. If roles are identified with interfaces then they are not able to have state and thus to have attributes. As a consequence, multiplicity of roles is not possible because it depends on the state of the role. Regarding multiple inheritance an object inherits from all its role objects. Then public attributes and methods are visible to all other objects during the lifetime of the objects. Therefore, the visibility and dynamicity properties of roles get lost. Regarding aggregation, roles depend on the object they belong to. This kind of dependency demands the role, i.e. the part object to disappear when the object does. In an aggregation relationship the part objects may exist independently of the aggregate object. In order to support this property the variant of aggregation named composition is the more appropriate, because it respects this kind of dependency. The visibility of public features of a role's base object can only be obtained indirectly, because attributes and methods are only accessible through navigation to the base object.

We conclude that there exist no appropriate mechanisms in the UML for simulating roles. The composition relationship fulfills most criteria of roles, but the behavior of the part objects lacks a role's ability to access its base object directly. Therefore we introduced the new relationship *role-of*. The subset of operations, attributes, and associations of a class required to play a role is represented by a *role class* which must be connected to a base class by a *role-of* relationship. This new relationship demands that features of the role class must be different from those of the base class. On the instance level, the existence of an object's role depends on that of the base object itself, thus resembling the behavior of *composition*. In a class diagram and also in an object diagram, a *role-of* relationship is displayed as an association with a full-filled triangle on the side of the base class, see figure 3. We introduce agent role classes as agent classes that depend on an agent class by a *role-of* relationship. In this way roles of agents have similar properties like roles of objects. The behavior is different because agent roles are active objects and thus run concurrently to their base agent.

In agent-oriented modeling, roles are used for modeling of protocols (see, e.g., [7]). In this case, each agent participating in a protocol is assigned a certain role which it plays as long as the protocol is performed. More generally, agent roles are used for capturing goals, tasks, or functions exhibited by the agent. According to Wooldridge et al. [17], a role has associated to it responsibilities, permissions, activities, and protocols which are defined by specific role schemata. Responsibilities comprise liveness conditions like the execution of a prescribed sequence of protocols. In this way the interaction of roles is specified. In the agent-oriented modeling approach of Zambonelli et al. [18] roles and role models are used to express organisational structure and (behavioral) rules of multi-agent systems. In an organisation an agent plays one or more roles. Interactions and thus protocols are well-identified and localised in the definition of a role. They express kinds of social behavior of an agent within an organisation. Thus, the notion of a

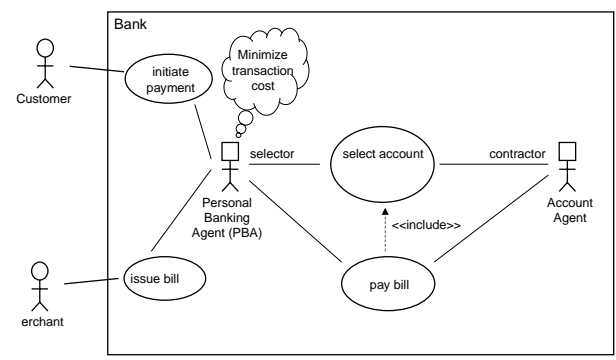


Figure 1: Use case diagram

role gives an agent a well-defined position in an organisation (stated in a role model) and a set of behaviors expressed by protocols for the roles. Considering roles as participants in protocols allows the specification of generic interactions which can be used and reused in various multi-agent systems of similar organizational structure. Wood et al. [16] introduce a 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 an abstract description for the functions it is responsible to fulfill in order to reach an assigned goal.

From the agent-oriented approaches above we conclude that roles are a proper means of refining agent-oriented modeling. Their protocols and their (social) relationships are main characteristics of roles. Different to the approaches above we will explain in the next two sections in which way our concept of roles fits into a diagrammatic agent-modeling language based on UML and how it can enhance the modeling process. We explain in which way graph transformation fits together with our concept of role. Using these two techniques the identification and appropriate generic description of protocols becomes feasible. Thereafter, protocols can be applied in a multi-agent system by just attaching roles to agents.

Next, we show how our concept of roles, i.e., (agent) role classes and the *role-of* relationship, can be used to reduce problems of the transition between different stages of the agent-oriented modeling process. We show that organizational structure and protocols can be derived step by step in analysis and design. In the design model role classes get attached to agent classes using *role-of* relationships. Thus, an agent participating in a protocol is built (instantiated) from role classes that belong to the protocol specification. First we explain how the transition from requirements specification to analysis can be eased and how protocols can be identified.

4. FROM REQUIREMENTS SPECIFICATION TO ANALYSIS

An example will illustrate the use of roles. We introduce roles in the model of an online banking system taken from our previous work [5]. A *personal banking agent* (PBA) is responsible to pay bills for a customer who controls the agent. A merchant issues bills not to the customer himself but to his PBA. The customer has to initiate the payment in order

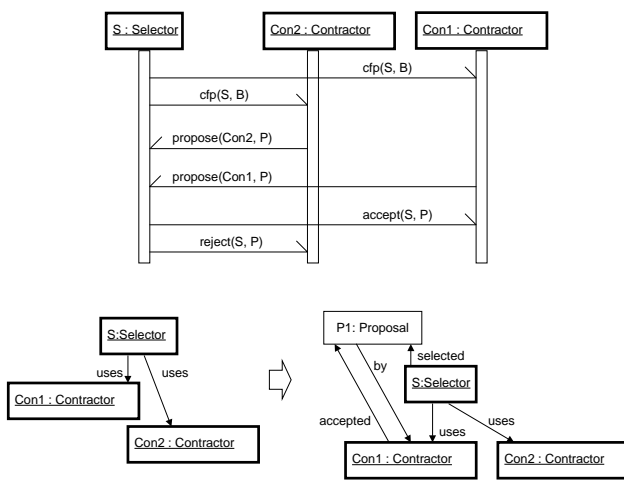


Figure 2: Sequence diagram (scenario) and corresponding graph transformation rule

to get a bill paid by the PBA. The agent selects an account from which money is to be taken and transfers the amount to the account indicated by the bill. In the following paragraphs we explain in which way the introduction of roles supports the methodology presented above.

At the beginning of a software development process functional requirements are given by means of use case diagrams. Examples for the behavior of the system (scenarios) are expressed by sequence diagrams and global graph transformation rules. The use case diagram of Figure 1 identifies, besides two kinds of users, the agents PBA and AccountAgent (actors with square heads). These agents participate in the use cases **select account** and **pay bill**.

In the requirements specification, an actor's interactions are determined by the use cases it is participating in. Each participation of an actor in an interaction is represented by a role, comprising the agent's features relevant to the corresponding use case. A role name may be attached to the association connecting actor and use case. In Figure 1 the agent PBA acts as a **selector** and the agent AccountAgent is involved as a **Contractor** in the use case **select account**.

The narrative description of the system's functionality is refined by *scenarios* which are typical examples of interactions necessary to perform a use case. The interaction of the roles that participate in a use case is expressed by the message flow in a *scenario diagram*, see the top of Figure 2. The interaction conforms to the *contract net protocol* [7] which refines the behavior of the use case **select account**. A *global graph transformation rule* describes the preconditions and the effect of the scenario. In the bottom of Figure 2 the result of an account selection is shown. Again, only the roles involved in the use case are taken over in the rule.

The introduction of roles at this stage of development has the advantage that the behavior of the agents is partitioned according to the use cases they are participating in. In this way possible complexity is reduced. Dynamicity of roles implies another advantage in the modeling of behavior described by use cases. A use case is activated only for some period of time during system execution. This behavior can be modeled adequately by attaching the roles to those agents

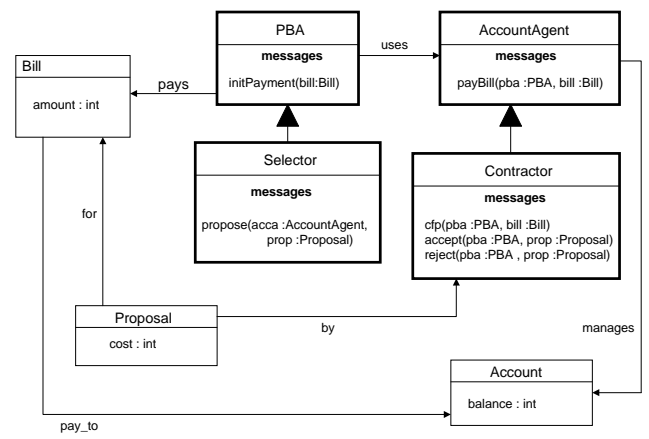


Figure 3: Class diagram

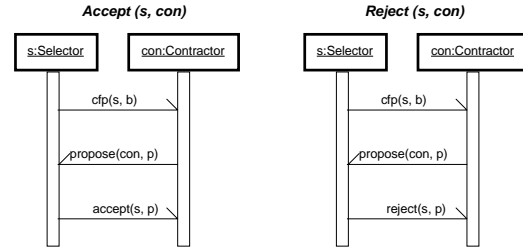


Figure 4: Basic interactions between agent roles

participating in a use case just before the execution period and to retract them afterwards. For example, in Figure 1 the use case **select account** is activated only for some period of time during execution of the use case **pay bill**.

When global graph transformations and sequence diagrams are used to describe the scenarios of a use case, each occurrence of an agent in a rule or a diagram corresponds to a different role. The corresponding role classes can be systematically constructed by collecting all the resources necessary in order to perform the specified activities. Then, the classes in the analysis model can be derived by integrating these role classes.

In Figure 2, the sequence diagram contains the roles that are participating in the interaction and also the messages sent to them. The roles are inserted in the class diagram and are related to the base classes by a *role-of* relationship. The corresponding graph transformation rule in the bottom of Figure 2 contains associations that are to be taken over in the class diagram. The resulting class diagram for our example is shown in Figure 3.

In the analysis of the scenarios from requirements specification protocols can be derived that agent roles have to execute. The interaction of pairs of roles is examined in order to find protocol steps and protocol alternatives. Basic interactions between two roles are projections of more complex interactions. Next, we introduce graph transformation rules that describe the precondition and the effect of the basic interactions. A set of graph transformation rules forms a protocol. It should be possible to derive all scenarios from requirements specification by the application of rules of a protocol.

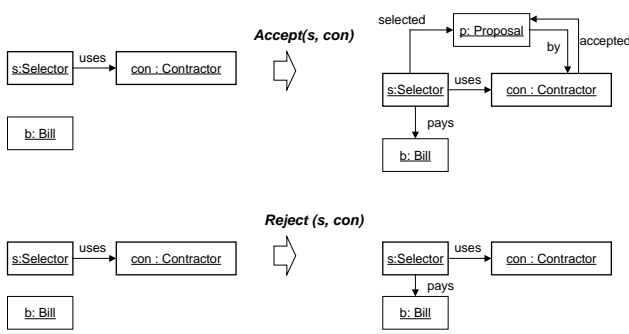


Figure 5: Rules for analysis

In Figure 3 the **Selector** role interacts independently with both **Contractor** roles. There are two alternative effects of the interaction that depend on a common precondition. In Figure 5, two rules describe these alternative steps within the protocol. They result from the basic interactions in Figure 4 which are projected from the scenario in Figure 2. The first rule models the case that the **Proposal** of the **AccountAgent** is accepted by the **PBA** and the second one the rejection of the **Proposal**. A rule is activated when the precondition of the corresponding rule is met. For the rules in Figure 5 the precondition requires that the **Account Agent** is connected with the **PBA** by a **uses** link, and that a **Bill** object exists.

We conclude that the analysis of scenarios results in graph transformation rules that specify a protocol based on the roles participating in the interaction. At this stage of the modeling process we can shorten the subsequent design activity if it is possible to reuse existing design models of protocols. Thus, if the analyzed protocol has been designed previously it is possible to just use the protocol by attaching prefabricated role classes with graph transformation rules as operations to the agent classes. Alternatively, the protocol has to be designed in detail in another process step which is described in the next section.

5. FROM ANALYSIS TO DESIGN

Roles also make the transition from analysis to design more systematic. Each role represents the local contribution of an object or agent to a global interaction. Thus, for each interaction specified in the analysis model, first the required local behavior is realized for each interaction role. In a second step, the behavior of all the role classes belonging to an agent class has to be coordinated, e.g., by synchronizing the statecharts of the individual roles, yielding the overall behavior of the class. In this way the complexity of the behavior of a class participating in many different interactions is structured, and the potential auto-concurrency of such classes can be explicitly modeled by creating a new role instance for each new interaction. This kind of partitioning structure and behavior of agents also makes the use of generic protocols feasible.

Role classes can be instantiated like object or agent classes, but their instances do only exist in connection with an agent or object. That means, a role instance automatically disappears together with the object, agent or role it depends on. Thus, from a structural point of view, the *role-of* relationship is similar to a composition in UML. Considering the be-

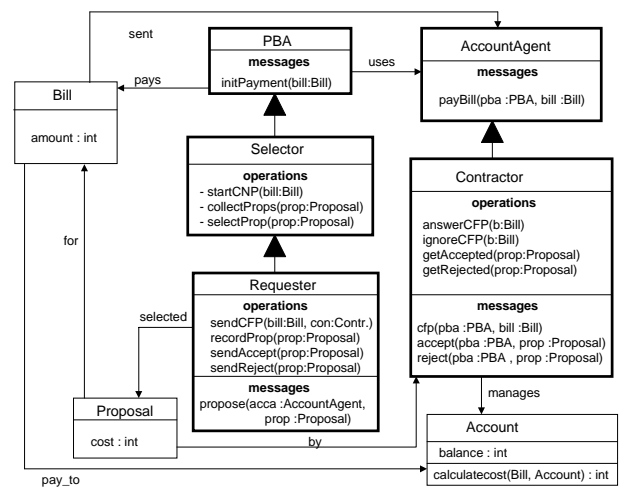


Figure 6: Class diagram

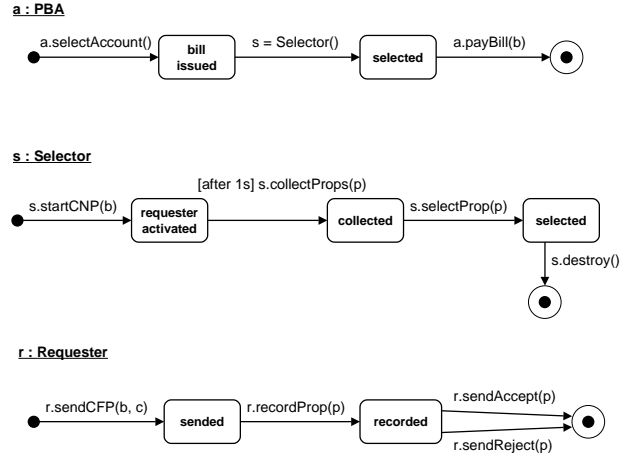


Figure 7: Statecharts for PBA, Selector role and Requester role

havior of roles, it supports implicit delegation because roles can access their parent's features as if they were their own. In Figure 6, the operations of the agents are distributed to their roles. In this way the agents' behavior is partitioned according to the use cases the agents are participating in.

The operations of a role are executed within the interactions the role is participating in. The possible sequences of operations in such a protocol are constrained by statecharts. This *dynamic model* of an agent is split into submodels that are attached to the agent's roles. In this way possible complexity of the agents' statecharts is reduced. In Figure 7 the statecharts for the **Requester** role, the **Selector** role and the **PBA** are shown. The selector role is created by the PBA after reception of an **initPayment** message (a rule for the operation **selectAccount** is left out). The **Selector** role acts independently of the PBA who just waits for completion of the selection. The behavior of the **Selector** role is given by the rules from analysis, see Figure 5. The **Requester** role is introduced in order to take over this functionality. A **Requester** role interacts with a **Contractor** role whose dynamic behavior is given by the statechart in Figure 8. Different to

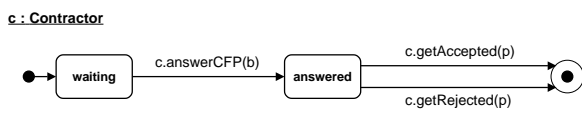


Figure 8: Statechart for Contractor role

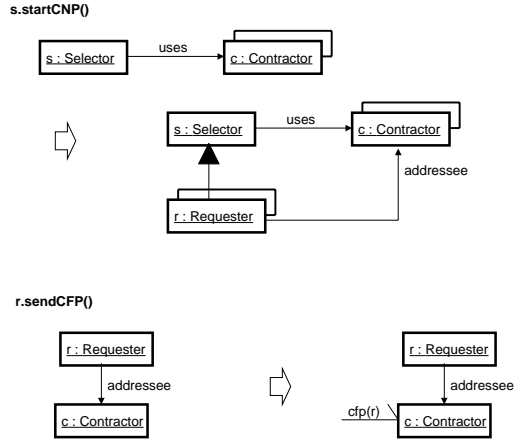


Figure 9: Operations startCNP and sendCFP

the functional behavior the dynamic model of the **Requester** role can not be specified independently. In a second step the **Requester** role has to coordinate with the **Selector** role to which it is associated by a *role-of* relationship. Therefore in the statechart of the **Requester** role the transitions to the final state depend on the event that the operation **selectProp** of the selector role is executed. The operation can be introduced in the statechart because all features of the base class are visible and accessible to the role.

In the *functional model*, the operations declared in the structural model are specified by graph transformation rules. Whereas the dynamic model is concerned with the order of operations, the functional model shows how operations change the state of the system. As agent's operations may only affect that part of the state which can be accessed locally, we require that all (role) objects in the left-hand side of a rule are reachable via a path originating at the *self* agent. In the rules it is not necessary to depict the roles' base objects or agents because the roles inherit all available features. By the introduction of roles the graph transformation rules typically describe only part of the state change that is effected by rules for whole agents. In this way, the rules for the agents are divided up to rules for the agents' roles. The rules for roles are more readable and less complex than the rules for objects und agents. This also eases the implementation of the graph transformation rules.

In Figure 9 a graph transformation rule for the operation **startCNP** of the role **Selector** is shown. By the application of this rule for each contractor a corresponding **Requester** role is created. In this way, it is modeled that the **Selector** role communicates independently because the **Selector's Requester** roles act concurrently. They are controlled by an instance of the statechart for the **Requester** role class, see Figure 7. After the **Requester** role is created it applies the operation **sendCFP** autonomously and sends a call-for-proposal

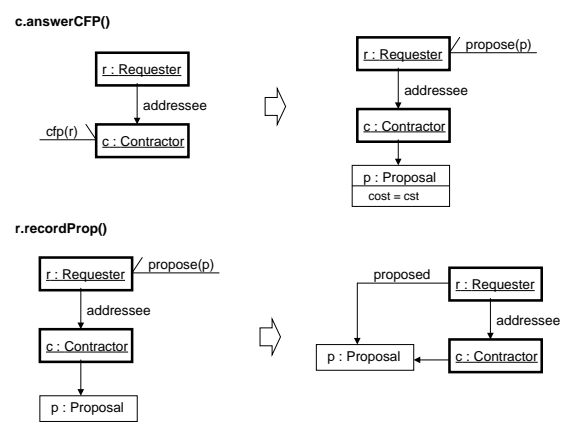


Figure 10: Operations answerCFP and recordProp

message (**cfp**) to the Contractor role it is linked to, see Figure 9. The Contractor role applies the operation **answerCFP** if it detects the **cfp** message, see Figure 10. It creates a **Proposal** and sends a **propose** message to the requester. The Requester role catches the **Proposal** when it executes the operation **recordProp**, see Figure 10.

The Selector, Requester and Contractor role classes in Figure 6 and its autonomous operations which are defined by graph transformation rules in Figures 9 and 10 and which are controlled by statecharts in Figures 7 and 8 specify a reduced kind of the contract net protocol [15]. This specification can be reused in another agent-oriented model just by attaching the role classes to agent classes which shall execute the protocol. Possibly, some protocol features have to be renamed in order to make them disjoint from base features or to fix interaction features with the base class explicitly.

Now, we arrived at the end of the modeling process. A variation from this process is to use a protocol library already from the beginning. In this case, in requirements specification a use case diagram contains application specific use cases and use cases representing protocols from a library. These protocol use cases would typically be included into the application specific use cases. The agents in the use case diagram are connected with a protocol use case according to different interaction roles of the protocol. In analysis the class diagram contains protocol role classes that are attached to agent classes by a *role-of* relationship according to the use case diagram. Graph transformation rules for these roles and statecharts for control of the roles' behavior can be taken from the library and integrated in the analysis model and the design model. In this way the whole modeling process can be further facilitated and sped up.

6. CONCLUSION

In this paper, we explained how a concept of roles supports the transition between different stages of the agent-oriented modeling process in a systematic way. In [5, 3] we have presented an approach to 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 like requirements specification, analysis and design. In the analysis stage the system's structure is captured in an agent class

diagram which is derived from a use case diagram. Communication protocols are extracted from the scenarios (test cases) specified in the requirements. In this paper, a concept of roles is introduced that supports the systematic derivation of agent and object classes and protocols in analysis. In the design model the analyzed functions and protocols are expressed by local operations whose order of execution is restricted by state charts. The fine-grained modeling by roles eases the identification and coordination of operations in design. A major benefit of our notion of roles is the ability to specify the structure and behavior of protocols. By use of the *role-of* relationship the roles of a protocol can be attached to agents participating in the protocol.

It has become clear that roles cover three important aspects: specification of both organizational structure and generic behavior and agent-building from components. In the class diagram the organizational structure of the multi-agent system is determined mainly by the role classes and its associations. Generic behavior is specified by protocols which incorporate only roles and which are introduced in the use case diagram as protocol use cases and in the class diagram by use of the *role-of* relationship. In this way agents are built from agent components, i.e., from its roles. In [4] we work out the different aspects of agent roles more thoroughly and we cover the use of protocols in a more technical manner. Especially, the relationship of protocol behavior and base agent behavior is examined in more detail.

7. 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] 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.
- [3] R. Depke, R. Heckel, and J. M. Küster. Formal agent-oriented modeling with graph transformation. *Science of Computer Programming*. To appear.
- [4] R. Depke, R. Heckel, and J. M. Küster. Roles in agent-oriented modeling. *Int. Journal on Software Engineering and Knowledge Engineering*. To appear.
- [5] 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.
- [6] G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A combined reference model- and view-based approach to system specification. *Int. Journal of Software and Knowledge Engineering*, 7(4):457–477, 1997.
- [7] Foundation for Intelligent Physical Agents (FIPA). Agent communication language. In *FIPA 97 Specification, Version 2.0*. FIPA, 1997.
- [8] 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.
- [9] Georg Gottlob, Michael Schrefl, and Brigitte Rock. Extending object-oriented systems with roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.
- [10] C. A. Iglesias, M. Garijo, J. C. González, and Juan R. Velasco, *Analysis and design of multiagent systems using MAS-CommonKADS*, Proc. 4th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-97) (M.P. Singh, A. Rao, and M.J. Wooldridge, eds.), *LNAI*, vol. 1365, Springer-Verlag, July 24–26 1998, pp. 313–328.
- [11] 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.
- [12] Bent B. Kristensen and Kasper Østerbye. Roles: Conceptual Abstraction Theory and Practical Language Issues. *Theory and Practice of Object Sytems*, 2(3):143–160, 1996.
- [13] Object Management Group. UML specification version 1.3, June 1999.
- [14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modelling and Design*. Prentice-Hall, 1991.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organisational Abstractions for the Analysis and Design of Multi-Agent 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 235–252. Springer-Verlag, Berlin, 2001.