In Information Systems 28:5 (2003).

The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior*

Gerd Wagner

Email: G.Wagner@tm.tue.nl

Homepage: http://tmitwww.tm.tue.nl/staff/gwagner

Eindhoven University of Technology, Faculty of Technology Management

We present an agent-oriented approach to the conceptual modeling of organizations and organizational information systems, called *Agent-Object-Relationship (AOR)* modeling, where an entity is either an agent, an event, an action, a claim, a commitment, or an ordinary object, and where special relationships between agents and events, actions, claims and commitments supplement the fundamental association, aggregation/composition and generalization relationship types of Entity-Relationship (ER) and UML class modeling. Business processes are viewed as social interaction processes emerging from the behavior of the participating agents. In the proposed approach, behavior is primarily modeled by means of interaction patterns expressed in the form of *reaction rules* that are visualized in *interaction pattern diagrams*. It is an option, though, to use UML activity and statemachine diagrams, in addition.

We propose an elaborate conceptual framework for agent-oriented modeling that is based on a set of 19 ontological principles including those of ER modeling, and a corresponding diagram language. In this approach, an organization is viewed as an *institutional agent* defining the rights and duties of its internal agents that act on behalf of it, and being involved in a number of interactions with external (and internal) agents, while an organizational information system is viewed as an artificial internal agent possessing a global view of the organization and interacting both with internal and with external agents on behalf of the organization.

We argue that AOR modeling offers a research perspective to conceptually integrate the static, dynamic and deontic aspects of organizations and organizational information systems.

1 Introduction

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, we propose to make an ontological distinction between active and passive entities, that is, between *agents* and ordinary *objects*. In particular, the semantics of *business processes* may be more adequately 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.

Our work is inspired by the *Agent-Oriented Programming* proposal of [Sho93]. Agent-Oriented Programming is an extension of object-oriented (OO) programming. The two main points of it are:

1. While the state of an object in OO programming has no generic structure, the state of an agent has a 'mentalistic' structure: it consists of *mental components* such as beliefs and commitments.

2. While messages in object-oriented programming are coded in an application-specific ad-hoc manner, a message in Agent-Oriented Programming is coded as a 'speech act' according to a standard *agent communication language* that is application-independent.

In this paper, we attempt to show that the intuitions underlying Agent-Oriented Programming have an even greater potential for information systems engineering than for general software engineering. We develop an agent-oriented approach to the conceptual modeling of organizational information systems, called *Agent-Object-Relationship (AOR)* modeling, where an entity is either an agent, an event, an

action, a claim, a commitment, or an ordinary object, and where special relationships between agents and events, actions, claims and commitments supplement the fundamental association, composition and generalization relationship types of Entity-Relationship modeling. AOR modeling can be viewed as an extension of the Unified Modeling Language (UML). We believe that AOR modeling, by virtue of its agent-oriented categorization of different entity types, allows more adequate models of organizations and organizational information systems than ER modeling and the UML.

There are two basic types of AOR models: external and internal ones. An external AOR model adopts the perspective of an external observer who is observing the (prototypical) agents and their interactions in the problem domain under consideration. In an internal AOR model, we adopt the internal (first-person) view of a particular agent to be modeled. This distinction suggests the following system development path: in the analysis phase, draw up an external AOR model including one or more focus agents as a domain model; in the design phase, for each focus agent, transform the external AOR model into an internal one according to the agent's perspective (this is called "internalization"); then, refine the internal AOR model of each focus agent into an implementation model for the target language (such as SQL or Java). A complete internal AOR model is a formal specification of a high-level state transition system, where perception, reaction and commitment/claim handling provide the basic transition types.

After reviewing the principles of Entity-Relationship modeling and discussing the basic features of agents and the 'agentification' of information systems in Section 2, we discuss the principles of *Agent-Object-Relationship (AOR)* modeling in Section 3. We then introduce the two basic types of AOR models, *external* and *internal* AOR models, in Section 4 and 5, and discuss further behavior modeling elements in Section 6. In Section 7, we briefly consider the issue of tool support. Finally, in Section 8, we review some related work, and point out the strengths and weaknesses of AOR modeling.

2 Agents and Agentified Information Systems

2.1 The Current Information System Paradigms

Current information system technologies are largely based on the Entity-Relationship (ER) metamodel of [Che76] and the Relational Database (RDB) model of [Cod70]. Driven both by the inherent shortcomings of relational databases and the success of the object-oriented (OO) programming paradigm, concepts and techniques from OO programming are now increasingly applied in the area of information systems. However, there is no single generally acknowledged model of object-orientation, and OO programming differs from object-oriented information systems engineering in various respects. In OO programming, all software artifacts are viewed as objects, from GUI push buttons to entire server programs (which also count as *active objects*), while in object-oriented information systems, typical examples of objects are customers, bank accounts and other 'business objects' which are represented in object-relational (i.e. SQL-99) databases. Obviously, these two disciplines do not share a common conceptual space. Concerning database systems, the most important object-oriented features are object IDs, object references, abstract data types (including user-defined base types) with subtype hierarchies (realizing inheritance relationships between ADTs), object-valued attributes, and subtable hierarchies (realizing generalization relationships between entity types).

While ER modeling has always been object-oriented to some degree, through its support of generalization and complex-valued attributes, the RDB model is going to be conservatively extended to the *Object-Relational* Database (ORDB) model as exemplified by a number of research prototypes and commercial systems, and as expressed by the new SQL-99 standard.[1]

Current information system technologies do not support the concept of an agent and the distinction between agents and objects: no matter if the customers of an enterprise are represented in a RDB table, in an object table of an ORDB, or in an object class of an Enterprise Java Beans-based framework, such as IBM's San Francisco, they are not explicitly represented and treated as agents but rather as objects in the same way as rental cars or bank accounts.

2.2 Principles of Entity-Relationship Modeling

Since they form the foundation of information modeling, we restate the ontological principles of Entity-Relationship (ER) modeling:

1. An information system has to represent information about **entities** that occur in the **universe of discourse** associated with its application domain, and that can be uniquely identified and distinguished from other entities.

[1] See [SM96] for a discussion of ORDBs from the SQL programming perspective, and [Wag98] for a theoretical presentation of the ORDB model.

2. Entities have **properties** and participate in **relationships** with other entities.

3. In order to represent entities in an information system, they are classified by means of **entity types**.

4. Each entity type defines a list of (stored and derived) **attributes** that are used to represent the relevant properties of the entities associated with it.

5. Together, the values of all attributes of an entity form the **state** of it.

6. In order to represent ordinary domain relationships (**associations**) between entities, they are classified by means of **relationship types**.

7. There are two designated relationships between entity types that are independent of the application domain: **generalization** (isSubclassOf) and **aggregation** (isPartOf).

In addition to the way generalization is visualized in ER diagrams and UML class diagrams, by means of a special arrow, we also visualize a *subclass* as a rectangle within its superclass, following [Har87]. In terms of visual clarity, this seems especially useful for the graphical rendering of subclasses that are defined by means of a Boolean status attribute, as illustrated in Figure 1.

Figure 1: A library database includes two object types: `Book` and `BookCopy`. Books for which the status predicate `isAvailable` holds (determined by checking if there is at least one copy of that book available) form a subclass of `Book`.

An ER model or a UML class model does not make a distinction between passive and active entities, that is, between objects and agents. Also, there is no distinction between a basic entity type like *Item* and an action event type like *Delivery* in these models.

2.3 Agents

The agent metaphor subsumes both natural and artificial systems. A formal agent concept for the purpose of representing agents in an information system, and for agentifying information systems, may abstract away from many of the higher-level cognitive aspects of human agents. It only needs to capture those aspects that are relevant for realizing the interactions of interest. In an enterprise information system, for instance, only perceptions (in the form of signals), beliefs, memories and commitments associated with business processes are of interest.

There are several approaches to defining agents in the literature, only two of them being relevant for our purposes:

1. The *software engineering* approach emphasizes the significance of application-independent high-level **agent-to-agent communication** as a basis for general software interoperability. E.g., in [GK94], the following definition of agents is proposed: *An entity is a software agent if and only if it communicates correctly in an agent communication language.*

2. The *mentalistic* approach, based on the knowledge representation paradigm of AI, points out that the state of an agent consists of **mental components** such as beliefs, perceptions, memory, commitments, expectations, goals and intentions, and its behavior is the result of the concurrent operation of its perception (or event handling) system, its knowledge system (comprising an update and an inference operation), and its action system (responsible for epistemic, communicative and physical actions and reactions). E.g., in the approach of [Sho93], *an agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.*[2]

According to our ontological distinction between agents and objects, only agents can perceive events, perform actions, communicate, or make commitments. Ordinary objects are passive entities with no such capacities.

This contrasts with the language used in the literature on object-oriented programming, where objects 'communicate' or 'interact' with each other by sending 'messages'. Notice that the UML term 'collaboration' between objects corresponds only to a very low-level sense of communication and interaction. In

[2] Another choice of basic mental components is proposed in the *BDI* approach of [RG91]: *beliefs*, *desires*, and *intentions*. Notice that in both lists of basic mental state components, two important components are missing: *perceptions*, e.g. in the form of incoming messages and signals representing communication and environment events, and *memory* of past events and actions. In fact, although perceptions are temporally not as stable as beliefs, they form the basis of reactive behavior, and are therefore more fundamental than many other mental components such as desires and intentions.

fact, sending a 'message' in the sense of OO programming corresponds rather to a (possibly remote) procedure call, and not to a communication act (or *speech act*): while an OO message has no generic structure at all, a speech act message has the mandatory form $m(c)$ where m is the message type (expressing the 'illocutionary force'), and c is the message content (composed of propositions and/or action terms).

Conceptually, it is therefore not justified to model customers and suppliers as 'business objects' in the same way as bank accounts and software artifacts (such as GUI push buttons). Object-orientation does not capture communication and interaction in the high-level sense of business processes carried out by business agents.

Our view is shared by Jacobson [Jac94] who remarks (p.36) that "it is bizarre to apply the way of thinking that governs computer systems to business processes".

2.4 Information Systems as Agents

The following definition of [HR95] summarizes the most important features of agency: *Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions.*

In the case of an agentified information system,

1. '*perception of dynamic conditions in the environment*' refers to incoming messages representing communication events (such as receiving a request for a sales quotation or an acknowledgment of a sales order) and to incoming signals representing environment events (such as receiving a payment);

2. '*action to affect conditions in the environment*' refers to communication acts of the agentified IS (such as acknowledging a sales order) and to physical acts (such as delivering goods or making a payment);

3. finally, '*reasoning to interpret perceptions, solve problems, draw inferences, and determine actions*' refers to things like the proper processing of incoming messages, the computational inference of correct answers to queries, and the determination of proper actions (such as locking all sales orders of a customer whose credibility is in question or issuing an alert when the fulfillment of a commitment is overdue).

An information system may be explicitly designed as an agent by

1. treating its information items as its *beliefs* or *knowledge*;[3]

2. adding further mental components such as *perceptions* (in the form of incoming messages and signals), *memory*, and *commitments*;

3. providing support for agent-to-agent communication on the basis of a standard agent communication language.

In order to agentify an information system, its schema has to be partitioned: in addition to the tables representing the current state of affairs that form its beliefs, special tables are needed for representing its memory (about past events and actions) and its commitments and claims. For querying the perception state by processing incoming messages a built-in data structure (such as an *event queue*) has to be added. Finally, the *reaction patterns* representing the reactive and communicative behavior of the information system have to be specified, e.g. by means of *reaction rules*. Depending on a triggering event type and possibly on a mental state condition (involving beliefs, memory, commitments and claims), a reaction pattern specifies an action and an associated mental effect that may lead to updates of belief, commitment and claim tables. In this way, an information system turns into a *knowledge-perception-memory-commitment (KPMC) agent.*

2.5 Potential Applications for Agent-Oriented IS Engineering

In principle, any organizational IS that has to represent, and to interact with, agents (such as customers, employees, or other information systems) can benefit from an agent-oriented approach. An agent-oriented approach to IS engineering is most useful for systems that have to support business processes across different organizations and organizational units. Such systems are required, for instance, to manage supply chains or to support virtual enterprises. In these application domains, large distributed information systems are needed where the nodes, representing organizations or organizational units, have to be able to interact with each other as peers. This is the type of interaction that is best captured by an agent-oriented approach, and where the possibility of coordinated co-development of several interacting systems offered by AOR modeling seems to be especially useful.

[3] Unlike in philosophy, it does not make sense to distinguish between *knowledge* and *beliefs* in the design and engineering of agent systems where both terms simply refer to the information that is available to the agent under construction. We will interchangeably use these terms without making any reference or commitment to philosophical theories.

Our considerations are supported by an increasing number of publications on agent-oriented supply chain management (see, e.g., [CPF+99, FBT00, GSP00]) and agent-based IT support for virtual enterprises (see, e.g., [FMHS96, WS00]).

3 Principles of Agent-Object-Relationship Modeling

In this section we introduce a new modeling paradigm: the agent-object-relationship (AOR) metamodel for modeling agent-oriented information systems. As in ER modeling, the purpose is to provide a generic methodology for information systems analysis and design. In the same way as an ER model can be effectively transformed into a relational or object-relational database schema, an AOR model should be transformable into a corresponding database schema. Notice that this implies that the elements of the AOR metamodel must have a formal semantics. We will sketch such a transformation briefly in Section 5.7.

ER modeling does not account for the dynamic aspects of information and knowledge processing systems. These aspects are related to notions like communication, interaction, events, activities and processes. For capturing semantic aspects related to the dynamics of information systems, it is necessary to distinguish between *agents* and passive *objects*. While both objects and agents are represented in the system, only agents *interact* with it, and the possible interactions may have to be represented in the system as well.

The UML does not support the concept of an agent as a first class citizen. In the UML, there is a certain ambiguity with respect to the agent concept. Human and artificial agents, if they are 'users' of a system, are called *actors* being involved in *use cases* but remaining external to the system model, while software agents within the boundaries of the system considered are called 'active objects'. In the UML, the customers and the employees of a company would have to be modeled as 'objects' in the same way as rental cars and bank accounts, while in the AOR approach they would be modeled as institutional or human agents to be represented in the system of that company (which itself could be modeled as an artificial agent).

Since interaction between agents takes place in a social context, deontic concepts such as commitments and claims with respect to external agents, and rights and duties with respect to internal agents, are essential for understanding and controlling coherent interaction between agents and other systems. Neither ER modeling nor UML provide any means to account for the deontic aspects of an information system.

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. Being entities, agents and objects of the same type share a number of attributes representing their properties or characteristics. So, in AOR modeling, there are the same notions as in ER modeling (such as entity types, relationship types, attributes, etc.).

The AOR modeling language (AORML) is based on the AOR metamodel. While ER modeling and UML support the design of *object-oriented* information systems realized with the help of relational and object-relational database technology, AORML is to support the high-level design of *agent-oriented* information systems.

3.1 Ontological Foundations of AOR Modeling

In [WW99], it is convincingly argued that a domain modeling approach should be based on clear ontological principles explaining and justifying the vocabulary employed independently of any specific implementation technique. In laying out the ontological foundations of a modeling method, one may consider both the concepts and terminologies established in the practice of developing and using information technologies and the theories of traditional and contemporary philosophy. In traditional philosophy, however, ontology was mainly an issue of theological and metaphysical speculation, while for the purpose of information system modeling, ontology is a purely pragmatic issue.

AOR modeling adds to the above seven ontological principles of ER modeling the following ones:

8. Different entities may belong to different epistemic categories. There are **agents**, **events**, **actions**, **commitments**, **claims**, and **objects**.

9. We distinguish between *communicative* and *non-communicative* actions and events.

10. Actions create events (called **action events**), but not all events are created by actions.

11. Some of these modeling concepts are **indexical**, that is, they depend on the perspective chosen: in the perspective of a particular agent a_1, *actions* of other agents are viewed as *events*, and *commitments* of an agent a_2 towards a_1 are viewed as *claims* of a_1 against a_2.

12. In the internal perspective of an agent, a **commitment** refers to a specific action to be performed in due time, while a **claim** refers to a specific action event that ought to happen in due time.

13. **Communication** is viewed as asynchronous point-to-point message passing. We take the expressions *receiving a message* and *sending a message* as synonyms of *perceiving a communication event* and *performing a communication act*.

14. There are six designated relationships in which specifically agents, but not objects, participate: only an agent perceives environment events, receives and sends messages, does non-communicative actions, hasCommitment to perform some action in due time, and hasClaim that some action event will happen in due time.

15. We distinguish between *artificial*, *biological* and *institutional* agents. Software agents and robots are artificial agents. For our purposes, humans form the only relevant subclass of biological agents. Institutional agents are social constructs,[4] such as organizations and organizational units.

16. An **institutional agent** consists of a certain number of (human, artificial or institutional) *internal agents* that act on behalf of it. This is illustrated in Figure 2. An institutional agent can only perceive and act through its human and artificial internal agents.

17. In the context of an institutional agent, each **internal agent** has certain *rights* and *duties*.

18. There are three kinds of **duties**: an internal agent may have the *duty to fulfill commitments* of a certain type, the *duty to monitor claims* of a certain type, or the *duty to react to events* of a certain type on behalf of the organization.

19. A **right** refers to an action type such that the internal agent is permitted to perform actions of that type on behalf of the organization.

3.2 Object Types

Object types, such as sales orders or product items, are visualized as rectangles essentially in the same way like entity types in ER diagrams, or object classes in UML class diagrams. They may participate in *association*, *generalization* or *aggregation/composition* relationships with other object types, and in *association* or *aggregation/composition* relationships with agent types.

The multiplicity constraints of an association are specified like in the UML (by means of declarations such as 0..1 or 1..$*$ at the respective association end).

3.3 Agent Types

We distinguish between *artificial* agents, *human* agents and *institutional* agents.[5] Examples of human agent types are Person, Employee, Student, Nurse, or Patient. Examples of institutional agents are organizations, such as a bank or a hospital, or organizational units. An institutional agent consists of a number of internal agents that perceive events and perform actions on behalf of it, by playing certain *roles*.

In many cases, the agent types of a problem domain represent a role played by agents of a more generic (domain-independent) type. Such a role can be represented as a subclass of a more generic class whose extension is more stable. Examples of generic agent types are: *Person* in the case of human agent roles, *Corporation* in the case of institutional agent roles, and *LegalEntity* in the case of mixed human/institutional agent roles.[6]

In certain application domains, there may also be artificial agent types, such as software agents (e.g., involved in electronic commerce transactions), embedded systems (such as automated teller machines), or robots. For instance, in an automated contract negotiation or in an automated purchase decision, a legal entity may be represented by an artificial agent. Typically, an artificial agent is owned, and is run, by a legal entity that is responsible for its actions.

In AOR diagrams, an agent type is visualized as a rectangle with rounded corners. Icons indicating a single human, a group, or a robot may be used for visualizing the distinction between human, institutional and artificial agent.

An agent type may be defined as a subclass of another agent type, thus inheriting all of its attributes (and operations). For instance, in Figure 3, `Person` is a subclass of `LegalEntity`.

Agents may be related to other entities by means of ordinary domain relationships (associations). In addition to the designated relationship types *generalization* and *composition* of ER/OO modeling, there are

[4] With this notion we refer to the terminology of [Sea95] where *social facts* are defined as those facts "involving two or more agents who have collective intentionality", and *institutional facts* are defined as those social facts that are based on the collective assignment of *status functions* and on *constitutive rules*.

[5] Notice that we do not distinguish between 'agents' and 'actors'. Both terms denote the same concept. By default, we use the term 'agent'.

[6] These concepts have been proposed in the *Enterprise Ontology* of [UKMZ98].

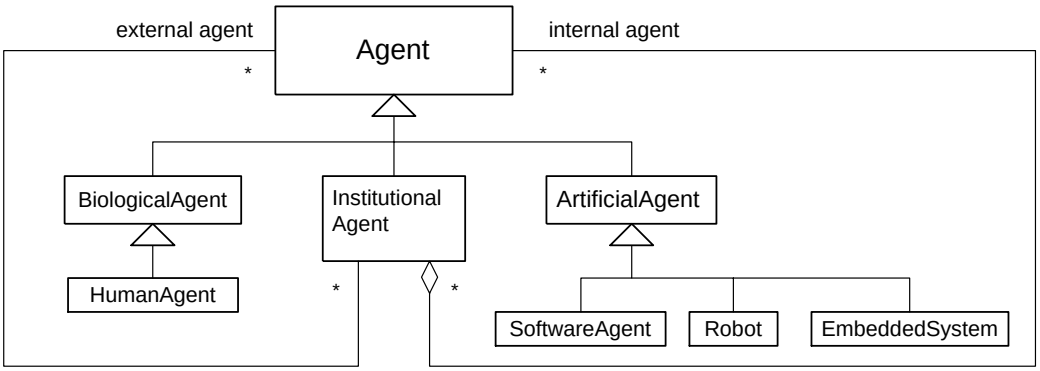

Figure 2: The meta-entity type Agent and its subtypes. An institutional agent aggregates a number of *internal agents*, and is associated with a number of *external agents*.

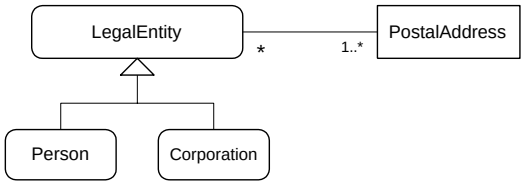

Figure 3: The agent types `Person` and `Corporation` are subclasses of the agent type `LegalEntity` that is many-to-many associated with the object type `PostalAddress`.

further designated relationship types relating agents with events, actions and commitments. They are discussed below.

3.4 External and Internal Agents

With respect to an institutional agent, one has to distinguish between external and internal agents. Internal agents, by virtue of their contractual status (or ownership status, in the case of artificial internal agents), have certain rights and duties, and assume a certain position within the subordination hierarchy of the institution they belong to.[7] In the case of a hospital, examples of human internal agents are doctors and nurses; examples of artificial internal agents are communication-enabled information systems and agentified embedded systems, such as patient monitoring systems.

3.5 Concrete and Prototypical Entities

As in the UML, instances of a type are graphically rendered by a respective rectangle with the underlined name of the particular instance as its title, possibly followed by a colon and its type (see Figure 4). For depicting a *prototypical* instance, the instance name is left empty, such as for *:LegalEntity* in Figure 4.

Figure 4: Two agents: the agent *GerdWagner*, as an instance of the agent type *Person*, and a prototypical instance of *LegalEntity*.

The same notation for instances applies also to objects, actions/events, and commitments/claims.

3.6 Commitments and Claims

Commitments and claims are fundamental components of social interaction processes. Consequently, a proper representation and handling of commitments and claims is vital for automating business processes. This is acknowledged by the ebXML standardization initiative in the statement "The business semantics of each commercial transaction are defined in terms of the Business Objects affected, and the *commitment(s)* formed or agreed."[8]

Representing and processing commitments and claims in information systems explicitly helps to achieve coherent behavior in (semi-)automated interaction processes. In [Sin99], the social dimension of coherent behavior is emphasized, and commitments are treated as ternary relationships between two agents and a 'context group' they both belong to. For simplicity, we treat commitments as binary relationships between two agents.

Commitments to perform certain actions, or to see to it that certain conditions hold, typically arise from certain communication acts. For instance, sending a

[7] We do not define the concept of *position* in this paper.

[8] From the *ebXML Technical Architecture Specification* v0.9.

sales quotation to a customer commits the vendor to reserve adequate stocks of the quoted item for some time. Likewise, acknowledging a sales order implies the creation of a commitment to deliver the ordered items on or before the specified delivery date.

There are two kinds of commitments: commitments to do an action and commitments to see to it that some condition holds. We call the former *to-do* commitments, and the latter *see-to-it-that* commitments. Formally, a to-do commitment of agent a_1 towards agent a_2 may be expressed as a quadruple,

$$\langle\, a_1,\ a_2,\ \alpha(c_1, \ldots, c_n),\ \mathit{TimeSpec} \,\rangle$$

where α denotes an action type, $c_1, \ldots, c_n$ is a suitable list of parameters, and *TimeSpec* specifies, e.g. in the form of a deadline, the time constraints for the fulfillment of the commitment. An example of a to-do commitment where Gerd Wagner is committed to return the book with inventory number 980114 to the department library by November 9, 2000, is expressed as

⟨ GerdWagner, DepLib, returnBook(980114),
9-Nov-2000 ⟩

A see-to-it-that commitment is expressed in the same form, but now $\alpha(c_1, \ldots, c_n)$ represents a proposition (logical sentence) instead of an action term. In the sequel, because they are more fundamental, we only consider to-do commitments.

Obviously, a commitment of a_1 (the debtor) towards a_2 (the creditor) to do the action α is mirrored as a claim of a_2 against a_1 to create the action event α. Commitment and claim processing (that is, the *operational semantics* of commitments and claims) includes the following operations:

- the *creation* of a commitment/claim through the performance of certain actions or the occurrence of certain events,
- the *cancellation* of a commitment by the debtor,
- *waiving* a claim by the creditor (or *releasing* the debtor from the corresponding commitment),
- the *delegation* of a commitment by the debtor to another agent who becomes the new debtor,
- *assigning* a claim by the creditor to another agent who becomes the new creditor
- *fulfilling* a commitment.

A commitment has to be fulfilled unless the debtor is released from it, or certain exceptional circumstances warrant its cancellation. If a commitment cannot be fulfilled or is otherwise violated, some form of compensation may have to be negotiated. We propose to express these commitment processing steps by means of reaction rules in a declarative way.

4 External AOR Models

In an external AOR model, we adopt the view of an external observer who is observing the (prototypical) agents and their interactions in the problem domain under consideration. Typically, an external AOR model has a *focus*, that is an agent, or a group of agents, for which we would like to develop a state and behavior model. In this external-observer-view, 'the world' (i.e. the application domain) consists of various types of

1. ***agents***,
2. communicative and non-communicative ***action events***,
3. ***non-action events***,
4. ***commitments/claims*** between two agent types,
5. ordinary ***objects***,
6. various ***designated relationships***, such as *sends* and *does*,
7. ordinary ***associations***.

In the view of an external observer, actions are also events, and commitments are also claims, exactly like two sides of the same coin. Therefore, an external AOR model contains, besides the agent and object types of interest, the action event types and commitment/claim types that are needed to describe the interaction between the focus agent(s) and the other types of agents. These meta-entity types of external AOR modeling are shown in Figure 5.

Object types, in an external AOR model, belong to one or more agents (or agent types). They define containers for beliefs. If an object type belongs exclusively to one agent or agent type (in the sense of a UML *component* class), the corresponding rectangle is drawn inside this agent (type) rectangle. If an object type represents beliefs that are shared among two or more agents (or agent types), the object type rectangle is drawn outside of the respective agent (type) rectangles. An object type may be shared by a number of agent types in two different ways: all agents may use the same representation (schema) for the shared object type, or each agent may use its own internal representation of it, in which case a dependency arrow between the internal representations and

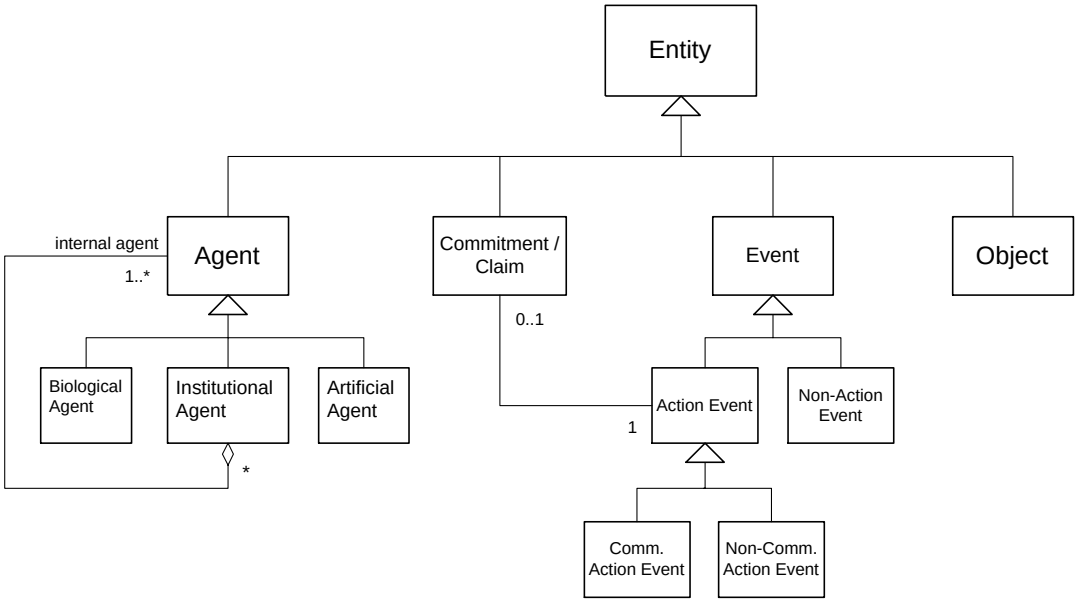

Figure 5: The meta-entity types of external AOR modeling. Notice that a commitment/claim type is conceptually coupled to an action event type.

the shared object type is used, as for the object type `Book` in the agent diagram shown in Figure 7. In any case, sharing an object type among a number of agents does not imply that all these agents have the same beliefs about it, or, in other words, that there is a common extension of it shared by all agents.

An external AOR model does not include any software artifacts. It rather represents a conceptual analysis view of the problem domain and may also contain elements which are merely descriptive and not executable by a computer program (as required for enterprise modeling).

An external AOR model may comprise one or more of the following diagrams:

Agent Diagrams depicting the agent types of the domain, certain relevant object types, and the relationships among them.

Interaction Frame Diagrams depicting the action event types and commitment/claim types that determine the possible interactions between two agent types (or instances).

Interaction Sequence Diagrams depicting prototypical instances of interaction processes.

Interaction Pattern Diagrams focusing on general interaction patterns expressed by means of a set of reaction rules defining an interaction process type.

The agent diagrams, interaction frame diagrams and interaction pattern diagrams of a model may be merged into a single all-encompassing *External AOR Diagram (EAORD)*. Interaction sequence diagrams are normally not included in such an EAORD, since they depict instances only, and are not at the type level.

4.1 Agent Diagrams

An *agent diagram* depicts the agents and agent types of an application domain, together with their internal agents and agent types, their beliefs about objects and the relationships among them. Object types occur in two forms: external representations of object types (graphically rendered by rectangles drawn outside any agent type rectangle) representing shared beliefs about objects, and internal representations of object types (graphically rendered by rectangles drawn inside an agent type rectangle) representing 'private' beliefs about objects. In certain cases, internal representations are views of external representations. We indicate such a dependency with a dashed arrow from the internal representations to the external representations. An example of an agent diagram is shown in Figure 7.

4.2 Actions Are Events but Not All Events are Actions

In the external observer perspective, all actions of agents are at the same time also events that may be perceived by other agents. The other way around, there are many events that are created by the corre-

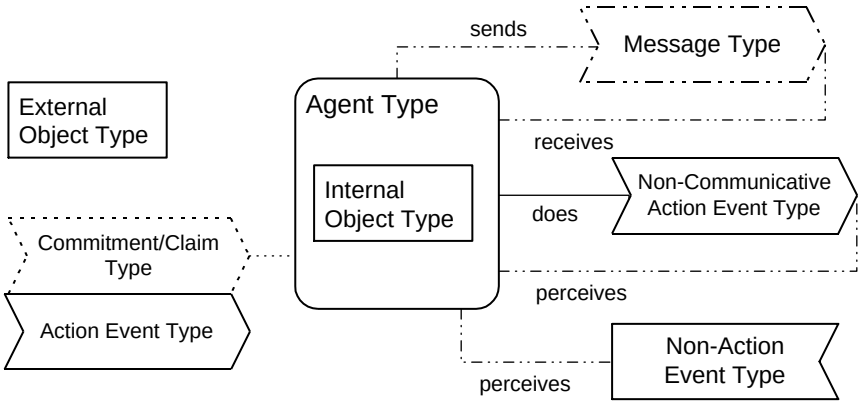

Figure 6: The core elements of external AOR modeling.

sponding actions of agents. However, there are also events which are not created by actions (e.g., temporal events, or events created by natural forces). Consequently, we make a distinction between *action events* and *non-action events*.

In an External AOR Diagram, an action event type is graphically rendered by a special arrow rectangle where one side is an incoming arrow linked to the agent (or agent type) that performs this type of action, and the other side is an outgoing arrow linked to the agent (or agent type) that perceives this type of event. Communicative action event rectangles have a dot-dashed line. In the case of a non-action event, the corresponding event rectangle does not have an outgoing arrow (see Figure 8).

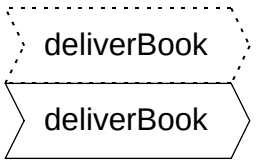

Figure 9: The commitments/claim type to deliver a book is coupled with the corresponding action event type of delivering books. Normally, the occurrence of a `deliverBook` action event is preceded by the formation of the corresponding commitment/claim (e.g., as a consequence of a communicative action of the type *confirm book request*).

4.3 Commitments/Claims

In external AOR modeling, a *commitment* of agent a_1 towards agent a_2 to perform an action of a certain type (such as a commitment to deliver an item) can also be viewed as a *claim* of a_2 against a_1 that an action event of that type will happen. Commitments/claims are conceptually coupled with the type of action event they refer to (such as *deliverItem* action events). This is graphically rendered by an arrow rectangle with a dotted line on top of the action event rectangle it refers to, as depicted in Figure 9.

4.4 Interaction Frame Diagrams

In an external AOR model, there are four types of designated relationships between agents and action events: sends and receives are relationship types that relate an agent with communicative action events, while does and perceives are relationship types that relate an agent with non-communicative action events. In addition, there are two types of designated relationships between agents and commitments/claims: hasCommitment and hasClaim. These designated relationship types are visualized with particular connector types as depicted in Figure 10. Notice that all of them come with the multiplicity constraint *one-to-many* which, for simplicity, is not explicitly shown in the diagram. The name of these designated relationship types will usually be omitted in AOR diagrams.

An *interaction frame diagram*, in an external AOR model, describes the possible interactions between two (types of) agents. It consists of various types of

1. communicative action events,
2. non-communicative action events,
3. commitments/claims (coupled with the corresponding types of action events), and
4. non-action events.

An example of an interaction frame diagram is shown in Figure 11.

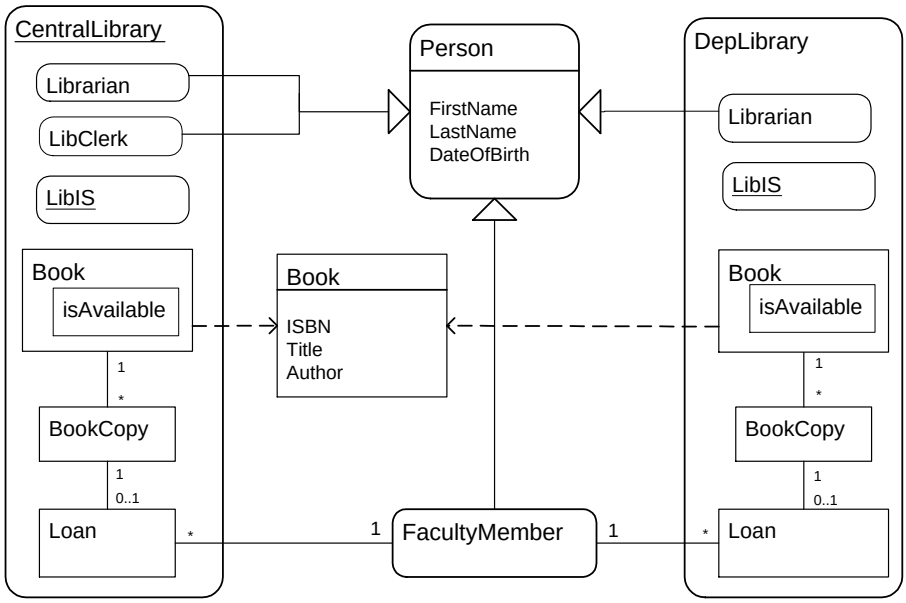

Figure 7: An AOR agent diagram for the university libraries domain. The central library and the department libraries are institutional agents, having librarians as human internal agents and a library information system (LibIS) as an artificial internal agent. `FacultyMember` is another agent type in this domain. Important object types are `Book`, `BookCopy` and `Loan`, that is, libraries have beliefs about their books, their book copies, and their loans. Notice that the dependency arrows from the internal `Book` rectangles of `CentralLibrary` and `DepLibrary` to the external `Book` rectangle indicates that `Book` is a shared object type with specific internal representations. The subclasses `CentralLibrary::Book.isAvailable` and `DepLibrary::Book.isAvailable` are formed by all books that satisfy the status predicate `isAvailable`, that is, for which there is at least one book copy available, in the respective context.

Figure 8: A communicative action event, a non-communicative action event, and a non-action event.

4.5 Interaction Sequence Diagrams

An interaction sequence diagram depicts (some part of) an instance of an interaction process. An *interaction process* is a sequence of action events and non-action events, performed and perceived by agents, and following a set of rules (or protocol) that specifies the type of the interaction process. Agents may interact with their inanimate environment, or they may interact with each other. A simple example of the former type of interaction process is my reaction to turn on the light in my office when it becomes dark outside, that is depicted in Figure 12.

A *social interaction process* is a temporally ordered, coherent set of action events and non-action events, involving at least one communicative action event, performed and perceived by agents, and following a set of rules, or protocol, that is governed by norms, and that specifies the type of the interaction process.[9] An example of a social interaction process is shown in Figure 13. Social norms imply, for instance, that after having confirmed a book request, the library is committed to deliver the requested book.

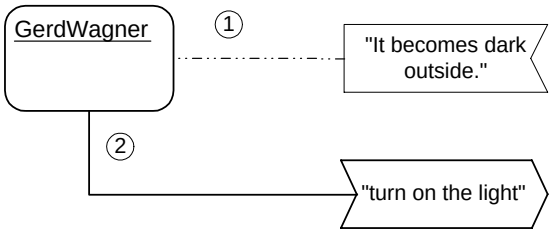

Figure 12: A non-social interaction process involving an agent and his inanimate environment.

We consider a business process as a special kind of a

[9] Notice that we did not choose *activities* as the basic elements of a process. While an *action* happens at a time instant (i.e., it is immediate), an *activity* is being performed during a time interval (i.e., it has duration), and may consist of a sequence of actions.

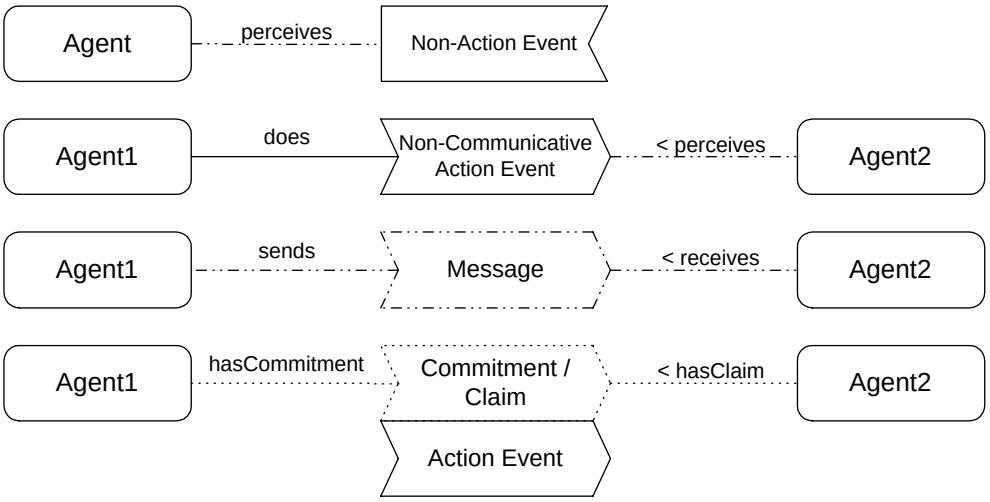

Figure 10: The designated relationship types sends, receives, does, perceives, hasCommitment and hasClaim.

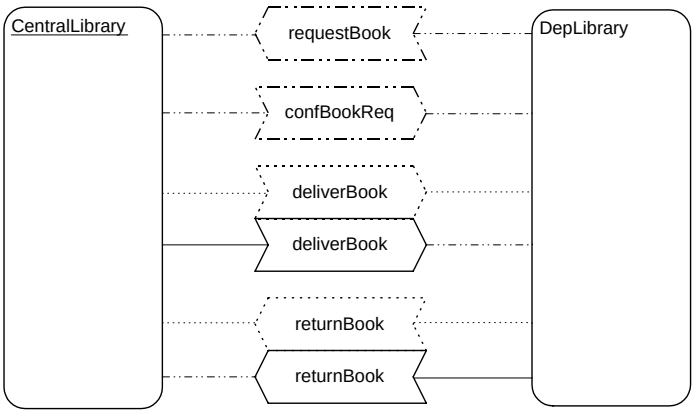

Figure 11: The interaction frame between the central library and the department libraries: a department library may request a book from the central library; when such a book request has been confirmed by the central library, then there is a commitment to deliver the requested book (visualized by the dashed-line *deliverBook* arrow rectangle); normally, such a commitment leads to a corresponding action (visualized by the solid-line *deliverBook* arrow rectangle); after a book has been delivered, there is a commitment to return it in due time.

social interaction process. Unlike physical or chemical processes, social interaction processes are based on communication acts that may create commitments and are governed by norms. We distinguish between an interaction process type and a concrete interaction process (instance), while in the literature the term 'business process' is ambiguously used both at the type and the instance level. Reaction rules are the most important type of *business rules*, as we argue in [TW01a].

4.6 Reaction Rules and Interaction Pattern Diagrams

We model interaction process types by identifying interaction patterns and expressing them by means of ***reaction rules***. Reaction rules may be used both for *describing* the reactive behavior of all kinds of agents, and for the *executable specification* of the reaction patterns of an artificial agent to be built.

An example of a reaction rule is the following: *When the central library receives a certain book request from a department library, it checks if a copy of that book is available, and if this is the case, the request is confirmed, and a new corresponding loan object as well as a commitment to deliver the requested book in due time is created.* This rule is visualized as rule R2 in Figure 14.

A reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the agent rectangle whose reaction pattern it represents. Each reaction rule has exactly one incoming arrow with

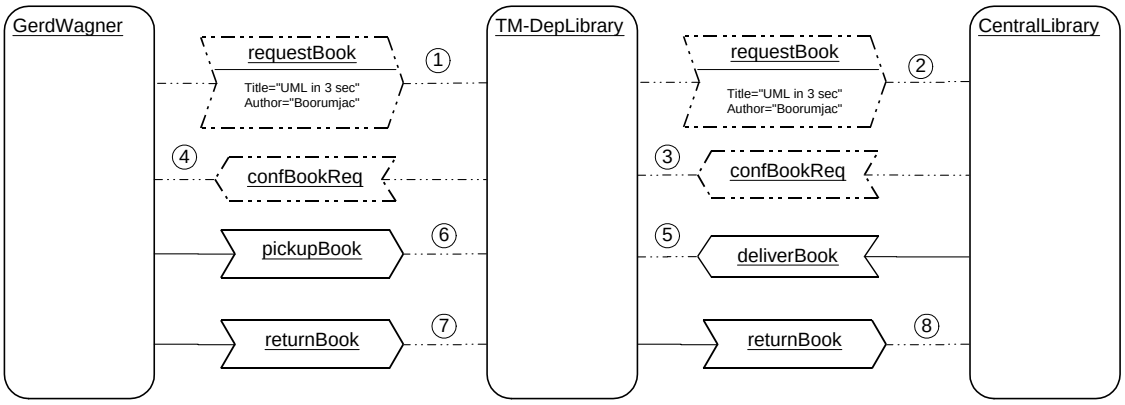

Figure 13: A social interaction process involving three agents: GerdWagner's request to lend the book "UML in 3 sec" (1) is forwarded by the TM-DepLibrary (the library of the technology management department) to the CentralLibrary (2).

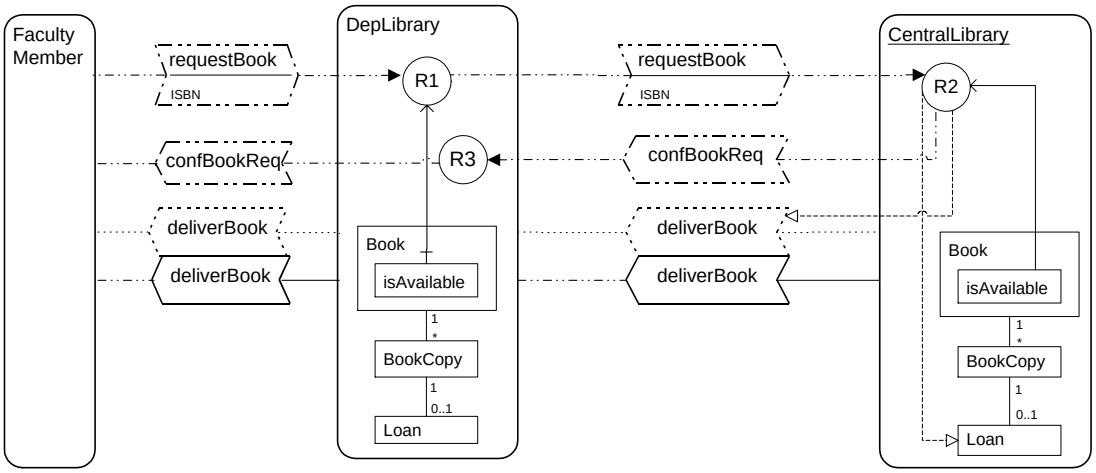

Figure 14: An interaction pattern diagram describing the process type where a faculty member requests a book from a department library such that the request is forwarded to the central library because the requested book is not available at the department library.

a solid arrowhead: it represents the triggering event condition which is also responsible for instantiating the reaction rule (binding its variables to certain values). In addition, there may be ordinary incoming arrows representing state conditions (referring to corresponding instances of other entity types). There are two kinds of outgoing arrows. An outgoing arrow of the form $\dashrightarrow$ denotes a mental effect referring to a change of beliefs and/or commitments. An outgoing connector to an action event type denotes the performance of an action of that type.

Reaction rules may also be represented in textual template form. For, instance, R2 could be expressed as in Table 1. In symbolic form, a reaction rule is defined as a quadruple

$$\varepsilon, C \longrightarrow \alpha, F$$

where ε denotes the triggering event term, C denotes the state condition formula, α denotes the resulting action term, and F denotes the mental effect formula. Both C and F are formulas from a logical language corresponding to the (mental state) schema of the agent whose reaction pattern is specified by the rule.[10]

Notice that in an EAORD, the actions performed by one agent may be at the same time the events perceived by another agent. An EAORD can therefore visualize the reaction chains that arise by one reaction triggering another one.

[10]The reader is referred to [Wag98] for further explanations of the formal semantics of reaction rules

ON	Event	RECEIVE requestBook(?ISBN) FROM ?DepLib
IF	Condition	BookCopy.isAvailable(?ISBN, ?InvNo)
THEN	Action	SEND confBookReq(?ISBN) TO ?DepLib
	Effect	CREATE COMMITMENT TOWARDS ?DepLib TO deliverBook(?ISBN) BY tomorrow(); CREATE BELIEF Loan(?DepLib, ?ISBN, ?InvNo, today())

Table 1: The reaction rule R2 of Figure 14 in textual template form.

5 Internal AOR Models

In an *internal* AOR model, we adopt the internal view of a particular agent to be modeled. In this first-person-view, 'the world' (i.e. the domain of interest) consists of various types of

1. other ***agents***;
2. ***actions***;
3. ***commitments*** towards other agents to perform certain actions;
4. ***events***, many of them created by actions of other agents;
5. ***claims*** against other agents that certain action events happen,
6. ordinary ***objects***;
7. various ***designated relationships***, such as *isSentTo* and *isPerceivedBy*;
8. ordinary ***associations***.

These meta-entity types of internal AOR modeling are shown in Figure 15.

An internal AOR model depicts 'the world' as it may be represented in the mental state of the focus agent. If the focus agent is an organization, the internal AOR model represents its view of 'the world', and may be used to design its information system. Thus, AOR modeling suggests the following development path for organizational information systems:

1. In the domain analysis, develop an external AOR model of an organization (or a group of organizations) and its (or their) environment from the perspective of an external observer of the scenario.
2. Transform the external AOR model into an internal AOR model for the focus agent for that an information system is to be developed (typically an organization or an organizational unit). If there are several focus agents, and for each of them an information system is to be developed, this step can be iterated.
3. Transform the internal AOR models obtained in the previous step into database design models (logical database schemas), e.g. for object-relational (SQL-99) database management systems, or into sets of corresponding logical data structure definitions in a target language such as Java.
4. Refine the design models into implementation models (physical database schemas) by taking performance and storage management issues, as well as the specific features of the target language (such as SQL-99 or Java), into consideration.
5. Generate the target language code.

An internal AOR model may comprise one or more of the following diagrams:

Reaction Frame Diagrams depicting other agents (or agent types) and the action and event types, as well as the commitment and claim types that determine the possible interactions with them.

Reaction Sequence Diagrams depicting prototypical instances of interaction processes in the internal perspective.

Reaction Pattern Diagrams focusing on the reaction patterns of the agent under consideration expressed by means of reaction rules.

The reaction frame diagrams and reaction pattern diagrams of a model may be merged into a single all-encompassing *Internal AOR Diagram (IAORD)*. Reaction sequence diagrams are normally not included in such an IAORD, since they depict instances only, and are not at the type level. All internal AOR diagrams are drawn within a surrounding frame, preferably with rounded corners and with the name of the agent that is being modeled in the top left corner.

5.1 External and Internal Agents

In an internal AOR Diagram for an institutional agent, the internal agents (and/or agent types) of the institutional agent to be modeled may appear at the top level, like the internal agent type *SalesDepartment* in Figure 16, where they are distinguished from

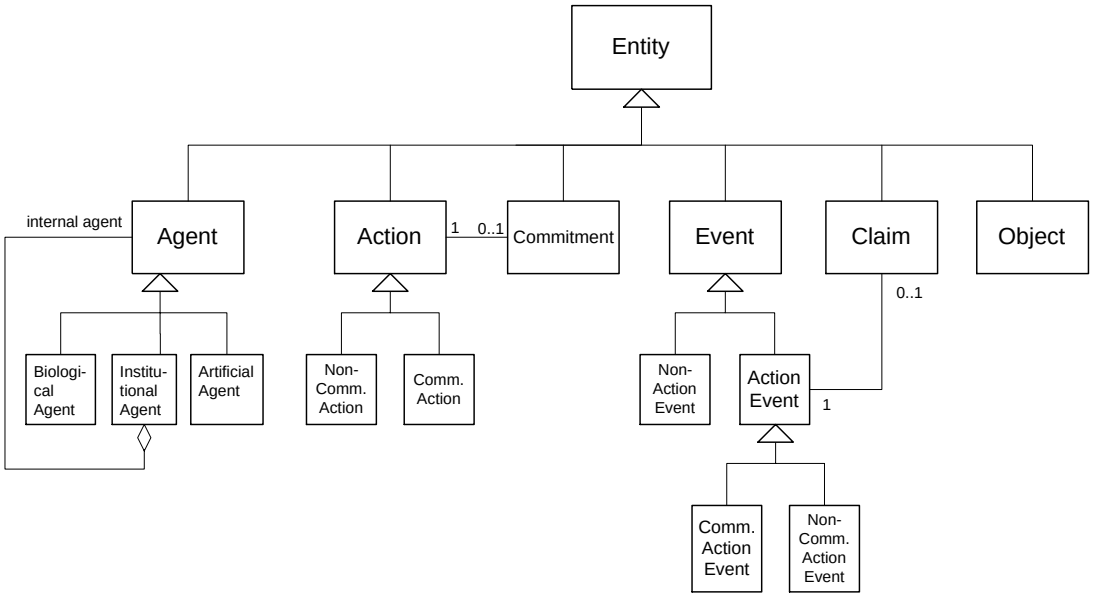

Figure 15: The most important meta-entity types of internal AOR modeling.

external agents by dashing their rectangle line. All internal agents of internal agents, like *SalesPerson* within *SalesDepartment*, are again internal agents.

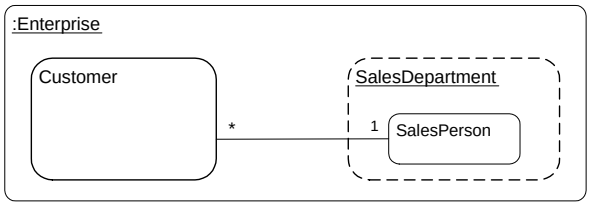

Figure 16: From the point of view of an enterprise, `Customer` is an external agent type, while `SalesDepartment` is an internal agent and `SalesPerson` is an internal agent type of `SalesDepartment`. Each salesperson has a certain number of associated customers.

5.2 Actions and Events

In the perspective of an institutional agent, only the actions of internal agents performed on behalf of the institution count as actions, while the actions of external agents count as events.

An event type is graphically rendered by a rectangle with an incoming arrow side, while an action type is graphically rendered by a rectangle with an outgoing arrow side. Communication event types and communication act types are visualized by a dashed-dotted line.

In an internal model of an agent, events and actions are related to other agents by means of four designated relationship types: an outgoing message isSentTo – and an incoming message isReceivedFrom – another agent, while a physical action isPerceivedBy – and an action event isCreatedBy – another agent. These special relationship types are designated with particular connector types as illustrated in Figure 17.

Notice that, typically, an organizational information system does not perceive environment events since it does usually not have any sensor devices. In many cases, environment events are reported to an information system through communication acts of internal agents. But in principle, information systems may receive perception signals representing environment events from sensors that are attached to it. Temporal events, such as represented by the signals "it is now 11:00 on 11-Nov-2000" or "the time for collecting bids is now over", are an important kind of non-action event. They are created by a timer that may be regarded as a particular sensor device.

An action type may be defined like a table in a SQL-like language in the following way:

```
CREATE ACTION TABLE  deliverItem(
ItemCode          INTEGER,
Quantity          DECIMAL(5,2),
SalesOrderNo      CHAR(10),
DeliveryAddress   CHAR(30)
)
```

Event types may be defined in the same way:

```
CREATE EVENT TABLE  payInvoice(
InvoiceNo   INTEGER,
Amount      DECIMAL(5,2)
)
```

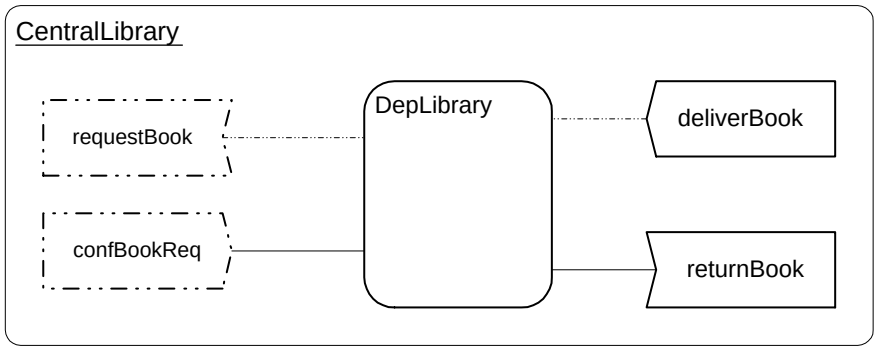

Figure 17: In an internal AOR model of the central library, a `requestBook` message isReceivedFrom a department library, an `confBookReq` message isSentTo the department library, a `deliverBook` action isPerceivedBy the department library, and a `returnBook` action event isCreatedBy a department library.

Unlike ordinary SQL tables, both types of tables would have additional implicit columns for recording

1. the internal agent that has performed the action, or the external agent that has created the event;
2. in the case of a communicative action/event: the addressee(s) of the message;
3. the time instant when the action has been performed, or when the event has happened.

By recording all perceived events and all performed actions, an artificial agent (such as an information system) implements a basic form of *memory*. It follows from the principle that the past cannot be changed, that the rows of action and event tables must not be deleted or modified.

5.3 Commitments and Claims

A commitment towards another agent (such as a commitment towards a customer to deliver an item) is coupled with the associated action (such as a *deliverItem* action) to be performed. It is visualized as a rectangle with a dotted line on top of the associated action rectangle as in Figure 18.

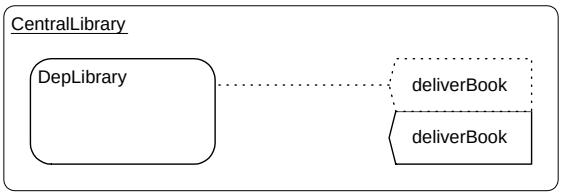

Figure 18: A commitment towards another agent (to deliver a book) is visualized together with the associated action (of carrying out the delivery).

A claim against another agent (such as a claim against a customer to pay an invoice) is coupled with the associated event (such as a *payInvoice* event). It is visualized as a rectangle with a dotted line on top of the associated event rectangle as in Figure 19.

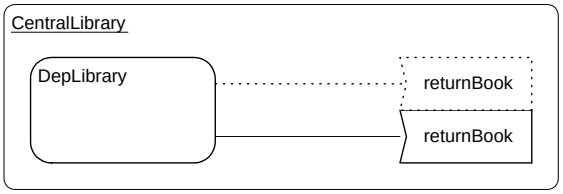

Figure 19: A claim against another agent (here: a claim against a department library to return a book) is visualized together with the associated event (of returning the book).

Technically, a commitment may be represented in a special table whose definition requires that there is a previously defined action type with the same name. For instance,

```
CREATE COMMITMENT TABLE deliverItem
```

defines a table for recording commitments to deliver items to customers. It requires that the action table *deliverItem* has already been defined. Commitment tables have two implicit additional columns: 1) for storing the ID of the agent towards whom the commitment holds, 2) for representing the deadline for the fulfillment of the commitment.

Likewise, for representing claims against other agents, a claim table may be defined by referring to a previously defined event type:

```
CREATE CLAIM TABLE payInvoice
```

Claim tables have two implicit additional columns: 1) for storing the ID of the agent against whom the claim holds, 2) for representing the deadline for the fulfillment of the claim.

5.4 Reaction Frames

In an internal AOR model, the reactive behavior of the agent (type) under consideration with respect to another agent (type) A is described by means of six kinds of entity types:

1. **communication events**, or incoming messages, created by the communication acts of instances of A;
2. **communication acts**, or outgoing messages, directed to instances of A;
3. **claims** against instances of A;
4. **non-communicative events** created by the actions of instances of A;
5. **commitments** towards instances of A;
6. **non-communicative actions** which may be performed in order to fulfill corresponding commitments towards instances of A, or in response to events within the interaction frame.

Figure 20 shows a Reaction Frame Diagram for the central library describing the reactive behavior with respect to the agent type `DepLibrary`. There is the incoming message type `requestBook`, the outgoing message type `confBookReq`, the commitment type `deliverBook` coupled with the corresponding action type, and the claim type `returnBook` coupled with the corresponding event type.

Notice that when actions and events are recorded in an artificial agent (e.g., in special tables), an interaction log is created that can be viewed as a kind of memory.

5.5 Internalization

An external AOR diagram can be transformed into an internal AOR diagram for one of the focus agents (or agent types) by

1. omitting the focus agent whose perspective is modeled (the 'first-person agent');
2. turning all action event rectangles directed towards the first-person agent into event rectangles;
3. turning all action event rectangles directed towards an interaction partner of the first-person agent into action rectangles;
4. turning all commitment/claim rectangles directed towards the first-person agent into claim rectangles;
5. turning all commitment/claim rectangles directed towards an interaction partner of the first-person agent into commitment rectangles;
6. dashing the rectangle lines of all top-level internal agents; and
7. merging the internal and external representations of object types.

This transformation is called *internalization*.

For instance, the interaction frame diagram shown in Figure 11, modeling the interaction frame between the central library and the department libraries, can be transformed into the reaction frame diagram for the central library information system shown in Figure 20.

In an Internal AOR Diagram (IAORD), since its scope is a single agent, we can no longer see complete interaction patterns involving two or more agents. The behavior of the agent under consideration is modeled by identifying its reaction patterns and expressing them in the form of ***reaction rules***, as in the IAORD for the central library shown in Figure 21 and in the IAORD for the department libraries shown in Figure 22. Both of these IAORDs are derived from the EAORD of Figure 14.

5.6 Modeling the Rights and Duties of Internal Agents

An internal agent, by virtue of its *position* and the *roles* assigned to it, has certain *rights* and *duties*.

Both duties and rights correspond to designated relationships between internal agent types (or specific internal agents) and commitment, claim, event, and action types. They are visualized with the help of special connectors having a bullet at the side of the internal agent type rectangle. The bullet is empty in the case of a right, while it is solid in the case of a duty.

5.6.1 Having a Duty to React to Certain Events

The duty to react to events of a certain type – the *hasDutyToReact* relationship – is visualized by means of a dotted connector line from the responsible internal agent type rectangle to the event type rectangle, as shown in Figure 23 between the `submitPurchaseOrder` event type and the `SalesPerson` agent type. If i denotes an agent, and ε denotes an event type, we can express the deontic statement that i has the duty to react to events of type ε by means of

$$\mathsf{DR}(i, \varepsilon)$$

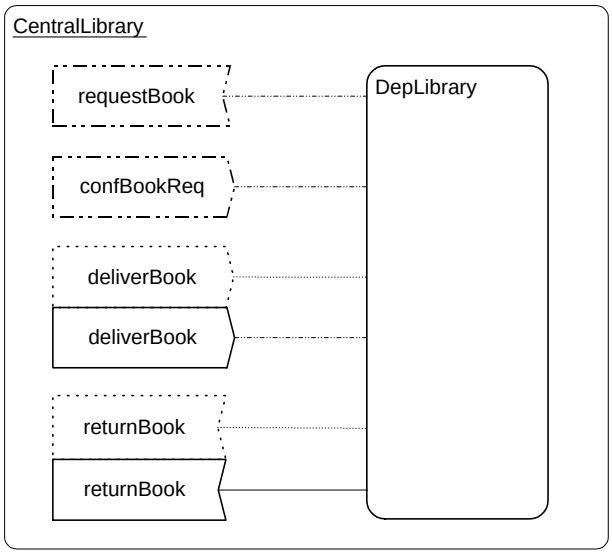

Figure 20: The interaction frame between department libraries and the central library, as shown in Figure 11, projected onto a *Reaction Frame Diagram* for the central library. Its content can be described as follows: the central library receives `requestBook` messages from department libraries; it sends `confBookReq` messages to department libraries; it has `deliverBook` commitments towards department libraries, and performs `deliverBook` actions that are perceived by department libraries; it has `returnBook` claims against department libraries, and it perceives `returnBook` events created by department libraries.

5.6.2 Having a Duty to Fulfill Certain Commitments

The duty to fulfill commitments of a certain type – the *hasDutyToFulfill* relationship – is visualized by means of a dotted connector line between the agent type rectangle and the commitment type rectangle. This is illustrated in Figure 23 where a hasDutyToFulfill connector is drawn between the `deliverItem` commitment type and the `DelivAgt` agent type in order to express the duty of a delivery agent to fulfill commitments to deliver items to customers.

If α denotes an action type, we can express the deontic statement that i has the duty to fulfill commitments to perform actions of type α in due time by means of

$$\mathsf{DfC}(i, \alpha)$$

5.6.3 Having a Duty to Monitor Certain Claims

The duty (say, of a Clerk) to monitor claims of a certain type (e.g., to pay an invoice) – the *hasDutyToMonitor* relationship – is visualized by means of a dotted connector line between the (`Clerk`) agent type rectangle and the (`payInvoice`) claim type rectangle.

If ε denotes an event type, we can express the deontic statement that i has the duty to monitor claims that action events of type ε happen in due time by means of

$$\mathsf{DmC}(i, \varepsilon)$$

5.6.4 Having a Right to Perform Certain Actions

The right to perform actions of a certain type – the *hasRightToPerform* relationship – is visualized by means of a dotted connector line between the internal agent type rectangle and the action type rectangle, as shown in Figure 23 between the `acknSalesOrder` communication act type and the `SalesPerson` agent type.

If α denotes an action type, we can express the deontic statement that i has the right to perform actions of type α by means of

$$\mathsf{Right}(i, \alpha)$$

There are also *derived rights*: whenever an agent has the duty to fulfill certain commitments, it also has the implicit right to do the respective actions. This can be expressed in the form of an implication:

$$\mathsf{DfC}(i, \alpha) \supset \mathsf{Right}(i, \alpha)$$

In such a case, we do usually not draw the derived *hasRightToPerform* relationship in the diagram.

5.6.5 Having No Right to Perform Certain Actions

It may be *prohibited* for certain agents to perform actions of a certain type. For instance, we may want to specify that delivery agents do not have the right to acknowledge sales orders using a *hasNoRightToPerform* connector, as in Figure 23.

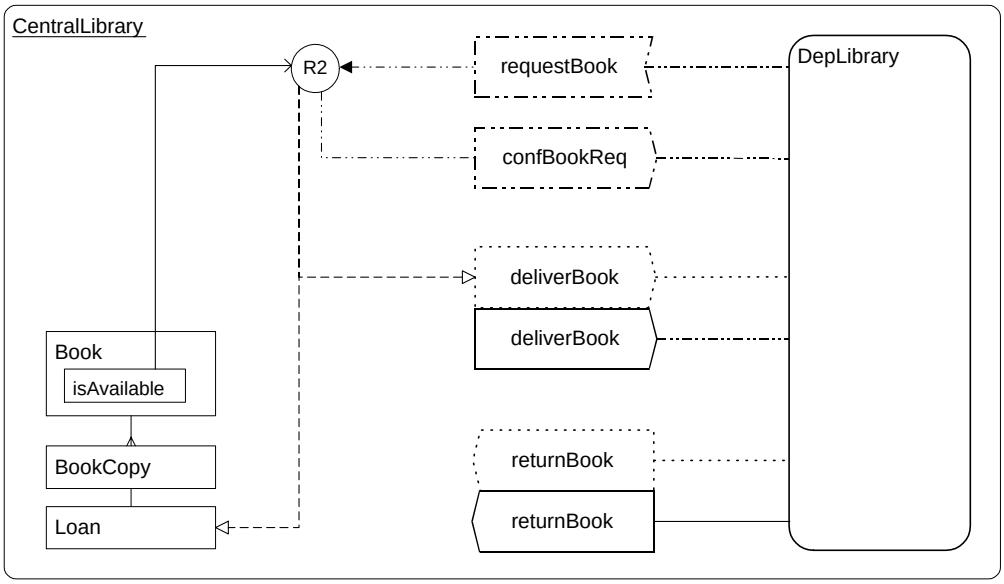

Figure 21: A *Reaction Pattern Diagram* for the central library describing the process step where the central library has to react to a book request from a department library.

Formally, we can express the deontic statement that it is prohibited for i to perform actions of type α by

$$\mathsf{Proh}(i, \alpha)$$

5.6.6 The Deontic Logic of AOR Modeling

The deontic logic arising from the AOR metamodel is still under investigation. However, we can already express some fundamental deontic principles.

It is quite common in an organization, that it is not completely determined for every action whether it is permitted or prohibited. In the AOR metamodel, this principle of *normative underdetermination* takes the following form: in a specific AOR model, it needs not be the case that for every internal agent i, and every action type α, either i has the right to do actions of type α, or it is prohibited for i to perform actions of type α. Symbolically,

$$\neg\forall i \forall \alpha \, (\mathsf{Right}(i, \alpha) \vee \mathsf{Proh}(i, \alpha))$$

or, equivalently,

$$\exists i \exists \alpha \, (\neg\mathsf{Right}(i, \alpha) \wedge \neg\mathsf{Proh}(i, \alpha))$$

Due to inconsistent specifications of regulations, it is also quite common in practice that certain actions are both permitted and prohibited. For a specific AOR model, this principle of *normative inconsistency* is expressed in the following way:

$$\exists i \exists \alpha \, (\mathsf{Right}(i, \alpha) \wedge \mathsf{Proh}(i, \alpha))$$

stating that for a certain internal agent i, there is an action type α, such that i has the right to do actions of type α and, at the same time, it is prohibited for i to perform actions of type α.

5.7 Transforming an Internal AOR Model into a Database Schema

We briefly sketch the transformation of an internal AOR model into an object-relational (SQL-99) database schema. Such a mapping may be viewed as defining a semantics for the AOR metamodel assigning an operational meaning to the modeling elements action, event, commitment, and claim.

Object types and association types are transformed like in the standard implementation of ER models (that is, object types and many-to-many association types are implemented as separate tables, one-to-many and one-to-one association types are represented as additional reference columns in the participating object tables). A component class is transformed into a corresponding complex-valued attribute of its superior class. Subclasses are represented by means of the SQL-99 subtable construct.

A communication event type (or incoming message type) is transformed into a special table schema with three additional implicit columns for the ID of the sender, the ID of the internal agent who is the addressee, and the time instant at which the message has arrived. Likewise, a communication act type (or outgoing message type) is transformed into a special table schema with three additional implicit columns for the ID of the internal agent who has sent the message, the ID of the external agent who is the addressee, and the time instant at which the message

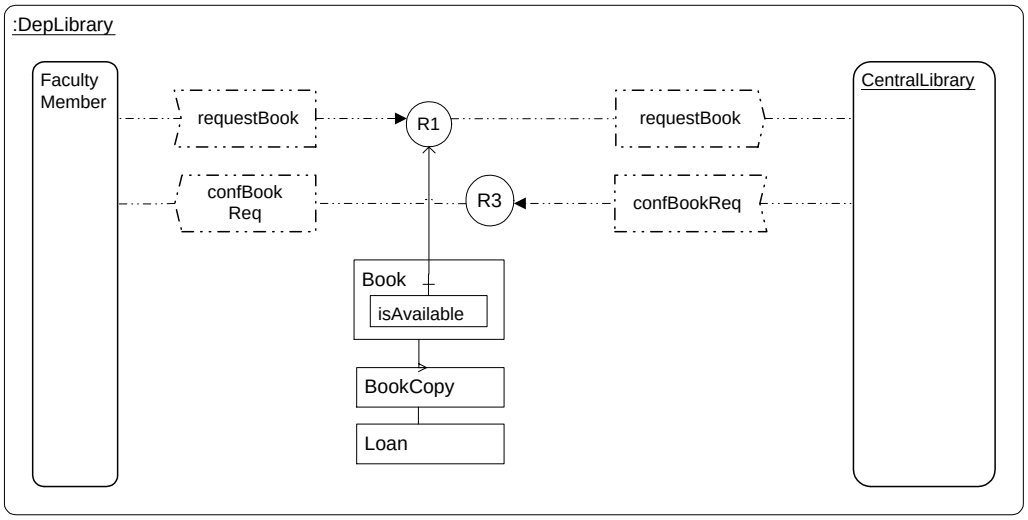

Figure 22: A *Reaction Pattern Diagram* for the department libraries describing the process steps where a department library has to react to a book request from a faculty member and to a confirmation message from the central library.

has been sent.

A commitment (or claim) type is transformed into a table with the same base schema as the action (or event) type it refers to but with different implicit columns: the first additional column represents the ID of the agent towards whom the commitment (or against whom the claim) holds, and the second additional column represents the temporal constraints for fulfilling the commitment (or for obtaining the benefits from the claim).

An external agent type is transformed into a special table schema including declarations of the actions and events that can be performed and perceived by representatives of that class, and of the commitments and claims that the organization may have towards and against them.

An internal agent type is transformed into a special table schema including declarations of the events, commitments and claims that its representatives must react to, fulfill and monitor, and the actions they may perform.

The assignment of event, commitment and claim types as responsibilities to internal agent types enables the organizational information system to play an active role in a number of basic business processes, such as to

1. define tasks to be performed in an automated fashion by artificial internal agents;
2. maintain role-specific to-do-lists for communicating the current duties to human internal agents;
3. remind the responsible internal agents of an approaching deadline to fulfill a commitment towards another agent;
4. remind the responsible internal agents of deadlines in connection with claims against other agents.

6 Further Techniques for Behavior Modeling

In addition to using *reaction rules* for behavior modeling in Interaction Pattern Diagrams in external AOR modeling, and in Reaction Pattern Diagrams in internal AOR modeling, we propose to use *activity diagrams* in requirements analysis and *behavior constraints* for defining the correct behavior of an agent in a declarative fashion in design. Furthermore, it is an option to include in an internal AOR model UML statemachine diagrams for adding more detail and further views.

6.1 Activity Diagrams

UML activity diagrams seem to be well-suited for agent-oriented requirements analysis since their *swimlane* construct can be used to assign activity types to agents, and the ≪signal≫ class stereotype can be used to model communicative action events. It is an issue for future research to adapt and integrate UML activity diagrams with external AOR diagrams.

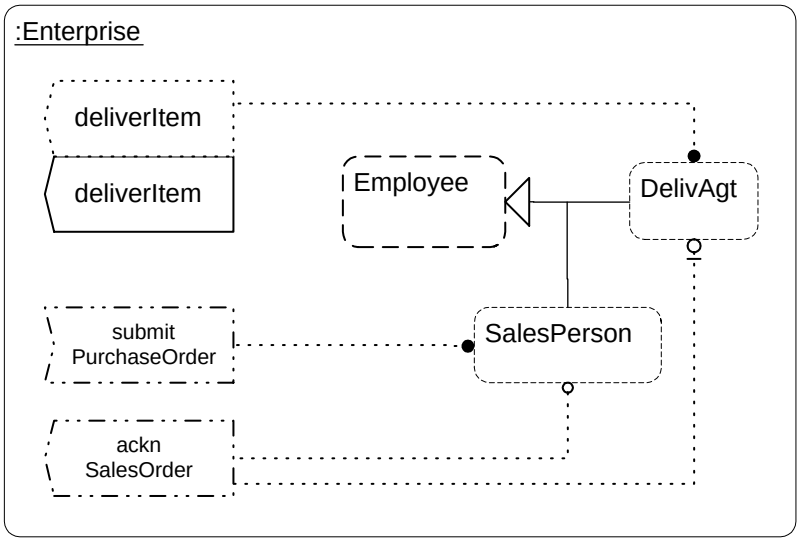

Figure 23: `SalesPerson` hasDutyToReact to `submitPurchaseOrder` messages and hasRightToSend `acknSalesOrder` messages. `DelivAgt` hasDutyToFulfill `deliverItem` commitments (implying the right to do `deliverItem` actions). `DelivAgt` hasNoRightToSend `acknSalesOrder` messages.

6.2 Behavior Constraints

Like *correctness properties* in the theory of formal verification,[11] behavior constraints can be viewed as temporal logic assertions expressing, e.g., a *safety* property ('something undesirable will never happen') or a *progress* property ('something desirable will happen'). Typically, these constraints refer to communication events (or messages) and to beliefs and commitments. We can only sketch this topic here (we plan to elaborate it in our future work).

Examples of progress constraints are the requirements that

- After a book has been requested by a faculty member, either a denial or a confirmation will be sent to him.
- After a book request has been confirmed by the central library, the requested book will be delivered to the department library.

Examples of safety constraints are the requirements that

- A book request must not be confirmed if the library IS does not belief that the requested book is available.
- A book delivery to a faculty member who still has books that are overdue must not be carried out before the concerned books have been returned.

[11] The formal verification of safety and progress properties in the software engineering of concurrent reactive systems is the topic of [MP92].

Behavior constraints can be used to express important properties of a system independently of its implementation. A particular implementation of an automated business process may be proved as correct by formally deriving the behavior constraints from (an executable specification of) the implementation code. Behavior constraints can also be used for the automated supervision of semi-automated business processes.

Basic progress constraints concern actions to be performed, while basic safety constraints concern actions to be suppressed. An agent-oriented information management system should support the declarative specification of behavior constraints in a similar way as SQL supports static integrity constraints by means of CREATE ASSERTION. Similarly like certain SQL assertions can be operationalized by means of SQL triggers, certain progress constraints in Internal AOR modeling can be operationalized by means of reaction rules.

The formalization and visualization of behavior constraints within AOR Diagrams is an issue for future research.

7 Tool Support for AOR Modeling

AORML tools are available from `www.AOR.rezearch.info`. There is a Microsoft Visio template for AOR modeling, providing the specific graphical shapes of the AOR modeling elements. Code generation tools may be provided in

the future.

8 Related Work

We briefly discuss the relationship of the AORML to Entity-Relationship (ER) modeling, to the Unified Modeling Language (UML), to some of its business modeling extensions, and to a number of other works.

8.1 Entity-Relationship Modeling

As discussed in section 3.1, AOR modeling follows ER modeling and extends it by introducing a distinction between a number of fundamentally different categories of entity types, and by adding modeling elements and a notation for behavior modeling. The commonalities and differences between ER, UML and AORML are summarized in Table 2.

8.2 The Unified Modeling Language

The Unified Modeling Language (UML) offers a comprehensive set of (visual) modeling constructs for object-oriented information and process modeling. The UML also has an extensibility mechanism that allows to introduce subcategories of UML meta-concepts, together with their specific graphical renderings, in the form of *'stereotype'* declarations. Some predefined UML 'stereotypes' come quite close to some of the AOR meta-concepts:

Signals are defined as a class 'stereotype'. They correspond to some degree to a communicative action event (or message) type in external AOR models. For activity diagrams, there are two signal symbols: one for sent signals, and one for received signals, corresponding to the AORML distinction between communication acts (outgoing messages) and communication events (incoming messages). Strangely, the receipt of a signal is treated as an action that may follow any activity (which seems to denote the special action type *wait for signal*).

Active objects are another example of a class 'stereotype'. An active object is an object that "owns a thread and can initiate control activity" (cited from the OMG Unified Modeling Language Specification). Thus, active objects are rather an implementation, and not a domain, modeling concept. In a certain sense, they form a superclass of software agents, but they do not reflect the AORML distinction between agent and object.

While AOR modeling allows the tight integration of state and behavior modeling by means of reaction rules and interaction pattern diagrams, a similar integration is not possible in the UML. Only a rudimentary integration of state and behavior modeling is possible: in activity diagrams, objects and signals can 'flow' between activities. However, in [War], it is admitted that "Activity diagrams have an ill-defined connection with the other diagrams and are too limited in their expressibility."

Furthermore, the UML provides no predefined 'stereotypes' corresponding to commitments and claims. But it allows to express all the AORML entity subcategories AgentType, ActionType, EventType, CommitmentType, ClaimType, and ObjectType as user-defined stereotypes of the UML meta-concept CLASS, and the designated AORML relationship types (sends, receives, does, perceives, hasCommitment, and hasClaim) as stereotypes of the UML meta-concept ASSOCIATION. In this way we could define a *UML profile* for AOR modeling (in fact, we would need to define two profiles, one for external and one for internal AOR modeling). We would, however, have difficulties with expressing reaction rules since these rules are not expressible as UML 'stereotypes'. So, we could only cast the AOR state modeling fragment as a UML profile. The inclusion of reaction rules for AOR behavior modeling is not supported by the UML extension mechanisms.

UML and AORML are compared with each other in Table 2.

8.2.1 Agent UML

Recently, in [OvDPB00], an agent-oriented extension of UML, called *AUML*, mainly concerning sequence diagrams and activity diagrams, has been proposed. However, UML class diagrams are not modified, and no distinction between agents and objects is made in AUML.

8.3 Business Modeling

In many business modeling approaches, such as CIMOSA (see 8.3.1), the UML 1.3 *Profile for Business Modeling* (see 8.3.2), or the *Enterprise Ontology* (see 8.4), a distinction between passive and active entities is made, as in AOR modeling. However, none of the approaches discussed below includes a systematic treatment of agent-oriented meta-concepts such as actions, events, commitments and claims.

8.3.1 The Open System Architecture for Computer Integrated Manufacturing (CIMOSA)

In CIMOSA (see [AMI93]), an enterprise is viewed as a large collection of concurrent processes being ex-

(Extended) ER	UML	External AORML	Internal AORML
entity type	class active class sent signal received signal	object type agent type message type commitment/claim type	object type agent type outgoing message type incoming message type commitment type claim type
relationship type	association	association sends receives does perceives hasCommitment hasClaim	association isSentTo isReceivedFrom isPerceivedBy isCreatedBy hasCommitment Towards hasClaimAgainst
–	–	reaction rule	reaction rule
ER diagram	class diagram	agent diagram interaction frame diagram	 reaction frame diagram
–	sequence diagram	interaction sequence diagram	reaction sequence diagram
–	activity diagram state machine d.	activity diagram	activity diagram state machine d.
–	–	interaction pattern diagram	reaction pattern diagram

Table 2: A comparison of some important concepts of ER, UML and AORML.

AORML	*CIMOSA*	*Enterprise Ontology*	*PfBM*
agent	functional entity	actor	worker
action	functional operation	activity, action	–
event	event	–	–
object	enterprise object	entity	entity
reaction rule	behavioral rule	–	–

Table 3: Comparing the basic terms of AORML with CIMOSA, the Enterprise Ontology and the UML Profile for Business Modeling (PfBM).

ecuted by agents (called 'functional entities') in the various functional areas (called 'domains') of the enterprise. A 'domain process' is described as 'a complete chain of activities flowing through the enterprise' ([BV99]) that is triggered by one or more events and further decomposed into subprocesses (called 'business processes') and/or elementary process steps (called 'enterprise activities').

A restricted form of reaction rules, called 'procedural rules' (and more recently 'behavioral rules'), is used to specify business process steps. These rules have the form

`WHEN` event `DO` action

where the event expression typically refers to the ending status of some activity, such as in the following rule

`WHEN` `ES`(ea1) = ok `DO` ea2.

specifying that the enterprise activity *ea2* is started when the ending status of the enterprise activity *ea1* is 'ok'.

Some CIMOSA concepts have a direct correspondence to AORML concepts, as shown in Table 3. Unlike the AORML, CIMOSA does not provide any graphical language for visualizing its textually specified models.

8.3.2 The UML Profile for Business Modeling

The UML 1.3 standard contains a *UML Profile for Business Modeling* that defines the following UML::Class stereotypes: 'worker', 'case worker', 'internal worker', and 'entity'. A ≪Worker≫ is "an abstraction of a human that acts within the system". Although it is not clear what "system" means here, the concept of a worker seems to correspond to the AORML concept of an internal human agent. While an ≪Internal Worker≫ does not interact with actors outside the system, a ≪Case Worker≫ does. All other (passive) business objects are called ≪Entity≫. In addition, the concepts *organization unit* and *work unit* are proposed as UML::Subsystem stereotypes. An ≪Organization Unit≫ is "a subsystem corresponding to an organization unit of the actual business"; it "contains organization units, work units, classes (workers and entities), and relationships"; thus, it corresponds to the AORML concept of an internal institutional agent. A ≪Work Unit≫ is "a subsystem that contains one or more entities"; it is "a task-oriented set of objects that form a recognizable whole to the end user".

The UML Profile for Business Modeling seems to be a rather ad-hoc proposal for making a distinction between active and passive 'business objects' and for resolving some of the conceptual difficulties arising from the UML definition of an ≪Actor≫. While it shares some of its motivations with the AOR metamodel, it is, in many respects, quite incomplete. For instance, the only specific semantics assigned to ≪Worker≫ (by means of well-formedness rules for associations) is that they may ≪communicate≫ with each other and may ≪subscribe≫ to an ≪Entity≫. However, it is not explained, what these special associations mean.

8.3.3 The Eriksson-Penker Business Extensions

In [EP99], Eriksson and Penker propose an approach to business modeling with UML based on four primary concepts: resources, processes, goals, and rules. In this proposal, there is no specific treatment of agents. They are subsumed, together with "material, information, and products" under the concept of *resources*. This unfortunate subsumption of human agents under the traditional 'resource' metaphor prevents a proper treatment of many agent-related concepts such as commitments, authorization, and communication/interaction.

A business process, in the Eriksson-Penker approach, is viewed as a sequence of activities, and a business process type is modeled by means of a UML activity diagram, while in AOR modeling, a business process is viewed as a sequence of events and actions/activities, and a business process type is specified by a set of reaction rules.

8.4 The Enterprise Ontology

The *Enterprise Ontology* was developed within the *Enterprise Project*, a collaborative effort to provide a framework for enterprise modeling, led by the AI Applications Institute at the University of Edinburgh (see [UKMZ98]). It consists of definitions for nearly 100 terms, starting with the fundamental concepts of its 'meta-ontology' (*entity*, *relationship* and *actor*), both in natural language and in the formalism of *Ontolingua*. The latter formalization is supposed to support reasoning about enterprises.

For simplicity, the distinction between an entity (instance) and an entity type (class) is avoided. *Actors* are defined as special entities that can play an actor role in certain relationships (such as performActivity, haveCapability, etc.).

In order to give a flavor of this work, we present some of the main concept definitions proposed.

An *activity* is an entity that is characterized by being performed by one or more actors over a particular *time interval*, having *pre-conditions* and *effects*, possibly being decomposable into more detailed *subactivities*, possibly using and/or consuming *resources*, being *owned* by an actor on behalf of whom the activity is performed.

There is no independent concept of an event: events are defined as "a kind of activity".[12] Synonyms of *activity* are: behavior, task, action.

A *person* is a human actor. A *machine* is a non-human actor. A *corporation* is a group of *persons* recognized in law as having existence, rights and duties distinct from those of the individual persons who comprise the group. A *legal entity* is either a *person* or a *corporation*.

The following points highlight some shortcomings of the Enterprise Ontology:

1. For analysis and design modeling, it is essential to distinguish between entities and entity types.
2. It seems to be questionable to view natural forces that cause certain events to happen, such as gravity, as actors.
3. Events should not be subsumed under activities. Rather, they should be first-class citizens.
4. Like in the UML, activities should be distinguished from actions which are conceived at the lowest level of temporal granularity, that is, as instantaneous events without duration.
5. The concepts of commitments/claims and duties/responsibilities are missing.

[12] Remarkably, the authors consider also events which take place as a result of natural necessity (such as "water flowing down a hill") as activities of *inanimate actors* (such as gravity).

8.5 Other Related Work

Agent-oriented modeling techniques are still in a very early stage. The *Resource-Event-Agent (REA)* modeling framework of [McC82] can be regarded as an early predecessor of agent-oriented information systems modeling. It was proposed as a new approach to accounting systems that aims at reconciling the specialist accounting view of enterprise resource management with the more general views of other business areas. In accounting, special attention is paid to economic resources that are subject to financial and managerial accounting requirements. These resources are associated with increment and decrement events which are, in turn, associated with economic agents. While the REA framework suggests to distinguish between the entity categories of resources, events and agents, it does not provide any visual or formal modeling language that allows to map these conceptual distinctions. AORML, by providing such visualization and formal language constructs, seems to support the REA accounting framework very well.

In [Yu95, YM95], an agent-oriented modeling framework, called $i^\star$, for early *requirements engineering* is proposed stressing the role of *dependencies* between agents. Since the AORML provides a (visual) language for *designing* information systems, it may be a possible target language for transforming $i^\star$ models into it.

In [KGR96], a methodology for the analysis and design of multiagent systems based on object-oriented modeling principles is presented, requiring compliance with the particular paradigm of 'Belief-Desire-Intention (BDI)' agents proposed in [RG91]. More general approaches, considering issues such as agent roles, rights and duties, contracts and communication protocols, are proposed in [EL99, WJK00]. However, these approaches provide no diagram language and their relationship to the UML is not clear.

A conceptual framework for agent-oriented workflow modeling based on agent roles and the communication protocols, qualifications, and rights and duties associated with them, is proposed in [YS99]. In the business-rules-centered approach to the modeling of agent-oriented information systems of [Tav99], software agents are used to represent "functional business units/actors and also external units/actors like customers or suppliers". The business case of a car rental company is used to demonstrate the agent-based implementation of business rules.

8.6 Strengths and Weaknesses of AOR Modeling

The main strengths of AORML with respect to conceptual modeling are:

1. AORML has a richer set of basic ontological concepts, allowing to capture more semantics of a domain, as compared to ER, UML and AUML.

2. AORML includes and unifies many of the fundamental domain modeling concepts found in enterprise modeling approaches such as CIMOSA and the *Eriksson-Penker business extensions*.

3. Unlike the UML, AORML allows to integrate state and behavior modeling in one diagram.

4. AORML allows to include the deontic concepts of *rights* and *duties* for organization modeling in an ER/UML-based information model.

5. AORML seems to be the first approach that allows to systematically distinguish between *external* and *internal* models, and to account for the phenomenon of *internalization*.

6. AORML seems to be the first approach that employs and visualizes the important concept of *reaction rules* for behavior modeling.

Weaknesses of AOR modeling in its current form include:

1. The entire development path from analysis to implementation is not fully defined yet.

2. AORML does currently not include the concept of *activities*. It will be added, however, in future work (a first sketch can be found in [TW01b]).

3. AORML does currently not include the concept of *goals* which is fundamental in several other approaches, such as [YM95, EP99].

4. AORML does currently not allow to model the *proactive* behavior of agents. This type of behavior, which is the focus of Artificial Intelligence approaches to agents, is based on action planning and plan execution for achieving goals.

9 Conclusion

Similar to 'object', the term 'agent' denotes an abstraction that leads to more natural and more modular software concepts. It helps to capture more semantics about natural and artificial systems an information system has to represent and to interact with. We have presented an agent-oriented approach to state and behavior modeling that allows an integrated treatment of the static, dynamic and deontic aspects of these systems, and thus offers a methodology that is semantically richer than many other approaches.

In future work we plan to extend the AOR metamodel by adding the meta-concepts of *activities* and *goals*. We also plan to develop a suitable method for agent-oriented requirements engineering. Furthermore, in two ongoing research projects (see [AGV, Coh]), we develop extensions and tools for modeling and running *simulations* of socio-technical, economic and social systems based on the AOR metamodel. For up-to-date information on AOR modeling, see `www.AOR.rezearch.info`.

Acknowledgements

I am grateful to Kuldar Taveter and to the anonymous referees for their valuable comments on preliminary versions of this article.

References

[AGV] AGV transport systems as coooperative and adaptive multiagent systems. Research project. http://www.inf.fu-berlin.de/inst/ag-ki/projects/fts/.

[AMI93] ESPRIT Consortium AMICE, editor. *CIMOSA – Open System Architecture for CIM*. Springer-Verlag, 2nd edition, 1993.

[BV99] G. Berio and F.B. Vernadat. New developments in enterprise modeling using CIMOSA. *Computers in Industry*, 40:99–114, 1999.

[Che76] P. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 1970.

[Coh] Coherence in automated and semi-automated interaction processes. Research project. http://tmitwww.tm.tue.nl/staff/gwagner/coherence.

[CPF+99] Y. Chen, Y. Peng, T. Finin, Y. Labrou, S. Cost, B. Chu, R. Sun, and B. Wilhelm. A negotiation-based multi-agent system for supply chain management. In *Proc. of Workshop on Agent based Decision-Support for*

Managing the Internet-Enabled Supply-Chain, at Third Conference on Autonomous Agents (Agents-99), Seattle, WA, May 1999.

[EL99] M. Elammari and W. Lalonde. An agent-oriented methodology: High-level and intermediate models. In G. Wagner and E. Yu, editors, *Proc. of the 1st Int. Workshop. on Agent-Oriented Information Systems*, 1999.

[EP99] H.E. Eriksson and M. Penker. *Business Modeling with UML: Business Patterns at Work*. John Wiley & Sons, 1999.

[FBT00] M.S. Fox, M. Barbuceanu, and R. Teigen. Agent-oriented supply chain management. *Int. J. of Flexible Manufacturing Systems*, 12:165–188, 2000.

[FMHS96] K. Fischer, J.P. Mueller, I. Heimig, and A.-W. Scheer. Intelligent agents in virtual enterprises. In *Proc. of PAAM96*, pages 205–223, 1996.

[GK94] M.R. Genesereth and S.P. Ketchpel. Software agents. *Communication of the ACM*, 37(7):48–53, 1994.

[GSP00] J. Gjerdrum, N. Shah, and L.G. Papageorgiou. A combined optimization and agent-based approach for supply chain modeling and performance assessment. *Production Planning and Control*, 12:81–88, 2000.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HR95] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72:329–365, 1995.

[Jac94] I. Jacobson. *The Object Advantage*. Addison-Wesley, Workingham (England), 1994.

[KGR96] D. Kinny, M. Georgeff, and A. Rao. A methodology and modeling technique for systems of BDI agents. In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away*, volume 1038 of *Lecture Notes in Artificial Intelligence*, pages 56–71. Springer-Verlag, 1996.

[McC82] W.E. McCarthy. The REA accounting model: A generalized framework for accounting systems in a shared data environment. *The Accounting Review*, LVII(3):554–578, July 1982.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[OvDPB00] J. Odell, H. van Dyke Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proc. of the 2nd Int. Workshop on Agent-Oriented Information Systems*, Berlin, 2000. iCue Publishing.

[RG91] A.S. Rao and M.P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. KR-91*, San Mateo (CA), 1991. Morgan Kaufmann.

[Sea95] John R. Searle. *The Construction of Social Reality*. Free Press, New York, 1995.

[Sho93] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[Sin99] M.P. Singh. An ontology for commitments in multiagent systems. *Artificial Intelligence and Law*, 7:97–113, 1999.

[SM96] M. Stonebraker and D. Moore. *Object-Relational DBMS*. Morgan Kaufmann Publishers, San Francisco, 1996.

[Tav99] K. Taveter. Business rules approach to the modeling, design and implementation of agent-oriented information systems. In G. Wagner and E. Yu, editors, *Proc. of the 1st Int. Workshop on Agent-Oriented Information Systems*, 1999.

[TW01a] K. Taveter and G. Wagner. Agent-oriented enterprise modeling based on business rules. In *Proc. of 20th Int. Conf. on Conceptual Modeling (ER2001)*, pages 527–540, Yokohama, Japan, November 2001. Springer-Verlag. LNCS 2224.

[TW01b] K. Taveter and G. Wagner. A multi paradigm methodology for modelling inter-enterprises business processes. In *Proc. of 2nd Int. Workshop on Conceptual Modeling Approaches for e-Business, held in conjunction with the*

20th Int. Conf. on Conceptual Modeling (ER2001), pages 205–223, Yokohama, Japan, November 2001. Springer-Verlag.

[UKMZ98] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13, 1998.

[Wag98] G. Wagner. *Foundations of Knowledge Systems – with Applications to Databases and Agents*, volume 13 of *Advances in Database Systems*. Kluwer Academic Publishers, 1998. See http://www.inf.fu-berlin.de/∼wagnerg/ks.html.

[War] J. Warmer. The future of UML. http://www.klasse.nl/english/uml/uml2.pdf.

[WJK00] M. Wooldridge, N.R. Jennings, and D. Kinny. The GAIA methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3:285–312, 2000.

[WS00] J.C. Wortmann and N.B. Szirbik. ICT issues among collaborative enterprises: from rigid to adaptive agent-based technologies. *Production Planning and Control*, 12(5):452–465, 2000.

[WW99] Y. Wand and C.C. Woo. Ontology-based rules for object-oriented enterprise modeling. Technical Report 99-MIS-001, Faculty of Commerce and Business Administration, Univ. of British Columbia, 1999.

[YM95] E. Yu and J. Mylopoulos. From E-R to 'A-R' – modeling strategic actor relationships for business process reengineering. *Int. J. of Intelligent and Cooperative Information Systems*, 4(2-3):125–144, 1995.

[YS99] L. Yu and B.F. Schmid. A conceptual framework for agent-oriented and role-based workflow modeling. In G. Wagner and E. Yu, editors, *Proc. of the 1st Int. Workshop. on Agent-Oriented Information Systems*, 1999.

[Yu95] E.S.K. Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, Computer Science Department, Univ. of Toronto, Toronto (Canada), 1995.