Combining AOR Diagrams and Ross Business Rules’ Diagrams for Enterprise Modeling

Kuldar Taveter  
VTT Information Technology  
(Technical Research Centre of Finland)  
P.O.Box 1201, FIN-02044 VTT, Finland  
kuldar.taveter@vtt.fi

Gerd Wagner  
Institute of Informatics, Free University Berlin  
Takustr. 9, 14195 Berlin, Germany  
gw@inf.fu-berlin.de

Abstract

An enterprise model is a comprehensive description of the organizational structure, the information resources and the business processes that constitute an enterprise with the help of a conceptual modeling method. Existing methods, however, have difficulties to capture the physical and social dynamics which is inherent in organizations, and to integrate it with a static information model. Agent-oriented methods represent a promising approach to overcome these difficulties. In this paper, we investigate the combination of a recently proposed agent-oriented approach, called Agent-Object-Relationship (AOR) modeling [8], with the more established business rule modeling method of Ross [10] in modeling a car rental company.

1. Introduction

Agent-Orientation is emerging as a new paradigm in software and information systems engineering. It offers a range of high-level abstractions that facilitate the conceptual and technical integration of communication and control with conventional (object-oriented) information storage and retrieval. Agent-Orientation is highly significant for business information systems since business processes are driven by and directed towards agents, and hence have to comply with the physical and social dynamics of interacting individuals, groups and markets.

In order to capture more semantics of the dynamic aspects of information systems, such as the events and actions related to the ongoing business processes of an enterprise, it is necessary to make an ontological distinction between active and passive entities, that is, between agents and objects. In particular, the semantics of business transactions can only be captured if the specific business agents
associated with the involved events and actions are explicitly represented in the information system in addition to passive business objects.

Current information system technologies do not support the concept of an agent: no matter if the customers of an enterprise are represented in a relational or in an object-relational database table, they are not explicitly represented and treated as agents but rather as objects in the same way as items or bank accounts. E.g., UML [26], the current object-oriented modeling standard, does not support the concept of an agent as a first class citizen. In UML [26], agents are only considered as “actors” that are involved in “use cases” but remain external to the system model. Both the customers and the suppliers of a company would have to be modeled as UML objects in the same way as currencies and bank accounts. UML treats the dynamic aspects of an application system by providing a multitude of process modeling diagrams largely unrelated with each other and with the object class diagram of the system.

In this paper, we investigate the combined application of agent-oriented modeling and of business rule modeling by using the example of a car rental company where customers make rental reservations via Internet, and pick up and drop off cars by using chip cards. The geographical distribution of such an enterprise over a headquarter and a number of branches suggests to view and model it as a group of interacting agents represented by their respective information systems.

2. Notations Used For Enterprise Modeling

2.1 Agent-Object-Relationship Diagrams

Agent-Object-Relationship (AOR) diagrams were proposed in [8,9] as an agent-oriented extension of Entity-Relationship modeling. In AOR modeling, an entity is either an event, an action, a claim, a commitment, an agent, or an object. Only agents can communicate, perceive, act, make commitments, and satisfy claims. Objects do not communicate, cannot perceive anything, are unable to act, and do not have any commitments or claims. An organization is viewed as a complex institutional agent defining the rights and duties of its internal agents (or subagents) that act on behalf of it, and being involved in a number of interactions with them and with external agents. Communication is viewed as asynchronous point-to-point message passing. The expressions 'receiving a message' and 'sending a message' are taken as synonyms of 'perceiving a communication event' and 'performing a communication act'.

In addition to the two designated relationship types specialization and composition of ER/OO modeling, there are twelve designated relationships in which specifically agents, but not objects, participate. Six of them relate agents with events and commitments: an agent perceives environment events, receives and sends messages, does physical actions, is-committed-towards and has-claims-against other agents. The remaining six of these designated relationships associate subagents with particular rights and duties: a subagent may have the right-to-do an action, the right-to-send a
message, the *duty-to-respond* to a message, the *duty-to-react* to a physical event, the *duty-to-fulfill* a commitment, and the *duty-to-monitor* a claim.

Some of the AOR modeling concepts are indexical: taking the perspective of the agent to be modeled, actions of other agents are viewed as events, and commitments of other agents are viewed as claims against them. Likewise in the case of an organization: only the actions of the organization itself and of its subagents count as actions, while the actions of external agents count as events.

Figure 1. The AOR model of the car rental company from an objective observer’s point of view
Recall that entity types are visually represented by rectangles while relationship types are represented by connection lines (possibly with crows feet endings in order to indicate multiplicity). In AOR diagrams, a subclass is visualized as a rectangle within its superclass. A component class is visualized as a rectangle with dotted lines drawn within the superior class it belongs to (recall that a component cannot exist independently of the whole; if the whole ceases to exist, all of its components also cease to exist).

An agent class is visualized as a rectangle with rounded corners. In order to distinguish an internal agent (subagent) class from an external agent class and from an agent subclass, it is visualized by such a rectangle with a dotted line (like CarHandlingAgent in Figure 1).

Both actions and events may be communicative or physical. Events have a concave (incoming) rectangle side, while actions have an convex (outgoing) rectangle side. Communication event rectangles and communication act rectangles have a grey background color.

In the perspective of an organization, commitments are commitments towards other agents, while commitments of other agents are viewed as claims against them. A commitment towards another agent (such as a commitment towards a customer to provide a car) is coupled with the associated action (such as a provideCar action). It is visualized as a rectangle with a dotted line on top of the associated action rectangle like shown in Figure 3. A claim against another agent (such as a claim against a customer to return a car) is coupled with the associated event (such as a returnCar event). It is visualized as a rectangle with a dotted line on top of the associated event rectangle like shown in Figure 6.

Since the state of an entity can be interpreted as a subclass of the entity (see e.g. [6]), we use the notation for subclasses also for representing states. For example, an entity of the class RentalOrder in Figure 1 can be in the state reserved, allocated, effective, or dropped-off. States can also have substates, like in Figure 1 the state present of CarForRental has the substates available, requires-service, and scheduled-for-service.

The AOR model of the car rental company from an objective observer’s point of view is represented in Figure 1.

2.2 Ross Notation

AOR diagrams can be complemented by depicting intensional predicates (derivation rules) evaluated by the agents in the course of business processes. In Figure 2 the Ross Notation [10] is used for representing derivation rules. The Ross Notation enables to represent both materialized (i.e. instantiated) and computed-on-the-fly views of intensional predicates. According to the Ross Notation, each rule consists of an anchor, rule symbol, and correspondent. Anchor is a data type or another rule for whose instances a rule is specified. In the graphical representation of the Ross Notation, the anchor connection exits the anchor and enters the rule symbol. Correspondent is a data type, another rule, or action whose instances are subject to the test exercised by the rule. In the graphical representation of
the Ross Notation, the correspondent connection *exits* the rule symbol and *enters* the correspondent. Both the anchor connection and correspondent connection are dashed.

Every rule produces a value, called the *Yield Value* (abbreviated YV), at any point of time. Usually this value is hidden. It is used internally by the rule to achieve the appropriate truth value for the rule. Sometimes, rules require testing the Yield Value of a rule directly. To satisfy this need, the Yield Value of a rule may be externalized. When externalized, the Yield Value appears as an attribute type for the rule itself.

The symbols of the Ross Notation, used in our model of the car rental company, and their basic meanings are given in Table 1. They are used for representing the derivation rules in Figure 2.

The derivation rule D1 in Figure 2 subtracts 12 hours from the *pick-up-time* of a RentalOrder in the state *reserved*, and copies the yielded value to the attribute *allocation-time* of the RentalOrder.
The derivation rule D2 determines that the rental rate of a RentalOrder, expressed by its attribute rental-rate, is copied from the rental rate of the CarGroup that the car allocated for the RentalOrder belongs to.

The derivation rule D3 defines how to determine the set of cars that are available to rent. This rule determines that a given car CarForRental in the state present is available if and until it is not allocated for any RentalOrder, doesn’t require service (i.e. does not belong to the subclass requires-service of present), and is not already scheduled for service (i.e. does not belong to the subclass scheduled-for-service of present).

The derivation rule D4 determines that the return value of the intensional predicate get-available of the object class CarGroup is equal to the instance of CarForRental in the substate available with the minimal value of the attribute mileage. The RCOP component rule reduces the scope of the overall rule to the CarGroup of the given RentalOrder (see also the explanation for RCOP in Table 1).

The derivation rule D5 determines that if an instance of CarForRental is assigned to an instance of RentalOrder, the state of the RentalOrder changes to allocated, and stays that for exactly as long as this relationship persists.

The rule D6 is a derivation rule prescribing that a customer belongs to the class has-car if and until any RentalOrder related to it is in the state effective.

The derivation rule D7 says that if the mileage since the last service of a car physically present at the branch, represented by the value of the attribute mileage-since-last-service of CarForRental in the state present, is greater or equal than 10,000 km, the substate of the corresponding instance of CarForRental changes to requires-service, and stays that until the car is scheduled for service (because the substates requires-service and scheduled-for-service are mutually exclusive).

Since the Ross Notation does not allow for graphical modeling of intensional predicates whose values depend on the values of parameters, such intensional predicates should be represented textually rather than graphically. In our case study, the intensional predicate has-capacity of the object class CarGroup determines the existence of rental capacity in the given CarGroup during the requested rental period at the time of making a rental reservation. Therefore the truth value of this predicate is also dependent on the value of the parameter rental-period.

AOR diagrams together with derivations rules expressed by the Ross Notation and textually form a graphical representation of an ontology of the problem domain. A partial representation of the ontology of car rental is depicted in Figure 2.

---

1 A problem-oriented ontology is a description by truth values of the concepts and relationships of the problem domain that exist for an agent or more commonly for a community of agents [24].
Table 1. A selection of rule symbols and other graphical symbols according to the Ross Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Basic meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Given an instance of the anchor, do instances of all the correspondent types simultaneously exist for that instance?</td>
</tr>
<tr>
<td>GE</td>
<td>Is the value of the anchor greater or equal than the value of the correspondent?</td>
</tr>
<tr>
<td>SUB</td>
<td>The Yield Value produced by the rule is the subtraction of the values of the correspondents</td>
</tr>
<tr>
<td>EA</td>
<td>Creates an instance of the correspondent</td>
</tr>
<tr>
<td>REA</td>
<td>Creates an instance of the correspondent, but does not materialize it (i.e. terminates such an instance when the instance of the anchor is deleted)</td>
</tr>
<tr>
<td>COP</td>
<td>Requires propagation (i.e. copying) of the value of an instance of the anchor to instance(s) of the correspondent.</td>
</tr>
<tr>
<td>RCOP</td>
<td>Same as COP, except that reverses the value of instance(s) of the correspondent upon deletion of the instance of the anchor, if ever. <em>If the correspondent is another rule, forces interpretation of this other rule exclusively across instances of the copier’s anchor</em></td>
</tr>
<tr>
<td>Negation</td>
<td></td>
</tr>
<tr>
<td>Attribute type</td>
<td></td>
</tr>
</tbody>
</table>

3. Representing Business Rules

3.1 The Nature and Classification of Business Rules

*At the business level, a business rule* is defined as a statement that defines or constrains some aspect of the business [11]. A business rule is based on a business policy. An example of a business policy in a car rental company is "only cars in legal, roadworthy condition can be rented to customers" [11]. A business rule is also subject to one of the following enforcement levels: *mandate* (must be followed), *requirement* (may be deviated from only with permission), and *guideline* (suggestion) [4]. Many
business rules are of a declarative nature: they describe certain states of affairs that are either required or prohibited while not prescribing the steps to be taken to achieve the transition from one state to another, or the steps to be taken to prohibit a transition [11].

Alternatively, a business rule may be defined as a law or custom that guides the behaviour or actions of the actors connected to the organization [12]. We view all actors connected to the business, which can be individuals, organizational units, software systems, or external units like customers or suppliers, as agents and assign actions to them. We view an agent’s action in a broader sense as something that the agent does: a human may make a decision, an agent wrapping a database may execute certain retrieval primitives, a statistical computation agent may run certain mathematical procedures, and one agent may send a message to another agent [1]. Consequently, business rules define and constrain agents’ actions. Actions consume and affect different resources, including information resources.

Examples of business rules from the problem domain of car rental are:
1. A car is available for rental when it is physically present, is not assigned to any rental, and is not scheduled for service.
2. When receiving from a customer the request to reserve a car of some specified car group, the branch checks with the headquarter to make sure that the customer is not blacklisted.
3. Transferring a car to the automotive service station requires that the car has been scheduled for service and commits the automotive service station to return the car after completing the service.

Rule 1 defines the conditions how to determine the set of cars that are available to rent. It can be naturally represented as a derivation rule that may be applied either top-down to compute answers on the fly, or bottom-up (like a “production rule”) to compute a materialized view. Rule 2 defines how to proceed when some event (a reservation request) occurs. It corresponds to a reaction rule. Finally, rule 3 defines the conditions under which some action (transferring a car to the automotive service station) may be performed, and the effects of its actual performance. It corresponds to an action rule. Notice that reaction rules are triggered by the occurrence of specific events, and thus represent automated business process steps, while action rules are applied when an agent decides to perform an action of that type.

### 3.2 Mapping Business Rules to Operational Rules of Agents

Business rules, being of a declarative nature, can in principle also be declaratively represented as functions between different knowledge states of an agent and accordingly implemented by using e.g. the BDI agent architecture [18, 21]. The biggest benefit of such an approach lies in a small gap between the formal specifications and actual implementations of the rules. However, if we want to create an effective agent system, business rules should be operationalized in order to facilitate their implementation. According to the thesis [3], in practice very few, if any, declarative implementations for industrial use live up the criterion of effectiveness.

We have chosen to map business rules to action and reaction rules of the vivid agent architecture of [5] because of the relative straightforwardness of this kind of mapping and the effectiveness of the
vivid agent model needed for business applications in comparison with other candidate agent architectures like e.g. the BDI-architecture [18, 21] mentioned above. Following the work [7], we define an agent to be consisting of three components:

- a virtual knowledge base \( 2X \), consisting of the agent’s beliefs;
- an event queue \( EQ \), i.e. a buffer receiving messages from other agents or from perception subsystems running as concurrent processes;
- a set of action rules \( AR \) and reaction rules \( RR \) respectively determining the agent’s proactive and reactive behaviour.

Action rules have the general form of \( \text{Action} \leftarrow \text{Condition} \) where \( \text{Condition} \) refers to the agent’s information state represented in its VKB. According to the actions prescribed by action rules, action rules are divided into [5]:

- epistemic action rules of the form \( \text{Eff} \leftarrow \text{Cond} \) where \( \text{Eff} \) is an epistemic effect formula specifying a corresponding update of the agent’s VKB;
- physical action rules of the form \( \text{do}(\alpha), \text{Eff} \leftarrow \text{Cond} \) where \( \text{do}(\alpha) \) calls the procedure \( \alpha \) affecting some actuators available to the agent;
- communicative action rules of the form \( \text{sendMsg}[m(c), i], \text{Eff} \leftarrow \text{Cond} \) where \( \text{sendMsg}[m(c), i] \) is a procedure call to send the message \( m(c) \) to agent \( i \).

Agents communicate in some high-level agent-communication language (ACL) that is based on typed messages such as “ASK”, “TELL”, “REQUEST”, and “PROPOSE”. In contrast to the application-specific messages in OO-programming, ACL message types are application-independent and therefore, in combination with an ontology, defining the semantic vocabulary of a problem domain, allow for true software interoperability [7].

Reaction rules encode the behaviour of an agent in response to perception events created by the agent’s perception subsystems, and to communication events created by communication acts of other agents. Both perception and communication events are represented by incoming messages of an agent [7].

There are three types of reaction rules [7]:

- epistemic reaction rules of the form \( \text{Eff} \leftarrow \text{recvMsg}[m(c), j], \text{Cond} \) where the event condition \( \text{recvMsg}[m(c), j] \) is a test whether the event queue \( EQ \) of the agent contains the message \( m(c) \) sent by agent \( j \);
- physical reaction rules of the form \( \text{do}(\alpha), \text{Eff} \leftarrow \text{recvMsg}[m(c), j], \text{Cond} \);
- communicative reaction rules of the form \( \text{sendMsg}[m'(c'), i], \text{Eff} \leftarrow \text{recvMsg}[m(c), j], \text{Cond} \).

Additionally there are derivation rules of the form \( \text{Conclusion} \leftarrow \text{Premise} \) which define intensional predicates in the agent’s virtual knowledge base [7]. They are described by the ontology of the problem domain (v. Figure 2).

---

\(^{2}\) An agent’s virtual knowledge base (VKB) is called “virtual” because it is not necessarily implemented as a classical knowledge base.
Table 1 shows how business rules of the Examples 1-3 from section 3.1 can be respectively mapped to the derivation, reaction, and action rule of the vivid agent model.

While reaction rules are triggered by events, thus representing automated business process steps performed by an enterprise information system (or, for instance, by an automated teller machine as a subagent), action rules represent process steps recorded in the enterprise information system but performed by human agents.

Table 2. Correspondences between business rules and their formal representations by means of derivation, reaction and action rules

<table>
<thead>
<tr>
<th>Business Rule</th>
<th>Operational Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 available(x)</td>
<td>[\text{CarForRental}\text{.present (x)} \land \neg\exists y (\text{RentalOrder}(y) \land y\text{.CarID} = x) \land \neg\text{CarForRental}\text{.requires-service (x)} \land \neg\text{CarForRental}\text{.scheduled-for-service (x)}]</td>
</tr>
<tr>
<td>2 sendMsg (ASK-IF (blacklisted (customer)), HeadquarterAgent)</td>
<td>[\text{recvMsg (request (reserve (car-group rental-period)), customer)}]</td>
</tr>
<tr>
<td>3 do (sendCarToService (x, AutomotiveServiceAgent)), claim (returnCar (x), AutomotiveServiceAgent)</td>
<td>[\text{commit (CarForRental}\text{.scheduled-for-service (x), service-period)}]</td>
</tr>
</tbody>
</table>

3.3 Modeling Action and Reaction Rules by AOR Diagrams

In AOR diagrams, a (re)action rule is visualized as a named circle with incoming and outgoing arrows. The incoming arrows start from the graphical symbols representing the triggering event of a rule and the epistemic conditions to be evaluated. The epistemic effects of a rule are visualized as update arrows from the circle representing the rule to the entities or their specific (sub)states affected. The communicative and physical effects of a rule are represented as arrows from the rule symbol to the symbols representing communicative and physical actions. For example, the triggering event of the rule R1 in Figure 3 is the reception of the reservation request message, the condition to be checked is \text{has-capacity (rental-period)}, and the communicative effect is sending the query message with the content \text{?blacklisted (customer)}. The mental effect caused by the rule R2 in Figure 3 is the creation of a \text{RentalOrder} in the state \text{reserved}.

3.4 Multi-Perspective Modeling of Business Processes

Business rules define and control business processes. A \textit{business process} can be defined as a collection of activities that takes one or more kinds of input, and creates an output that is of value to the customer [13, 25]. A business process describes from start to finish the sequence of events required to produce the product or service [13]. Business processes typically involve several different autonomous units of an organization. Often business processes also cross organizational boundaries.
Business processes can be modeled from an objective observer’s point of view like, for example, described in [13]. However, in this paper business processes are modeled from the perspectives of different agents (resp. actors) involved in them, that is, we take the design perspective. We model a business process by a set of related reaction and action rules representing single process steps. The business processes of the car rental company to be modeled are those of rental reservation, allocating a car for a rental order, picking up a car, dropping off a car, and scheduling a car for automotive service. Because of the lack of space, we have omitted from this paper the business processes of paying for a rental and transferring a car from one branch to another.

The reaction and action rules defining the above-mentioned business processes are described below along with the AOR diagrams modeling them. In the diagrams reaction rules are denoted by $R_a$ and action rules by $A_a$. The reaction and action rules described make use of the derivation rules of the ontology represented in Figure 2.

In the modeling of reaction rules, we have omitted the reaction rules describing the standard behavior for answering queries with the content like $?blacklisted (customer)$ and $?has-car (customer)$.

In Figure 3 the business process of rental reservation is represented from the perspective of a Branch Agent. It contains the following reaction rules:

R1. Upon receiving from a Customer the request to reserve a car of some specified CarGroup for some specified rental period, if that CarGroup has enough rental capacity during the requested rental period (found by evaluating the intensional predicate has-capacity (rental-period) of
CarGroup), the Branch Agent sends a query to the Headquarter Agent to make sure that the Customer is not blacklisted (see also Operational Rule 2 in Table 2);

R2. Upon receiving from the Headquarter Agent a reply telling that the Customer is not blacklisted, the Branch Agent creates the corresponding rental reservation (i.e. an instance of RentalOrder in the state reserved), commits towards the Customer to provide a car, sends to its subagent Timer Agent a request to remind about the allocation time of a car for the given RentalOrder, computed and assigned to the attribute allocation-time of the RentalOrder by the derivation rule D1 (v. Figure 2), and sends an acknowledgement to the Customer.

Figure 4 models the business process of car allocation for advance reservations from the perspective of a Branch Agent. This business process is defined by just one reaction rule:

R3. When the allocation-time of a RentalOrder arrives, the Branch Agent receives from the Timer Agent a reminder to allocate a car for the given RentalOrder, and if there is an available car of the specified CarGroup, expressed as the return value of the intensional predicate get-available () that, in turn, makes use of the derivation rules D3 and D4 as shown in Figure 2 (see also Operational Rule 1 in Table 2), this car is assigned to the RentalOrder by creating the corresponding relationship between the RentalOrder and CarForRental.

Figure 4. The AOR model of the business process of car allocation for advance reservations from the perspective of a Branch Agent
The business process of picking up a car from the perspective of a Branch Agent is represented in Figure 5. It consists of the following reaction rules:

R4. Upon receiving from a Customer a pick-up-request referring to some RentalOrder, the Branch Agent first makes sure that the Customer does not already have a car rented from any branch of the company by asking that from the Headquarter Agent;

R5. Upon receiving from the Headquarter Agent a reply confirming that the Customer does not have a car rented from any branch of the company, the Branch Agent provides the car (and the customer respectively picks up the car), changes the state of the corresponding instance of CarForRental to picked-up, and informs the Headquarter Agent about the new effective RentalOrder (i.e. a RentalOrder where the car has been picked up).

![Diagram of the AOR model of the business process of picking up a car from the perspective of a Branch Agent](image)

Figure 5. The AOR model of the business process of picking up a car from the perspective of a Branch Agent

Figure 6 depicts the business processes of picking up a car and dropping off a car from the perspective of the Headquarter Agent. The reaction rules represented in Figure 6 are:

R6. Upon receiving from a Branch Agent the message about the new effective RentalOrder, the Headquarter Agent inserts into its VKB the corresponding instance of RentalOrder in the state effective (as a result of which the derivation rule D6 changes the state of the Customer to has-car, see Figure 2), and inserts a claim against the Customer to return the car;

R7. Upon receiving from a Branch Agent a message telling that the car of the given RentalOrder has been dropped off, the Headquarter Agent changes the state of the corresponding instance of RentalOrder to dropped-off.
And finally, the business processes of dropping off a car for automotive service from the perspective of a Branch Agent are represented in Figure 7. The rules of these business processes are:

R8. When the Customer drops a car off at the branch, then:
- the Branch Agent informs the Headquarters Agent about the drop-off;
- an instance of CarForRental in the state present is created for that car, or if pick-up-branch = drop-off-branch, the state of the corresponding instance of CarForRental is changed from picked-up to present;

R9. When the Customer drops a car off at the branch, then if the car requires service (i.e. the corresponding instance of CarForRental is in the substate requires-service, determined by the derivation rule D7 depicted in Figure 2), the request to schedule the car for service is sent to the Automotive Service Agent;

R10. Upon receiving from the Automotive Service Agent the automotive service confirmation, the Branch Agent changes the state of the corresponding instance of CarForRental to scheduled-for-service, and inserts the commitment to send the car to service;

A1. In order to fulfill the sendCarToService commitment, the human subagent Car Handling Agent of the Branch Agent sends or takes the car himself to the Automotive Service Agent for service (see also Operational Rule 3 in Table 2) which results in the change of the state of CarForRental from scheduled-for-service to in-service and in the insertion of the claim against the Automotive Service Agent to return the car.
Figure 7. The AOR model of the business processes of dropping off a car and scheduling a car for automotive service from the perspective of a Branch Agent

4. Related Work

In the paper [22] a general methodology for agent-oriented analysis and design is presented. The proposed methodology deals with both the macro-level (societal) and the micro-level (agent) aspects of systems. In the analysis phase of the methodology, the roles in the system are identified and the patterns of interaction that occur in the system between various roles are recognized. The functionality of each role is defined by its liveness and safety responsibilities. Liveness responsibilities are those that say “something will be done”, e.g. “whenever the coffee machine is empty, fill it up”. Safety responsibilities relate to the absence of some undesirable condition arising, e.g. “the coffee stock should never be empty”. In the design phase, the liveness and safety responsibilities are respectively
mapped to agents’ services and pre- and postconditions on each service. Liveness and safety responsibilities thus bear a close resemblance to business rules. The difference from our work is that the methodology proposed in [22] is a software engineering approach, while our approach is aimed at creating business information systems. Another important difference is that while [22] has adopted an objective observer’s point of view in modeling agent systems, the AOR modeling enables modeling from the perspectives of different agents involved.

In the work described in [15] agents are directly applied to managing business processes. The main difference from our work is that [15] focuses on the interaction and negotiation aspects of business processes, and does not explicitly treat conceptual models of the problem domain, and agents’ beliefs and (re)actions.

The paper [19] also concentrates on the interaction aspects of agents in the domain of integrated supply chain management, and particularly on the agents’ mutual obligations and interdictions.

Conceptual modeling of the problem domain is included in the paper [23] where concepts and relations between concepts are defined in hierarchies and rules that are used for automatic generation of prototype agent applications directly from their specifications. The latter is also one of our future intentions.

As was already mentioned in section 1, object-oriented approaches such as described in [13], [25], and [26], do not support the concept of an agent, and are therefore not relevant to be discussed here.

5. Conclusions and Future Work

Agent-oriented concepts allow the seamless integration of business rule modeling and information modeling. Our approach is based on the rather well-developed methodology of capturing information systems’ requirements in the form of business rules (see e.g. [10, 11, 20]). Implementation of business rules has been traditionally connected to (active) databases [16]. We have widened the sphere of applying business rules by showing that they can also be interpreted and implemented as a combination of action and reaction rules, and of derivation rules associated with the ontology of the problem domain.

Additionally, we have empirically proved that the claim in [14] according to which dynamic integrity constraints cannot be represented through a visual formalism is not entirely true. In particular, the conditional parts of dynamic integrity constraints can be represented by using the Ross Notation, while the AOR modeling enables to represent their event and action parts. However, the Ross Notation does not allow for graphical modeling of intensional predicates whose values depend on the values of parameters. The AOR modeling proposal in its present stage leaves several issues unanswered. Although [9] provides a sketch how an AOR model can be transformed into an object-relational database schema, the logical and operational semantics of AOR models is not yet sufficiently established. In particular, the semantics of the deontic concepts of AOR modeling seems
to be a challenging research issue. Furthermore, the relationship of AOR diagrams to the process modeling concepts of UML needs to be investigated.

We think that agents are well-suited to be used in cooperative information systems [2] where both data and application logic are distributed like e.g. in our experimental information system of car rental. We hope our work to be a step from the currently predominant client/server systems [17] towards the peer-to-peer systems of the future.

Our present models represent just positive scenarios, and do not address the cases where something goes wrong, like e.g. when a customer does not appear to pick up a car as agreed, or when the automotive service station fails to return a car on time. We plan to introduce models for exceptions in our future work.

Our other future aims include further formalization, verification, and validation of our work. Another important aim is to work out the environment that would enable semiautomatic generation of object-oriented implementations of agent-oriented business information systems from their high-level descriptions by graphical agent-oriented models. Such an environment should also enable interactive visualizing of graphical models. Since many business rules in real life are essentially of a “fuzzy” nature, we plan to introduce fuzzy business rules for agents. We also plan to introduce modeling-time consistency checks for the rules, commitments, and claims of an agent and deal with the delegation of commitments between agents.

6. References