A New Approach to the Modeling, Design, and Implementation of Business Information Systems

Kuldar Taveter
Department of Informatics, Tallinn Technical University, Tallinn, Estonia
kuldar@cc.ttu.ee

Boris Tamm
Cybernetica AS, Tallinn, Estonia
btamm@is.cyber.ee

Abstract
We are proposing an approach where the modeling, design, and implementation of business information systems is based on the notion of agent. By using an example of car rental, we show how the process of agent-oriented modeling and design could look like. It starts from modeling of autonomous functional units of an organization by agents, and continues by analyzing business rules of the organization and modeling a common understanding of the problem domain for the agents by creating an ontology. We show how business rules can be further expressed through beliefs and reaction rules of the agents, and how business processes can be described through repetitive execution of reaction rules and high-level communication between the agents.

Keywords: agent, ontology, information system, business modeling, business rule, business process

1. Introduction

Agent is an emerging abstraction that the field of business information systems may also benefit from. Agent is understood as an active entity, possessing the features of autonomy, proactiveness, responsiveness, and social behaviour in contrast to a passive entity – object [5]. Agents thus promote autonomous action and decision-making which enables peer-to-peer interaction, while objects are better suited to the more rigid client-server model [26]. Agent-Oriented is therefore highly relevant for business information systems, because business processes are essentially interactions of human or artificial agents.

According to the emerging paradigm of cooperative information systems [2], information systems are viewed as consisting of agents who relate to each other as a social organization. Agents cooperate when they share goals and work together to fulfill those goals. The process of requirements’ capture for cooperative information systems includes the steps of identifying the agents in the environment, mapping business goals to system goals, and operationalizing these system goals by reducing them into constraints that agents can be responsible for through their actions [2]. We have followed these steps in our work. Under the paradigm of cooperative information systems, once captured organizational objectives and systems requirements must also be “kept alive” and remain a part of the running system, due to ongoing evolution of the system. An adequate objectives’ and requirements’ representation language should support a declarative style of specification which offers the possibility to model requirements adopting an aerial view perspective. For example: “the borrowing of a book should be followed by its return within the next three weeks” [2]. In our opinion representing organizational objectives and systems requirements as business rules is a step towards this kind of language.

The goal of our approach is to integrate and develop different modeling techniques to be used in the modeling and design of agent-oriented information systems.
2. Traditional Business Modeling

Traditional business modeling, including object-oriented modeling, can be described by the so-called “triangle” model [7]. The “triangle” consists of three interrelated components: data model (class hierarchy) of the problem domain, relevant events taking place in the external world, and actions of the system that are triggered by the corresponding events and operate on the instances of entities (classes) of the data model (v. Figure 1).

![Figure 1. The "triangle model"](image)

The main shortcoming of the “triangle”-based business modeling, including object-oriented modeling, is that it doesn’t explicitly deal with actors that perform different functions of the business. For example, the DFD diagram, which is one of the most popular ways of modeling the vertex of functions of the “triangle” model [7], views functions as transformers of data flows that are not attached to any actors performing these functions. True, on workflow diagrams functions are attached to actors, but these actors do not belong to the model of the information system. Actors are usually represented as instances of the entities of the data model, but there they are passive objects to be manipulated with rather than true active performers of business functions. This is reflected by the existing methodologies of modeling and designing object-oriented systems like e.g. UML [3]. It has also been noticed by the others [4] that in UML actors are only considered in “use cases”, but remain external to the system model. Due to this business rules remain up in the “air” and are not attached to any processors/executors.

On the other hand, business rules have a global nature, i.e. they possibly involve objects of several object types. This doesn’t fit with the principle of encapsulation that we have in object-oriented modeling. For example, the rule “product of the type A should never be cheaper than product of the type B” involves two different object classes: it cannot be expressed within just one class. The rule “when the payment of a bill is two weeks overdue, it is required to send a reminder to the customer” involves several object classes, an action, and time, and cannot therefore be encapsulated within one specific class [36].

There have been proposals to express business rules in an object-oriented fashion by using metamodeling [8, 9] but we are not aware of their any further consequences. The rule-modeling approach described in [10] and [11] enables to model global business rules in a natural way. This approach is, however, data-centered, and doesn’t therefore include any semantics for actions. We have widened this approach by viewing data as agents’ beliefs, and expressing business rules in terms of agents’ beliefs and actions. This may constitute a powerful paradigm for the modeling, design, and implementation of business information systems. Followingly we will take a closer look at the approach proposed by us.
3. The Business Rules’ Approach

From the modeling point of view, we apply the metaphor of an agent as a natural and convenient antropomorphic abstraction of a functional business unit/actor and also of an external unit/actor like a customer or supplier. From the functional and technical point of view, we see a software agent in business information systems as a rule-based intelligent distributed unit that implements business logic. We are interested in how agents understood this way could be utilized in business information systems, and how they could facilitate the modeling, design, and development of such systems.

3.1 Metamodel

We have worked out a preliminary methodology how information systems could be modeled on a high level of abstraction by expressing business rules as a combination of events perceived by agents, and beliefs and actions of the agents where the events and beliefs respectively constitute triggering conditions and preconditions for the actions. This methodology consists of modeling functional organizational units by agents, analyzing the business rules of the organization, modeling the problem domain by ontology, mapping the business rules to the reaction rules of agents, and modeling business processes from the perspectives of different agents by using reaction rules and from an objective observer’s point of view through agents’ interactions. These steps should be applied in an iterative manner. The metamodel depicted in Figure 2 will be referred to throughout this paper.

In evaluating the methodology proposed by us, we have used the case study of a car rental company where customers make rental reservations via Internet, and pick up and drop off cars by using chip cards (v. Figure 3). 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.

![Figure 2. The metamodel of the business rules’ approach](image-url)
3.2 Organization Modeling

Following the definition presented in [34], we consider an organization as a social unit which lastingly strives to achieve common goals, and has a formal structure which coordinates the activities of all its members in order to achieve the goals.

According to our methodology, firstly functional organizational units of the organization to be modeled are found. A functional organizational unit (v. Figure 2) can be defined as an entity for managing the performance of activities to achieve one or more goals of the organization [29]. Each functional organizational unit maintains knowledge about a certain functional subfield of the problem domain that has common business rules and goals [14]. In our example case study of car rental, such functional organizational units are the branch, headquarter, and automotive service station. Each functional organizational unit can be represented by one or more organization-agents (v. Figure 2). An organization-agent is an individual processing entity that may be of human or automated nature [28]. Each functional organizational unit defines a set of roles that are played by the organization-agents representing the given unit (v. Figure 2), or their subagents. In the example of car rental, the roles “rental reservation manager”, “manager of handovers and returns”, and “manager of walk-in rentals” are played by the automated Branch Agent (v. Figure 3) himself, while the role of “managing automotive service” is played by the human subagent called Car Handling Agent. Each role contains one or more prototypical job functions - tasks - in an organization (v. Figure 2). For example, in a car rental company the tasks corresponding to the role “rental reservation manager” are “checking the capacity of the pick-up branch on the pick-up day”, “checking that the customer is not blacklisted”, “creating a rental reservation”, and “allocating a car for the rental reservation”. Organization-agents interact with each other with respect to the roles played by them (v. Figure 2). Each task consists of one or several actions. The role containing a task has an authority to perform the actions the given task consists of (v. Figure 2). An action is an atomic unit of work done by an organization-agent. Actions are controlled by business rules. 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], while in the “triangle” model (v. Figure 1) action is understood narrowly as something that changes the state of a data object. Example actions in our car rental company are sending a car to service, asking another agent to transfer a car, and creating an instance of the RentalOrder.

Other organization-agents besides Branch Agents in our example are the Automotive Service Agent representing the service station, the Headquarter Agent, representing the headquarter of the car rental company and mediating the database of rental orders and customers, and the Customers themselves (v. Figure 3). The Customers reserve cars via Internet, and pick up and drop off cars by using chip cards that are sent to them by mail. They pay for rentals by using the services of an external institutional agent – Bank (v. Figure 3). Each Branch Agent has his own database of the cars at his disposal. As we will see in section 3.4, agents of all the types mentioned are also explicitly represented in the agent-oriented information system of car rental.
3.3 Analysis of Business Rules

When we take a look at the requirements’ description for some business, we find that the goals of the business are represented in the form of a big number of different rules. Business goals are thus expressed as business rules (v. Figure 2). 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) [35]. 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]. It is a duty of the system analyst to attach these rules to certain roles played by certain organization-agents.

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]. As already explained earlier, we view all actors connected to the business, which can be humans, software agents, or external units like customers or suppliers, as organization-agents and assign actions to them. Consequently, business rules define and constrain the actions of organization-agents. Actions consume and affect different resources, including information resources (v. Figure 2).
At the level of an information system created to support a business, a business rule expresses specific constraints on the creation, updating, and removal of persistent data in the information system [11].

Examples of business rules from the problem domain of car rental are:

1. 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.
2. A car is available to be allocated for rental when it is physically present, is not assigned to any rental, and is not scheduled for service.
3. Upon receiving from a branch the request to authorize a rental, if the customer doesn’t already have a car rented from the car rental company, the pick-up is authorized by the headquarter.
4. The rental rate of a rental is inferred from the rental rate of the group of the car assigned to the rental.
5. Each car must get automotive service after every 10,000 km.
6. Only a branch manager of the “donor” branch may assign a car for transfer to another branch.

Rule 5 is based on the business policy “only cars in legal, roadworthy condition can be rented to customers”, given as an example above. The business rules of Examples 1 – 5 will be used in the subsequent sections of this paper.

Business rules can be divided into authorizations, derivations, and action rules. An authorization defines a specific prerogative or privilege of a role with respect to one or more actions. Derivation is a statement of knowledge that is derived from other knowledge in the enterprise. Action rule specifies that if some condition is true, a certain action should be performed. The rules of Examples 1, 3, and 5 are action rules, the rules of Examples 2 and 4 are derivations, and the rule of Example 6 is an authorization.

3.4 Domain Modeling by Ontology

A common framework of knowledge for the organization-agents is created in the form of an ontology. 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 [15]. An ontology consists of the concepts (classes), relations between them like e.g. subsumption (inheritance), aggregation, and association, and axioms of the problem domain. Ontology should provide all the data structures, relations, and axioms that are necessary for the agents for performing their actions. Ontology should also represent agents themselves. Each agent of the problem domain can see only a part of the ontology, i.e. each agent has a specific view to the ontology.

We used the Ontolingua [15] formalism for creating the ontology of car rental, because the Ontology Editor [16] for Ontolingua enables to check the consistency of an ontology created by its use.

Ontologies can be viewed as extensions of object-oriented (OO) models of problem domains described e.g. in [13]. However, most OO frameworks neither provide the necessary axioms that constrain the interpretation and well-formed use of the terms defined by them, nor support other ontological constructs, such as metamodels (i.e. defining a model by using the model itself) [30].

Ontologies can be represented using different graphical notations. We have chosen to represent the ontology of car rental by using a combination of Agent-Object-Relationship (AOR) diagrams [17, 18] and Ross Notation [10].

A simplified version of the ontology of car rental is represented in Figure 4. In AOR diagrams [17, 18], 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 4).

Since the state of an entity can be interpreted as a subclass of the entity (see e.g. [19]), we use the notation for subclasses also for representing states. For example, an entity of the class RentalOrder in Figure 4 can be in the state reserved, allocated, effective, or dropped-off. States can also have substates, like in Figure 4 the state present of CarForRental has the substates available, requires-service, and scheduled-for-service.

Different conditions and derivations evaluated by the agents are represented by an ontology using the Ontolingua [15] formalism as shown in the second column of Table 2 for the business rules of Example 1 - 5. According to the semantics of Ontolingua, all free variables that appear in the examples have implicit universal quantification.

In Figure 4 the Ross Notation [10] is used for representing the same conditions and derivations. The Ross Notation enables to represent both materialized (i.e. instantiated) and computed-on-the-fly views of derivations. According to the Ross Notation, each business 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 ontology of the car rental company, and their basic meanings are given in Table 1.

<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>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></td>
<td>Negation</td>
</tr>
<tr>
<td></td>
<td>Attribute type</td>
</tr>
</tbody>
</table>
The condition of the business rule of Example 1 simply checks that an instance of Customer does not belong to the subclass blacklist. Its ontological representations in Table 2 and Figure 4 are therefore straightforward.

The business rule of Example 2 is a derivation defining how to determine the set of cars that are available to rent. Rule 2 can be expressed as an axiom of Ontolingua that looks like shown in the second row of Table 2. This axiom determines that a given car ?Car is available if it is physically present at the branch (i.e. belongs to the subclass present of CarForRental), is not allocated for any RentalOrder, doesn’t require service (i.e. does not belong to the subclass requires-service of CarForRental), and is not already scheduled for service (i.e. does not belong to the subclass scheduled-for-service of CarForRental). Figure 4 depicts the visualization of Rule 2 by using the Ross Notation (rule D2).

The condition of Example 3 is a derivation prescribing that a customer belongs to the class has-car if any RentalOrder related to it is in the state effective. It is represented by Ontolingua as shown in the third row of Table 2. With the help of the Ross Notation this derivation can be visualized as shown in Figure 4 (rule D3).

The derivation formulated in Example 4 can be expressed in Ontolingua as shown in the fourth row of Table 2. This rule 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. In Figure 4 this corresponds to the rule of the type copier (v. Table 1) of the Ross Notation (rule D4).

The conditional part of the action rule of Example 5 can be expressed as the axiom of Ontolingua shown in the fifth row of Table 2. This axiom evaluates to true if a car ?Car belongs to the subclass requires-service of CarForRental, i.e. when its mileage since last service, represented by the value of the attribute mileage-since-last-service of CarForRental, is equal or greater than 10,000. The visual representation of this axiom can be seen in Figure 4 (rule D5).

Table 2. Correspondences between business rules, their formal representations in Ontolingua, and operational rules of agents.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Ontological Representation of Condition/Derivation</th>
<th>Operational Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(blacklisted (?Customer))</td>
<td><code>sendMsg</code> (ASK-IF (blacklisted (customer)), HeadquarterAgent) <code>recvMsg</code> (request (reserve (car-group ...)), customer)</td>
</tr>
<tr>
<td>2</td>
<td>(&lt;=&gt; (available ?Car) (And (present ?Car) (Not (Exists (?Rental) (And (RentalOrder ?Rental) (= (Car-Of ?Rental) ?Car) ))) (Not (requires-service ?Car) (Not (scheduled-for-service ?Car))))</td>
<td>available(x) <code>recvMsg</code> (ASK-IF (blacklisted (customer)), BranchAgent) <code>recvMsg</code> (request (reserve (car-group ...)), customer)</td>
</tr>
<tr>
<td>3</td>
<td>(&lt;=&gt; (has-car ?Customer) (Exists (?Rental) (And (effective ?Rental) (^ ?Customer (Customer-Of ?Rental))))</td>
<td><code>sendMsg</code> (REPLY-IF (has-car (customer), no), BranchAgent) <code>recvMsg</code> (ASK-IF (blacklisted (customer)), BranchAgent), ¬ ^Customer.has-car (customer)</td>
</tr>
<tr>
<td>4</td>
<td>(=&gt; (RentalOrder ?Rental) (= (rental-rate ?Rental) (rental-rate (Car-Group-Of (Car-Of ?Car) )))</td>
<td><code>sendMsg</code> (REQUEST (schedule_for_service (car OF rental-ID)), AutomotiveServiceAgent) <code>recvMsg</code> (DROP-OFF-Car (rental-ID), customer), CarForRental.requires-service (car OF rental-ID)</td>
</tr>
<tr>
<td>5</td>
<td>(=&gt; (requires-service ?Car) (And (CarForRental ?Car) (^ (mileage-since-last-service ?Car) 10000)))</td>
<td><code>sendMsg</code> (REQUEST (schedule_for_service (car OF rental-ID)), AutomotiveServiceAgent) <code>recvMsg</code> (DROP-OFF-Car (rental-ID), customer), CarForRental.requires-service (car OF rental-ID)</td>
</tr>
</tbody>
</table>

3.5 Mapping Business Rules to Operational Rules of Agents

At the next step business rules are mapped to the individual agents representing functional organizational units introduced in section 3.2. We have chosen to map business rules to reaction rules and derivations of the vivid agent model of [23] because of the relative straightforwardness of this kind of mapping. Following the works [6, 23], we define an agent to be consisting of three components:

- a virtual knowledge base\(^1\) \(X\), 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 reactive and communicative behaviour.

Agents communicate in some high-level agent-communication language, such as ACL proposed by FIPA [20], 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 [6].

\(^1\) An agent’s virtual knowledge base (VKB) is called “virtual” because it is not necessarily implemented as a classical knowledge base.
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 [6].

There are three types of reaction rules [6]:

- **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 \( j \). \( \text{Cond} \) refers to the agent’s information state represented in its VKB, and \( \text{Eff} \) is an epistemic effect formula specifying a corresponding update of the agent’s VKB;

- **physical reaction rules** of the form \( \text{do}(\alpha), \text{Eff} \leftarrow \text{recvMsg}(m(c), j), \text{Cond} \) where \( \text{do}(\alpha) \) calls the procedure \( \alpha \) affecting some actuators available to the agent;

- **communicative reaction rules** of the form \( \text{sendMsg}(m'(c'), i), \text{Eff} \leftarrow \text{recvMsg}(m(c), j), \text{Cond} \) where \( \text{sendMsg}(m'(c'), i) \) is a procedure call to send the message \( m'(c') \) to agent \( i \).

Additionally there are **derivation rules** of the form \( \text{Conclusion} \leftarrow \text{Premise} \) which define intensional predicates in the agent’s virtual knowledge base [6].

Table 2 shows how business rules of Examples 1 - 5 can be mapped to reaction rules and derivation rules, i.e. to **operational rules**, of vivid agents. The operational rules, corresponding to the business rules of Examples 1 and 3, are reaction rules in response to the communication events originating from other agents, and the operational rule of Example 5 is a reaction rule in response to the environment event \( \text{dropOffCar} \) caused by the customer. The derivations of Examples 2 and 4 are straightforwardly mapped to the corresponding derivation rules of vivid agents.

Events on AOR diagrams [17, 18] 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.

From 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 \( \text{provideCar} \) action). It is visualized as a rectangle with a dotted line on top of the associated action rectangle like shown in Figure 5. 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 \( \text{returnCar} \) event). It is visualized as a rectangle with a dotted line on top of the associated event rectangle like shown in Figure 6.

In AOR diagrams, a reaction rule is visualized as a named circle with incoming and outgoing arrows. The incoming arrows come from the graphical symbols representing the triggering event of a rule and the epistemic condition(s) 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 \( \text{R1} \) in Figure 5 is the reception of the request message \( \text{reserve} \) from the \text{Customer}, the condition to be checked is \( \text{has-capacity (rental-period)} \), and the communicative effect is sending the query message \( \text{?blacklisted} \) to the \text{Headquarter Agent}. The mental effect caused by the rule \( \text{R2} \) in Figure 5 is the creation of a \( \text{RentalOrder} \) in the state \( \text{reserved} \). The physical effect of the rule \( \text{R8} \) in Figure 7 is the execution of the action \( \text{sendCarToService} \).

3.6 Modeling Business Processes

Business rules define and control business processes (v. Figure 2). A **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 [7, 13]. 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 organizational units (v. Figure 2). Often business processes also cross organizational boundaries.

We firstly model business processes from the perspectives of different organization-agents (resp. actors) involved in them. We model a business process by a set of related reaction rules representing single process steps. In order to offer a better overview of business processes, we model them also from an objective observer’s point of view. 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. A selection of business processes of the car rental company modeled from different perspectives follows.

In Figure 5 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 group has enough rental capacity during the requested rental period (found by evaluating the intensional predicate has-capacity (rental-period)), the Branch Agent sends a query to the Headquarter Agent to make sure that the Customer is not blacklisted (v. Operational Rule 1 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, and sends an acknowledgement to the Customer.

![Figure 5. The AOR model of the business process of rental reservation from the perspective of a Branch Agent]

The night before the pick-up day a car is allocated for the rental reservation by executing the corresponding reaction rule where the Branch Agent searches its virtual knowledge base for an available car with the required parameters, and if the car has been found, allocates the car for the reservation. In searching for an available car, the Branch Agent makes use of the derivation available (v. Operational Rule 2 in Table 2).

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:

R3. 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, and inserts a claim against the Customer to return the car;
**R4.** 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*.

In order to save space, we have omitted the reaction rules describing the standard behavior for answering queries like ?blacklisted and ?has-car (v. Operational Rule 3 in Table 2).

![Diagram of the AOR model of the business processes of picking up and dropping off a car from the perspective of the Headquarter Agent](image)

**Figure 6.** The AOR model of the business processes of picking up and dropping off a car from the perspective of the Headquarter Agent

And finally, the business processes of *dropping off a car* and *scheduling a car for automotive service from the perspective of a Branch Agent* are represented in Figure 7. The business rules are:

**R5.** When the *Customer* drops a car off at the branch, then:
- the *Branch Agent* informs the *Headquarter 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*;

**R6.** 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*), the request to schedule the car for service is sent to the *Automotive Service Agent* (v. Operational Rule 5 in Table 2);

**R7.** 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;

**R8.** In order to fulfill the *sendCarToService* commitment, the human subagent *Car Handling Agent* of the *Branch Agent* sends or takes himself the car to the *Automotive Service Agent* for service.

The rule R8 is actually an *action rule* of the form *Action ← Condition* where *Condition* refers to the agent’s information state represented in its VKB. While reaction rules are triggered by events, thus representing automated business process steps performed by an enterprise information system,
action rules represent process steps recorded in the enterprise information system but performed by human agents.

As an example of modeling business processes from an objective observer’s point of view, the process of negotiations of three Branch Agents about transferring a car is depicted in Figure 8. Please note that since agent communication languages are based on well-defined, small sets of general communicative actions, whereas objects communicate through unrestricted and idiosyncratic messages with ad hoc semantics, UML diagrams [3] can only limitedly be used for modeling agent-oriented information systems as it is done in Figure 8.

The requirements for the car rental company contain a business rule according to which if more cars have been requested than are available in a car group at a branch, the branch manager may ask other branches whether they have cars they can transfer to him. This business rule can be automated with the help of agents. The Branch Agent of the branch in need of a car starts the business process by initiating a contract net for performing the action transfer_car with the lowest possible cost (in the interests of the car rental company as a whole). In our example, this is reflected by the Call-For-Proposals message that the Branch Agent I broadcasts to other branch agents (messages 1 and 3). The parameters’ parts of these messages contain the car group and the rental period a car is required for. The Branch Agents II and III, whose branches happen to have a suitable car, respond with the PROPOSE messages where each of them specifies the cost of the transfer of the car from his branch to the branch of the Branch Agent I (messages 2 and 4). The costs that are calculated by the Branch Agents II and III take into account the distance between the candidate for

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
the “donor” branch and the receiving branch and possible losses of the “donor” branch due to the transfer. Since the cost to transfer a car from the branch of the Branch Agent II appears to be lower than the cost to transfer a car from the branch of the Branch Agent III, the proposal of the Branch Agent II is accepted (message 5) without any preconditions on the part of the Branch Agent I (the parameter true of the message). The proposal of the Branch Agent III is accordingly rejected (message 6) by stating that the reason for the rejection is too high cost of transfer. The business process ends with the commitment of the Branch Agent II towards the Branch Agent I to transfer a car of the specified car group by the pick-up time of the specified rental period.

![Diagram of negotiations of branch agents about transferring a car](image)

**Figure 8. Negotiations of branch agents about transferring a car**

### 4. RELATED WORK

We have integrated our approach with an extension of Entity-Relationship modeling, called Agent-Object-Relationship (AOR) Modeling, proposed in [17] and [18] where an entity is either an object, event, action, commitment, claim, or agent. The integration is further elaborated in [37].

In the paper [32] 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 [32] is a software engineering approach, while our approach is aimed at creating business information systems.

In the work described in [21] agents are directly applied to managing business processes. The main difference from our work is that [21] 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 [26] 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 [33] where concepts and relations between concepts are defined in hierarchies. That approach also includes 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 2, object-oriented approaches such as described in [3], [7], and [13], do not support the concept of an agent, and are therefore not relevant to be discussed here.

5. CONCLUSIONS AND FUTURE WORK

The goal of our approach has been to integrate and develop different modeling techniques to be used for modeling and design of agent-oriented information systems. In particular, we have integrated the AOR Modeling proposed in [17, 18] with the rather well-developed methodology of capturing information systems’ requirements in the form of business rules (see e.g. [10, 11, 27]).

Implementation of business rules has been traditionally connected to (active) databases [22]. We have widened the sphere of using and implementing business rules by showing that they can also be interpreted and implemented as a combination of agents’ beliefs and actions. With our approach, agents are viewed as meta-level intelligent entities that implement business rules and thus control the behaviour of data objects.

Based on the case study of car rental that we have used in our work, we can conclude that the approach, where we first model the functional units of the organization, then create the model of the information they deal with, and after that design the ways the agents process the information at their disposal seems to be a pretty natural one from the cognitive point of view. True, further formalization, verification, and validation of our work needs to be done, but these will be important subjects of our future work.

We think that our approach can make the design and implementation of complicated information systems considerably easier by raising the level of abstraction. We believe that just like object-oriented programming has given rise to object-oriented modeling and design, agent-oriented programming, that is programming in terms of agents’ beliefs and actions, may give rise to the proper agent-oriented modeling and design. We hope our work to be a step towards that.

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 [24] towards the peer-to-peer systems of the future.

A major weakness of our work is dealing with interactions of agents. We plan to introduce additional techniques for modeling agents’ interactions and the protocols needed for them.

Like it was mentioned before, our 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 ontologies and reaction rules of agents. Since many business rules in real life are essentially of a “fuzzy” nature, we plan to introduce fuzzy business rules for agents. Our aims also include modeling of business rules for inter-organizational setting and particularly for business-to-business automated e-commerce.

We are also interested in how vivid agents could be implemented by mapping them to some existing agent implementations like e.g. to Jam-agents [31] that are based on the BDI-architecture [25].
6. REFERENCES


