Chapter X

Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modelling

Kuldar Taveter
VTT Information Technology, Finland

Gerd Wagner
Eindhoven University of Technology, The Netherlands

Abstract

This chapter proposes a new agent-oriented software engineering process called RAP, which follows the Rational Unified Process (RUP) in many ways, but is based on Agent-Object-Relationship (AOR) modelling instead of object-oriented modelling. The chapter briefly presents the foundational ontology that supports the methodology and introduces the RAP/AOR viewpoint modelling framework. It then describes the modelling from the interaction, information, and behavior aspects of the framework by using a case study of business-to-business electronic commerce. Finally, the chapter describes an implementation approach based on the Model Driven

Architecture (MDA) and Extended Markup Language (XML). The methodology is aimed at the creation of distributed socio-technical systems consisting of both humans and technical, including software, components that may, in turn, include software agents.

Introduction

A *Radical Agent-Oriented Process* (RAP) defines a software engineering process[1] using the Agent-Object-Relationship (AOR) modelling language proposed by Wagner (2003a). In AOR modelling, the agents in a problem domain are distinguished from the non-agentive objects. The agents' actions, event perceptions, commitments, and claims, as well as their rights and duties, are explicitly included in the models.

The RAP/AOR methodology is based on Wagner (2003a) and Taveter (2004a). Wagner (2003a) presents an agent-oriented approach to the conceptual modelling of organizations and organizational information systems, called AOR modelling, 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, and generalization relationship types of Entity-Relationship (ER) and UML class modelling. Business processes are viewed as social interaction processes emerging from the behaviour of the participating agents. In the proposed approach, behaviour is primarily modelled by means of interaction patterns expressed in the form of reaction rules that are visualized in interaction pattern diagrams.

Taveter (2004a) proposes an integrated business modelling technique – the Business Agents' Approach – that is based on AOR modelling. Taveter (2004a) emphasizes that in addition to being a technological building block, an agent is an important modelling abstraction that can be used at different logical levels in the creation and development of an information system. The Business Agents' Approach suggests an elaboration of the existing business modelling frameworks – six perspectives of agent-oriented business modelling for distributed domains. These perspectives are the organizational, informational, interactional, functional, motivational, and behavioural perspective. The Business Agents' Approach covers modelling from all the perspectives mentioned by employing a combination of goal-based use cases, the AOR Modelling Language (AORML), and Object Constraint Language (OCL), forming a part of UML 2.0 (OMG, 2003b). The Business Agents' Approach also extends the graphical notation of AORML by activity diagrams that are executable and enable to represent models of several or all perspectives in just one diagram.

The RAP/AOR methodology can be viewed as an agent-oriented refinement of the *Unified Software Development Process* proposed by Jacobson, Booch, and Rumbaugh (1999) and its commercial complement, the *Rational Unified Process* (RUP) (Kruchten, 1999). It aims at achieving more *agility* than the RUP by using simulation for early testing of analysis and design models.

Agile methodologies, such as the *Extreme Programming* proposal of Beck (1999), have received much attention recently (Fowler, 2003). They emphasize the value of lightweight ad hoc processes based on test-case-based development and rapid prototyping and de-emphasize the value of detailed modelling on which they blame the heavy weight and inflexibility of traditional methodologies and the RUP. While we acknowledge the significance of agility, we disagree with their analysis that blames modelling as the source of inflexibility. Rather, we agree with the *Model-Driven Architecture* (MDA, http://www.omg.org/mda/) approach of the Object Management Group (OMG) where modelling is identified as the core of state-of-the-art software engineering that is scientifically well-founded. When a model-driven approach includes early testing of models by means of simulation, agility is achieved even without setting a focus on code and rapid prototyping.

We are aware of two other agent-oriented methodologies that also claim to follow the RUP.

The ADELFE methodology described in (Bernon, Glaizes, Peyruqueou, & Picard, 2002) is targeted at the engineering of adaptive multi-agent systems to be used in situations where the environment is unpredictable or the system is open. This niche methodology is based on the Adaptive Multi-Agent Systems (AMAS) theory (Capera, Georgé, Gleizes, & Glize, 2003) and shares, with RAP/AOR, prototyping by simulation.

The MESSAGE methodology (Evans et al., 2001) has been developed for the particular needs of the telecommunications industry. The analysis model views of MESSAGE can be compared to the viewpoint aspects of RAP/AOR: the organization view and interaction view are subsumed under the interaction viewpoint aspect, the goal/task view and agent/role view roughly correspond to the behaviour aspect, and the domain view corresponds to the information aspect. Although both MESSAGE and RAP/AOR have business processes as a starting point, for business process modelling MESSAGE employs UML activity diagrams that are, in our opinion, not truly agent-oriented. The reason is that it is questionable to model the invocations of activities in the spheres of responsibility of different "actor objects" (resp. agents) as state transitions, as it is done between "swim lanes" in activity diagrams because "actor objects," which are independent of each other, do not share states.

Unlike these two RUP-based agent-oriented methodologies, RAP/AOR is more concerned with distributed agent-based information systems (such as business

process automation and supply-chain management systems) for the business domain and not so much with Artificial Intelligence (AI) systems.[2] This difference implies that we are not so ambitious about capturing human-like intelligence features such as *desires* and *intentions*, or sophisticated forms of proactive behaviour. Rather, in RAP/AOR we focus on declarative models of communication and interaction founded on reactive behaviour and on the basic mental state components of *beliefs*, *perceptions*, and *commitments*.

Note that perceptions, as the mental state component that is the basis of communication and interaction, have been neglected in the popular Belief-Desire-Intention (BDI) model (Rao & Georgeff, 1991). Following the logic-based AI tradition, which is only interested in reasoning/thinking and tends to ignore other cognitive aspects and activities, the BDI paradigm has treated beliefs and reasoning as primary and perceptions and communication as secondary. However, as has been pointed out in the speech act theory (Austin, 1962), in reality, it is the other way around: communication is primary, and the concepts of beliefs and assertions are based on communication. In some BDI approaches, perceptions are indeed treated as beliefs. However, this is clearly unsatisfactory, both conceptually and technically. The perception of an event can indeed be mapped into a corresponding "event has happened" belief; but we cannot assume that this is the case with all perceptions, since this would not be the case with irrelevant perceptions and would lead to an overflow of the belief/knowledge base of an agent. Conceptually, perceptions are transient (and are consumed by the attention process), while beliefs are persistent.

In comparison with recent techniques and notations for creating executable business process specifications based on Web Services (WS) (http://www.w3.org/2002/ws/), such as BPEL4WS (*http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/*) and BPML (BPML, 2002), the difference between an approach based on WS and an agent-oriented approach like RAP/AOR should first be clarified. Even though Web Services Description Language (WSDL) (W3C, 2004) allows the representation of a message sequence consisting of, for example, receiving a message and replying to it, this is far from the dynamics of agent communication protocols like Contract Net standardized by the Foundation for Intelligent Physical Agents (FIPA) http://www.fipa.org/). Secondly, in principle, an interface to a software agent can be embedded in a WS-based interface that seems to be in line with the Web Services Architecture (WSA) of the World Wide Web Consortium (W3C, 2003), according to which services are provided by software agents.

An essential objective of the RAP/AOR methodology is to enhance team productivity by agent-based work process management including both (well-structured) workflows and spontaneous interactions among team members and their software assistants. This issue is not covered in the present version of RAP/AOR but will be the topic of future work.

The RAP/AOR Methodology

The RAP/AOR methodology emphasizes the importance of precise modelling of the problem domain at hand for achieving an agent-based information system with the required functionality. The preciseness of domain modelling is ensured by the ontological foundation of the RAP/AOR methodology. It defines concepts such as physical agent, non-agentive object, action event, social moment, and institutional agent. Based on the ontological foundation, AORML introduces a graphical notation for modelling the structure of an agent's mental state, comprising the modelling of agent types, their internal and external object types, action and non-action event types, and commitment/claim types. Additionally, AORML introduces reaction rules as its most important behaviour modelling element.

The RAP/AOR viewpoint modelling framework enables viewing and modelling a problem domain from the interaction, information, and behaviour viewpoint aspects. Different kinds of models/diagrams are used for modelling from different aspects, such as agent diagrams and interaction-frame diagrams of AORML and use-case diagrams of UML from the interaction aspect, agent diagrams from the information aspect and goal-based use-case models, and interaction-pattern diagrams and AORML activity diagrams from the behaviour aspect. The RAP/AOR viewpoint modelling framework also distinguishes between the abstraction levels of conceptual domain modelling, computational design, and implementation. At the level of conceptual domain modelling, we adopt the perspective of an external observer who is observing the (prototypical) agents and their interactions in the problem domain under consideration. At the levels of computational design and implementation, we adopt the internal (first-person) view of a particular agent to be modelled and implemented, for example, of an *agentified information system* (i.e., an information system represented as one or more software agents).

Ontological Foundations of the RAP/AOR Methodology

The ontological foundation of the RAP/AOR concepts is provided by the Unified Foundational Ontology (UFO) proposed by Guizzardi and Wagner (2004). In addition to a foundation layer, called UFO-A, and the perdurant ontology layer UFO-B, UFO includes an agent ontology layer, UFO-C, which is the basis of AORML. UFO-C is summarized in the form of a UML class diagram in Figure 1, and in the form of a controlled English vocabulary presented below. While beliefs and perceptions are categorized as *mental moments* (endurants that existentially depend on one agent, their "bearer"), commitments are categorized as *social moments* (endurants that existentially depend on several agents).

We include this summary of UFO in this chapter because we want to give the reader an idea of what a foundational ontology is and how it can help in establishing a modelling method. The reader should know, however, that, lacking the necessary space, we cannot treat this topic in greater depth here.

The most important concepts for the purposes of the RAP/AOR methodology, presented in controlled English, are:

- **physical agent**: *physical object* that creates *action events* affecting other *physical objects*, that perceives *events*, possibly created by other *physical agents* and to which we can ascribe a mental state

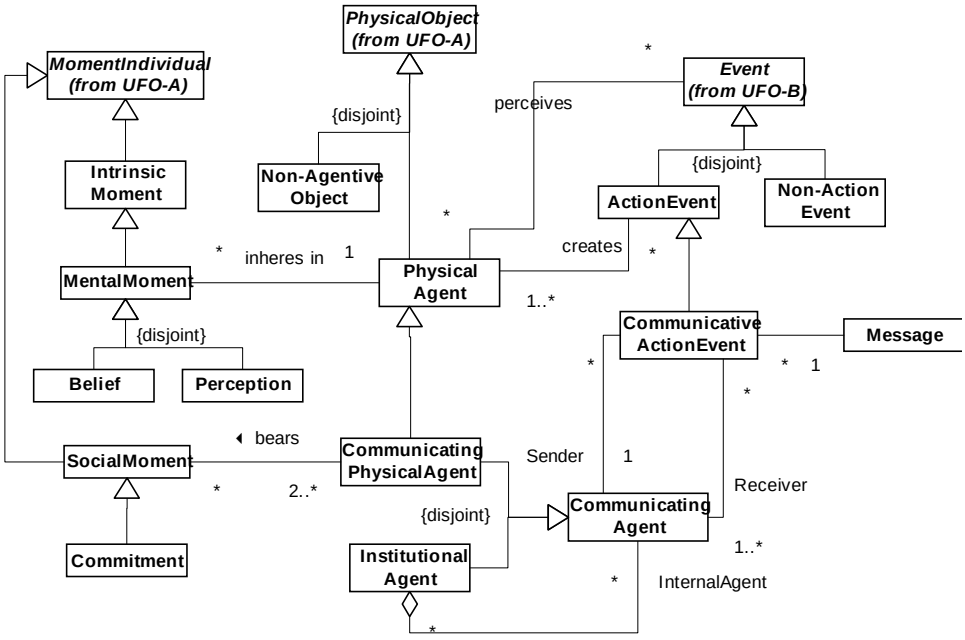
  Examples: a dog; a human; a robot
- **action event**: *event* that is created through the action of an *agent*
- **agent creates action event**: designated *relationship*
- **agent perceives event**: designated *relationship*
- **non-agentive object**: *physical object* that is not a *physical agent*

  Examples: a chair; a mountain
- **communicating agent**: *agent* that creates *communicative action events* directed to other *communicating agents* and that perceives *communicative action events* that possibly lead to changes in its mental state
- **social moment**: *moment individual* that is existentially dependent on more than one *communicating agent*

  Examples: a commitment; a joint intention
- **communicative action event**: *action event* by which a *communicating agent*, the *sender*, sends a *message* to one or more other *communicating agents*, the *receivers*
- **message**: *social moment* that is exchanged between *communicating agents* in a *communicative action event*
- **communicating agent** sends **message** to **communicating agent**: designated *relationship*

  Inverse relationship: *communicating agent* receives *message* from *communicating agent*
- ***sender***: role name that refers to the first argument of the *communicating agent sends message to communicating agent relationship* type
- ***receiver***: role name that refers to the second argument of the *communicating agent sends message to communicating agent relationship* type
- **institutional agent**: social fact (Searle, 1995) that is an aggregate consisting of *communicating agents* (its internal agents, which share a

Figure 1. The main categories of UFO-C described as a MOF/UML model



collective mental state), and that acts, perceives, and communicates through them

Examples: a business unit; a voluntary association

- **agent**: *endurant* that is either a *physical agent*, an *institutional agent* or a *software agent*

Note that the term *agent* is defined as an abstraction that subsumes physical agents, social agents, and software agents. Since software entities are difficult to understand ontologically (in which sense does a software entity exist in the real world?), the category of software agents is not included in the current version of UFO. But even if no ontological account of software agents is provided by the foundational ontology, it may still be of great value for a software engineering method, such as RAP/AOR, since it can help to motivate and explain the choice of its modelling constructs and to provide guidelines on how to use a modelling language. For instance, UFO provides an account of the meaning of *roles* on the basis of its distinction between *role type* and *base type* (see the following). Role modelling is an important issue in all agent-oriented modelling methods.

UFO distinguishes between different kinds of entity types as shown in Figure 2. These distinctions are defined as follows:

- **sortal type**: *type* that provides an identity criterion for determining if two instances are the same

  Examples: Person; Student; City
- **mixin type**: *type* that is not a *sortal type* and can be partitioned into disjoint subtypes, which are *sortal types* with different identity criteria

  Examples: Agent; Customer; Product
- **base type**: *sortal type* that is rigid (all its instances are necessarily its instances)

  Examples: Mountain; Person
- **phase type**: *sortal type* that is anti-rigid (its *instances* could also not be *instances* of it without losing their identity) and that is an element of a subtype partition of a *base type*

  Examples: Town and Metropolis are *phase subtypes* of City; Baby, Teenager and Adult are *phase subtypes* of Person
- **role type**: *sortal type* that is anti-rigid and for which there is a *relationship* type such that it is the *subtype* of a *base type* formed by all *instances* participating in the *relationship* type

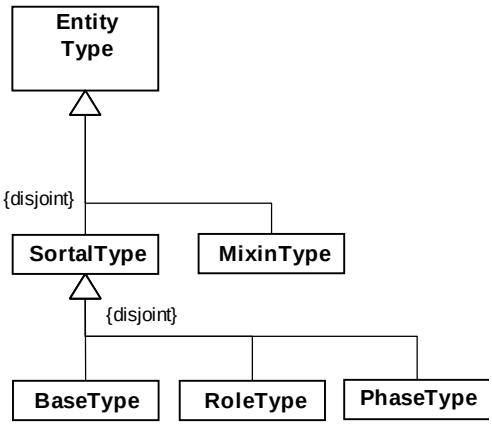
  Examples: DestinationCity as a *role subtype* of City; Student as a *role subtype* of Person

We understand a mixin type as a union of other types that does not have a uniform identity criterion for all its instances. Many mixin types are *role mixin types*, that is, mixin types that can be partitioned into role subtypes. For instance, Customer is a role mixin type: it can be partitioned into PersonalCustomer and CorporateCustomer, both of which are role subtypes (of Person and Corporation, respectively).

Note that an *actor* (more precisely, *actor type*) is an agent role type. For instance, the actor type CEO is a role subtype of the base type Person. In many cases, an *actor type* is an agent role mixin type. For instance, the actor type BookingClerk can be partitioned into HumanBookingClerk (being a role subclass of Person) and SoftwareAgentBookingClerk (being a role subclass of SoftwareAgent).

In the RAP/AOR methodology, we view the autonomy of an agent as a relative rather than an absolute characteristic. An institutional agent consisting of internal agents is thus autonomous in relation to other institutional agents, at least

Figure 2. Different kinds of types in UFO-A



to some degree. The autonomy of an internal agent is even not always desired in systems of the kind this methodology targets. For example, an autonomous agent could reject commitments arising from its duties, which is something we would not want for a communicating internal agent (e.g., a software agent) forming a part of an institutional agent (e.g., a company). However, we may allow that an internal agent makes its own prioritization decisions, which could also be viewed as a kind of autonomy. We thus understand an internal software agent as a kind of communicating decision-support system that makes decisions or proposes them to a human user based on the information retrieved from other enterprise systems of the company.

The AOR Modelling Language

AORML is used as the main graphical description for work products of RAP/AOR. We thus describe it first as a prelude to our use of it in the methodology itself.

AORML is based on an ontological distinction between active and passive entities, that is, between agents and (non-agentive) objects of the real world. The agent metaphor subsumes *artificial* (software and robotic), *natural* (human and animal), and *social/institutional* agents (groups, organizations, etc.).

In AORML, an entity is an agent, an event, an action, a claim, a commitment, or an ordinary object. Only agents can communicate, perceive, act, make commitments, and satisfy claims. Objects are passive entities with no such capabilities.

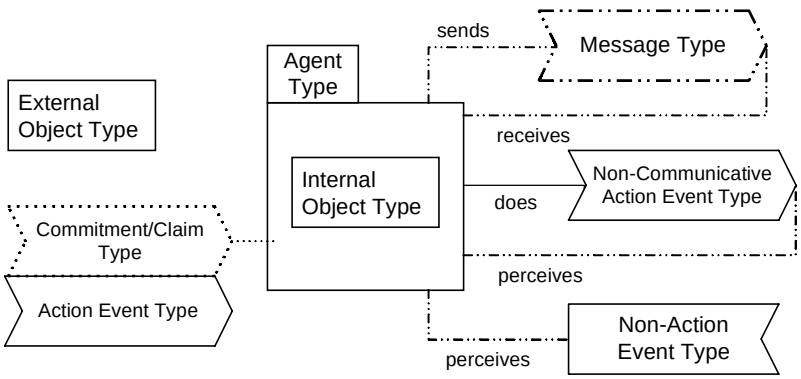
In addition to human and artificial agents, AORML also includes the concept of *institutional agents*, which are composed of a number of other agents that act on their behalf. Organizations and organizational units are important examples of institutional agents.

There are two basic types of AOR models: *external* and *internal* models. An external AOR model adopts the perspective of an external observer who is looking at 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 modelled. While a (business) domain model corresponds to an external model, a design model (for a specific agentified information system) corresponds to an internal model that can be derived from the external one.

Figure 3 shows the most important elements of external AOR mental state structure modelling. In order to restore compatibility with the diagram notation of UML 2.0 (OMG, 2003b), the graphical shape for an agent type has been changed in this chapter from a round-cornered rectangle, which has been used previously but which is now the UML symbol for an action/activity, to the shape shown in Figure 3 (see also the agent types Buyer and Seller in Figure 4). In AORML, we have overloaded this shape, which stands for a "subsystem" in UML 2.0 (OMG, 2003b), with a different semantic by using it as a graphical symbol for the <<AgentType>> stereotype of the base class Class of UML. In other words, an agent type is viewed as a UML class instead of a "subsystem."

An *external AOR diagram*, as represented in Figure 3, depicts the agent types and instances (if applicable) of a problem domain, together with their internal agent types and instances, their beliefs about objects and external agents, and the relationships among agents and/or objects.

Figure 3. The core mental state structure modelling elements of external AOR diagrams



As has been shown by Wagner (2003b), mental state structure modelling in AORML can be defined as a UML Profile, that is, it is a conservative extension of UML class modelling.

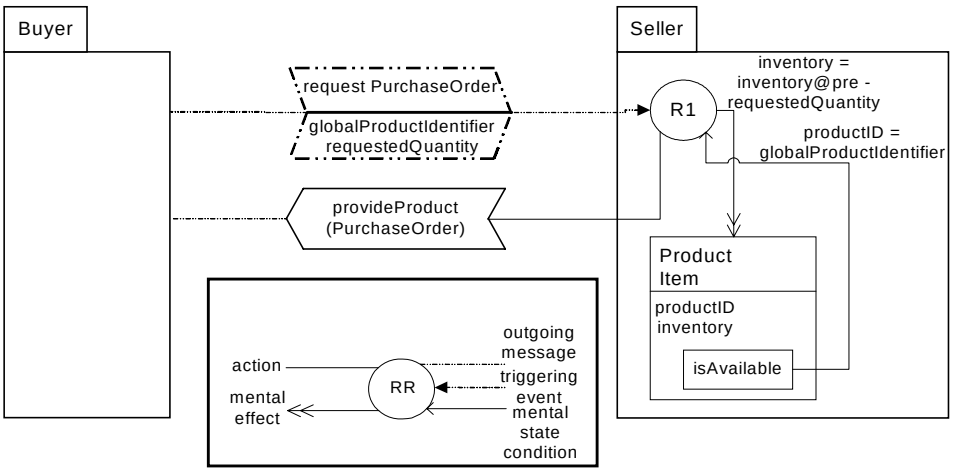
According to Figure 3, the graphical notation of AORML distinguishes between an *action event type* and a *non-action event type* and between a *communicative* action event (or *message*) type and a *non-communicative* action event type. Figure 3 also reflects that a *commitment/claim type* is coupled with the action event type whose instance fulfills the corresponding commitment (or satisfies the claim). Please note that for the sake of simplicity, in AORML, a communicative action event is identified with a message while in the UFO-C ontology depicted in Figure 1 they form different categories.

The most important behaviour modelling elements of AORML are *reaction rules*, which are used to express *interaction patterns*. In symbolic form, a reaction rule is defined as a quadruple ε, $C \rightarrow \alpha$, P where ε denotes an event term (the triggering event), C denotes a logical formula (the mental state condition), α denotes an action term (the triggered action), and P denotes a logical formula (specifying the mental effect or post-condition).

As is shown in the legend of Figure 4, 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 a solid arrowhead that specifies the triggering event type. In addition, there may be ordinary incoming arrows representing mental state conditions (referring to corresponding instances of other entity types). There are two kinds of outgoing arrows: one for specifying mental effects (changing beliefs and/or commitments) and one for specifying the performance of (physical and communicative) actions. An outgoing arrow with a double arrowhead denotes a mental effect. An outgoing connector to an action event type denotes the performance of an action of that type.

As an example, Figure 4 shows an *interaction pattern diagram* that specifies the reaction rule R1 for a Seller's behaviour in response to perceiving a communicative action event of the type request PurchaseOrder. Both the mental state condition and mental effect of the reaction rule are presented as expressions in the Object Constraint Language (OCL) of UML 2.0 (OMG, 2003b) attached to the corresponding arrows. The pre-condition for providing the Buyer with the requested product is the availability of the ProductItem whose checking in the internal database of the Seller is specified with the help of the corresponding status predicate isAvailable and the OCL expression productID = globalProductIdentifier. The post-condition representing the mental effect is expressed as inventory = inventory@pre - requestedQuantity. The post-condition affects the representation of the corresponding ProductItem in the Seller's internal database by decreasing its attribute inventory by the value of the attribute requestedQuantity of the message received.

Figure 4. An interaction pattern: when receiving a PurchaseOrder, the Seller provides the Buyer with a product according to the PurchaseOrder if there are products available.



The RAP/AOR Viewpoint Modelling Framework

The RAP/AOR viewpoint modelling framework described in Table 1 is based on the perspectives of agent-oriented business modelling proposed by Taveter (2004a) which is, in turn, rooted in the ideas of the Zachman framework (Sowa & Zachman, 1992). The RAP/AOR viewpoint modelling framework is also well-aligned with the MDA (http://www.omg.org/mda/) framework of OMG. It consists of a matrix with three rows representing different abstraction levels and three columns representing the viewpoint aspects *interaction*, *information*, and *behaviour*. Each cell in this matrix represents a specific viewpoint, such as *Conceptual Interaction Modelling*, *Computational Information Design*, or *Behaviour Implementation*.

In the literature, the concept of viewpoints has been used differently in different approaches. For example, the MDA defines only three viewpoints: computation-independent modelling (CIM), platform-independent modelling (PIM), and platform-specific modelling (PSM). We consider these viewpoints as viewpoint dimensions and call them *abstraction levels*. Another approach, the Reference Model for Open Distributed Processing (RM-ODP) (Putnam & Boehm, 2001) defines five viewpoints. The correspondences between the MDA and RM-ODP viewpoints and the Zachman and the RAP/AOR viewpoint modelling frameworks are summarized in Table 2.

Table 1. The RAP/AOR viewpoint modelling framework

Viewpoint models	***Viewpoint aspect***		
Abstraction level	*Interaction*	*Information*	*Behaviour*
Conceptual Domain Modelling	AOR Agent Diagrams, UML Use Case Diagrams, AOR Interaction Frame Diagrams, AOR Interaction Sequence Diagrams	AOR Agent Diagrams	AOR Interaction Pattern Diagrams, Goal-Based Use Case Models, AOR Activity Diagrams
Platform-Independent Computational Design	UML Use Case Diagrams, AOR Reaction Frame Diagrams, User Interface Design Models, Security Models	AOR Agent Diagrams	AOR Reaction Pattern Diagrams, AOR Internal Activity Diagrams
Platform-Specific Implementation	UML Deployment Diagrams	UML Class Diagrams	UML Class Diagrams

Table 2. RAP/AOR stakeholders and the corresponding viewpoint names in other frameworks

Abstraction level	*Audience/Stakeholders*	*Viewpoint Names*		
		MDA	*RM-ODP*	*Zachman*
Conceptual Domain Modelling	owners/customers, users, domain experts	CIM	Enterprise	Rows 1+2
Computational Design	systems analysts, software architects	PIM	Information + Computational	Row 3
Implementation	programmers, database implementers, system integrators	PSM	Engineering + Technology	Rows 4+5

Normally, in a software engineering project, one or more views are created for each viewpoint, using the respective modelling language(s). A *view* is a diagram or a model of another kind, like a tabular use case or a textual description. Thus, a viewpoint-modelling framework defines the collection of engineering documents created and used in a software engineering project. In the following, we briefly describe different viewpoints of the framework

Domain-Interaction Viewpoint

The domain-interaction viewpoint (Column 1 in Table 1) concerns the analysis and modelling of *active entities*, that is, of agent types and instances and relationships, as well as the *interactions* and *communication* between them. The domain-interaction viewpoint comprises organization modelling. The purposes of organization modelling are to identify:

a) the organization(s) of the problem domain;
b) the relevant organizational units that of which each organization to be modelled consists;
c) the *roles*[3] included by the organizational units; and
d) the types of relationships occurring between these agent types.

According to Zambonelli, Jennings, and Wooldridge (2001), among the five types of relationships that can be identified between institutional agent types and/or role types, control, benevolence, and dependency relationships are the most relevant ones to modelling interactions between agents. *Control* relationships identify the authority structures within an organization. *Benevolence* relationships identify agents with shared interests. *Dependency* relationships exist between agents because of resource restrictions where the depender depends on the dependee for the availability of a physical or an informational resource. For example, in Figure 5 there is the isBenevolentTo relationship between instances of the role types Seller and Buyer.

The domain-interaction viewpoint is described with the help of four views:

1. AOR Agent Diagrams;
2. UML Use Case Diagrams;
3. AOR Interaction Frame Diagrams; and
4. AOR Interaction Sequence Diagrams.

Types of organizational units and roles can be represented by AOR agent diagrams where different agent types may relate to each other through the relationships of *generalization* and *aggregation*. An *AOR agent diagram* depicts the agents and agent types of a problem domain, together with their internal agents and agent types and the relationships among them. An agent diagram, like the one shown in Figure 5, includes all agent (role) types of a business domain. An important purpose of an agent diagram is to describe all *stakeholders* that are involved in the business processes to be supported by the socio-technical business system and to give an overview of the business system viewed as an MAS.

A *use case* is defined as the specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system (OMG, 2003b). Since the system is itself an actor[4] (Cockburn, 1997a), the model of interaction between an "actor" and a "system" of a use case can also be applied in agent-oriented modelling. A *UML use-case diagram*, for example, the diagram represented in Figure 6, shows the relationships among

Figure 5. The agent diagram of the domain of business-to-business electronic commerce

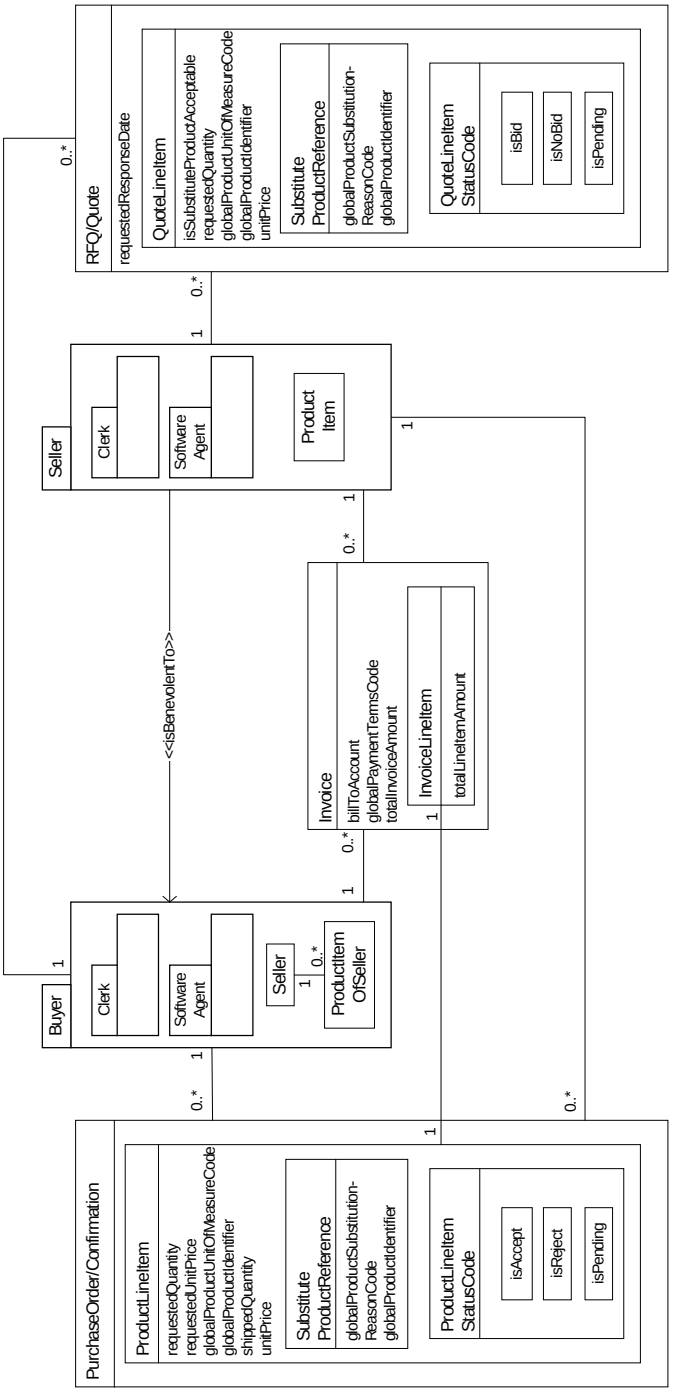


Figure 6. A use case diagram showing the types of interactions between agents of the types Buyer and Seller

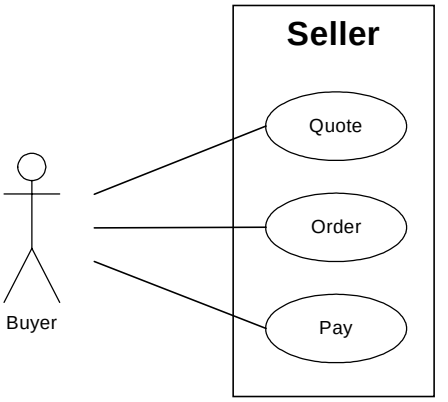
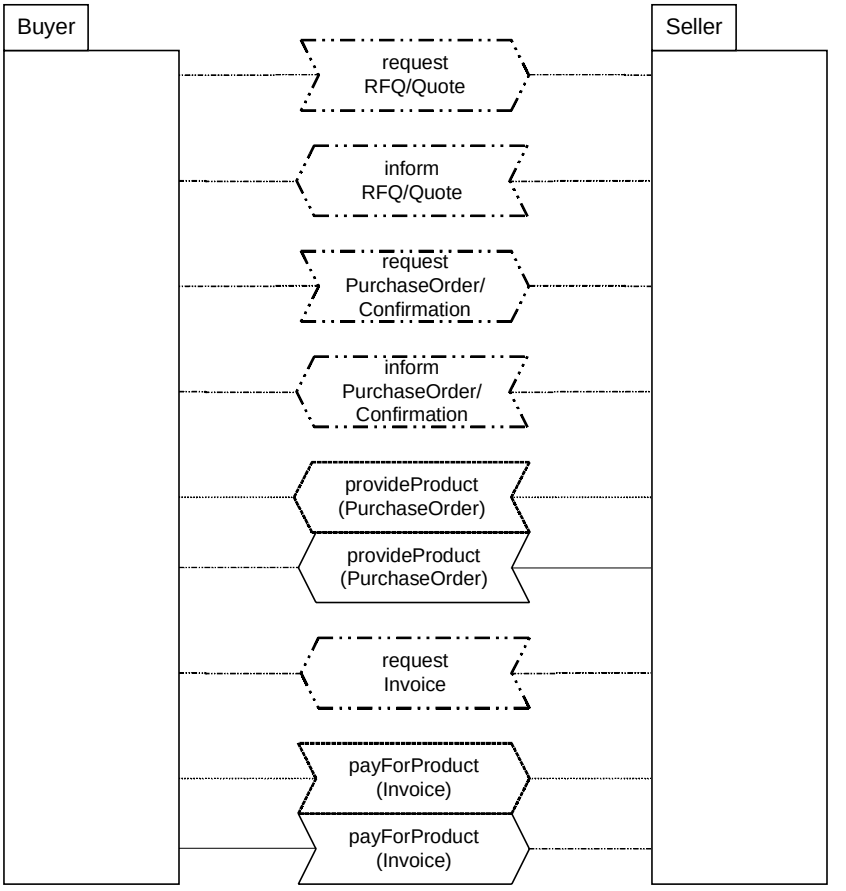


Figure 7. The interaction frame between agents of the types Buyer and Seller

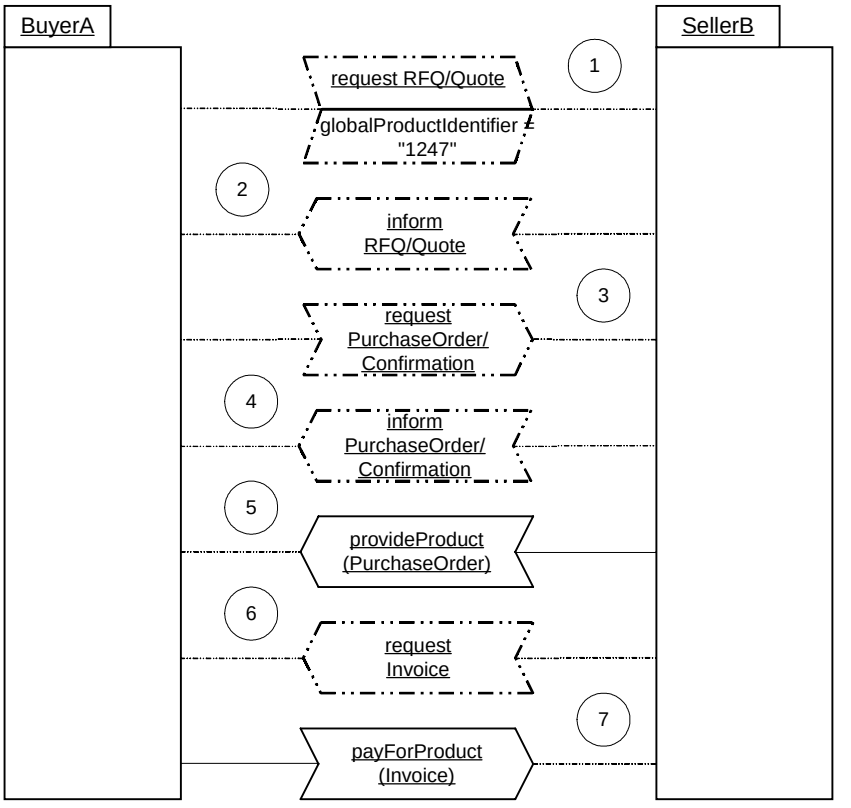


actors and the subject (system) and use cases (OMG, 2003b). The use cases represented in the domain interaction viewpoint are *business use cases* (Cockburn, 2001), because they summarize the types of interactions between a customer, which can be a human agent or an institutional agent, and the business, which is an institutional agent.

According to Wagner (2003a), an AOR *interaction-frame diagram* provides *a static picture* of the possible interactions between two (types of) agents without modelling any specific process instance. An interaction-frame diagram describes in more detail the types of interactions summarized by the corresponding use case. An interaction-frame diagram, like the one represented in Figure 7, consists of various types of communicative action events, non-communicative action events, commitments/claims (coupled with the corresponding types of action events), and non-action events. Agents of the problem domain share the entity types mentioned.

An *AOR interaction-sequence diagram*, like the one shown in Figure 8, depicts (some part of) a prototypical instance of an interaction process. An interaction

Figure 8. An interaction sequence between the agent instances BuyerA and SellerB



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.

Domain-Information Viewpoint

Representing the domain-information viewpoint (Column 2 in Table 1) for the focus organization(s) can be regarded as creating a *domain ontology* that provides a common framework of knowledge for the agents of the organization(s) and external agents connected to the organization(s). Each agent of the problem domain can see only a part of the ontology; that is, each agent views the ontology from a specific perspective.

The domain-information viewpoint is described with the help of one view—AOR Agent Diagrams.

In addition to describing agent types, an *AOR agent diagram*, like the one represented in Figure 5, describes object types of the problem domain, as well as their relationships to agent types and with each other. Each agent has beliefs about its internal agents, about its "private" objects, and about all external agents and shared objects that are related to it.

Domain-Behaviour Viewpoint

The domain-behaviour viewpoint (Column 3 in Table 1) addresses the modelling of an agent's functionality (what functions the agent has to perform), as well as of the agent's behaviour (when, how, and under what conditions work has to be done). In the models of the domain-behaviour viewpoint, a modeller may abstract away from particular internal agent types, like SoftwareAgent in Figure 5, and create models of behaviour for the corresponding institutional agent types, like Buyer and Seller in Figure 5. The domain-behaviour viewpoint is described with the help of three views:

1. AOR Interaction-Pattern Diagrams;
2. Goal-Based Use Case Models; and
3. AOR Activity Diagrams.

AOR interaction-pattern diagrams, like the one depicted in Figure 4, focus on general interaction patterns expressed by means of a set of reaction rules defining an interaction process type. In an interaction-pattern diagram, the

actions performed by one agent may be, at the same time, the events perceived by another agent. Figure 4 demonstrates that an interaction-pattern diagram can visualize the reaction chains that arise by one reaction triggering another. However, for adequate modelling of business processes, interaction-pattern diagrams are not sufficient because they do not allow modelling of action sequences. For this reason, we will introduce *activities* as a glue to connect the actions of an agent within a business process to each other.

Actor types (or *agent role types*) are always characterized by goals because, as noted by Kueng and Kawalek (1997), "human activity is inherently purposeful". In a business domain, a human or an institutional agent acting in the role of a "customer" has a goal of having something accomplished. To achieve its goal, the agent uses some service provided by another agent. An agent's autonomy implied by a *benevolence* relationship between the service provider and a service requester means that the service provider performs the service requested if it is able to do so, but the service provider also *has an option to refuse the service request*. Even though the agent requesting the service may not explicitly communicate its goals to the service provider agent, the latter always "internalizes" the whole or a part of the customer's goal in an attempt to provide the service. For example, assuming that a customer has a goal of renting a car, the goal of a car rental company is to provide the customer with a car, which is, of course, a subgoal of the company's higher level goal—to earn money through renting cars. The car rental company tries to achieve this higher level goal by "internalizing" as many customer goals as possible.

The "internalizations" of the goals of customers by service providers can be modelled in different ways. For example, in the ***i**** framework proposed by Yu (1995), a customer's goal can be captured by representing a *goal dependency* where a depender (i.e., a customer) depends on the dependee (i.e., a service provider) to bring about a certain state in the world. In the same framework, setting and achieving of the corresponding internal goal by the dependee can be modelled through a *means-ends link*, indicating a relationship between an internal goal to be achieved and a means for attaining it, which is usually performing a task (activity). As another example, in the KAOS framework described by Lamsweerde (2003), goals of agents are "operationalized" into specifications of services to be performed by an agent.

In our approach, we capture the goals of "customers" and their "internalizations" by service providers by employing *goal-based use case models*. Use cases as such were originally introduced by Jacobson (1992). Cockburn (1997a, 1997b) proposed an extended version of use cases that he calls "use cases with goals." He elaborated goal-based use cases in Cockburn (2001). While a graphical UML use-case diagram in the domain-interaction viewpoint summarizes types of interactions between an external actor and the focus agent (the "system"), a tabular goal-based use case models the behaviour of the focus agent.

Table 3. A goal-based use case for the business process type "Process the request for a quote"

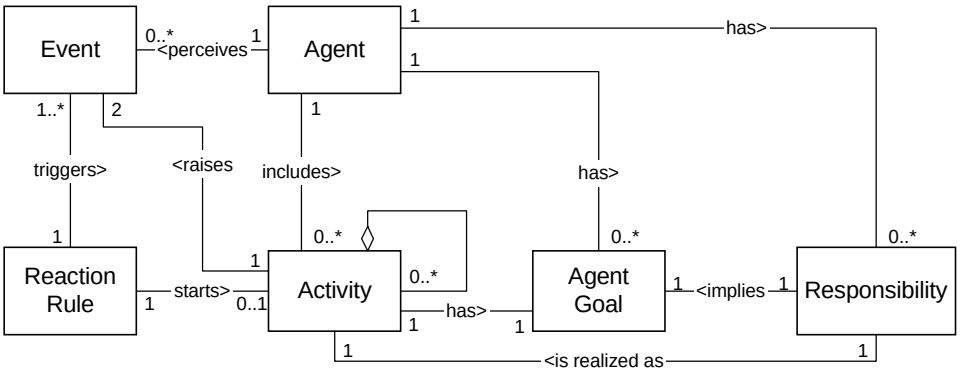
USE CASE 1	Process the request for a quote.	
Goal of the Primary Actor	To receive from the seller the quote.	
Goal of the Focus Actor	To provide the buyer with the quote.	
Scope & Level	Seller, primary task.	
Success End Condition	The buyer has received from the seller the quote.	
Primary Actor Secondary Actors	Buyer.	
Triggering event	A request for a quote by the buyer.	
DESCRIPTION	**Step**	**Action**
	1	Check and register the availability of the product items included in the request for a quote.
	2	Send the quote to the buyer.

A goal-based use case, such as the use case represented in Table 3, consists of a *primary actor,* the *system under discussion,* and a *secondary actor*. We will call the system under discussion the *actor in focus*. In the domain-behaviour viewpoint, where use cases describe the types of an organization's business processes, the actor in focus is the organization itself or an organization unit. According to Cockburn (2001), the *external primary actors* are the actors whose goals the organization is to satisfy. They include the company's customers and perhaps their suppliers. The external primary actors form a part of the company's stakeholders that includes the company shareholders, customers, vendors, and government regulatory agencies. A *secondary* or a *supporting actor* is an external actor that provides a service to the agent in focus, so that the latter could achieve its goals. In parallel with the identification of primary actors, the *triggering events* created by them to which the organization must respond should be identified (Cockburn, 2001).

Internal and external actors in goal-based use cases straightforwardly correspond to internal and external agents in AOR modelling. According to Cockburn (1997a), each actor has a set of *responsibilities* that have been assigned to it by external authorities. To carry out those responsibilities, it sets some goals. An agent's *responsibility* is realized as an *activity* that the agent performs in response to perceiving an event of the corresponding type. The relationship between an agent's responsibilities, goals, and activities is illustrated by the RAP/AOR metamodel fragment shown in Figure 9.

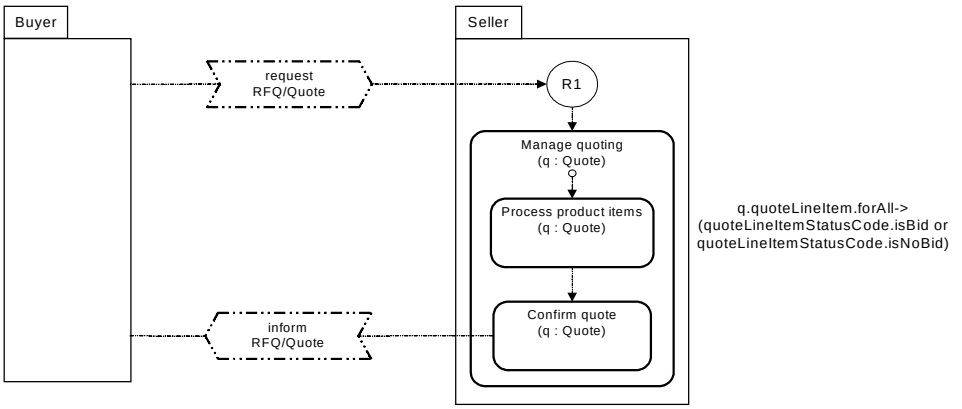
An ***activity*** is defined using workflow terminology as an uninterruptible amount of work that is performed in a non-zero span of time by an actor (Eshuis, Jansen,

Figure 9. The relationship between an agent's responsibilities, goals and activities



& Wieringa, 2002). Each activity belongs to some *activity type*. An *activity type* (or *task* in Yu [1995]), like "Manage quoting," is defined as a prototypical job function in an organization that specifies a particular way of doing something (Yu, 1995). It seems natural to allow specifying the start of an activity in the action part of a reaction rule. In other words, an instance of an activity type is created by means of a reaction rule in response to perceiving an event, which is also reflected by the RAP/AOR metamodel fragment represented in Figure 9. We define an *AOR activity diagram*, like the one shown in Figure 10, by making the definition provided in UML 2.0 (OMG, 2003b) slightly more precise as a

Figure 10. An incomplete model of the quoting business process type from the perspective of Seller



specification of parameterized behaviour that is expressed as a flow of execution via a sequencing of subordinate activities whose primitive elements are individual epistemic, physical, and communicative actions. As Figure 9 reflects, there are two activity border events (*start-of-activity* and *end-of-activity*) implicitly associated with the beginning and end of each activity. An activity border event starts either a subactivity or a subsequent activity or triggers a reaction rule. The *start-of-activity* event type is graphically represented by an empty circle with the outgoing arrow to the symbol of the subactivity type or internal reaction rule. The *end-of-activity* event type is visualized by drawing a triggering arrow from the activity-type symbol to either the symbol of the next activity type or to the symbol of the reaction rule triggered by an activity of the corresponding type.

The *goal* tied to an activity is defined as a condition or state of affairs in the world that the agent would like to achieve (Yu, 1995). When a goal-based use case is transformed into an activity diagram, the goal of the focus actor of the use case is attached to the diagram's outermost activity. The *pre-condition* of an activity is a *necessary* condition that must be true when the activity is started. Pre-conditions and goals may refer to status or intensional predicates of entity types. They are defined for activity types by means of OCL.

Viewpoints of Design

As explained in section "The AOR Modelling Language," an interacting system (or agent), as a subject in its domain, does not have an objective but rather a subjective view of the domain. This is reflected in RAP/AOR by a computational-design model, in which the internal (subjective) perspective of the system to be built is adopted, in contrast to the external (objective) perspective adopted in a conceptual-domain model. For instance, in the transformation of a domain-information model into an information- design model for a specific agent, the objective term *action event* is mapped onto the two indexical subjective terms, *action* (if performed by the agent under consideration) and *event* (if performed by other agents). Likewise, the objective term *message* is mapped onto the two subjective terms, *incoming message* and *outgoing message*. This mapping is also called *internalization* in RAP/AOR.

External models of the conceptual-domain modelling level are thus transformed into internal models of the level of platform-independent computational design. In particular, AOR agent diagrams are refined into more detailed agent diagrams and business use cases are turned to system use cases. Analogously, AOR interaction-frame diagrams are mapped to reaction-frame diagrams, AOR interaction-pattern diagrams to reaction-pattern diagrams, and AOR-activity diagrams to internal-activity diagrams.

The Role of Simulation in the RAP/AOR Methodology

We have shown in Wagner and Tulba (2003) that, with some minor extensions, AOR models can be used for a certain form of agent-based discrete event simulation called *Agent-Object-Relationship Simulation* (AORS). In RAP/AOR, we employ AORS for achieving more agility in the software engineering process by getting feedback from the execution of models before they are implemented in a target technology platform.

AORS allows animating and testing both AOR domain-interaction and behaviour models and AOR interaction and behaviour design models.

As has been shown in Taveter (2004a), simulation of behaviour models is facilitated by the executability of complete and incomplete AOR activity diagrams. An AORS system, for example, the implementation described in Luin, Tulba, and Wagner (2004), includes an environment simulator that is responsible for simulating exogenous events and the causality laws of the physical environment. Other actors of the problem domain can be simulated with various degrees of realism.

The Core Disciplines of the RAP/AOR Methodology

The seven core software engineering disciplines of RAP/AOR, based on those of RUP, are:

1. Domain modelling;
2. Requirements engineering;
3. Design;
4. Simulation;
5. Implementation;
6. Test; and
7. Deployment.

The first three disciplines listed above can be represented as the following task sequence divided between different viewpoints:

1. Domain-interaction viewpoint:
    1.1. Model agent types and instances of the problem domain and relationships between them by using an AOR-agent diagram.

1.2. Model possible interaction types between agents of the problem domain by UML use cases.

1.3. Refine possible interaction types between agents of the problem domain by AOR interaction-frame diagrams.

1.4. Model prototypical interaction processes of the problem domain by AOR interaction-sequence diagrams.

2. Domain-behaviour viewpoint:

2.1. Model the behaviours of agents of the problem domain by goal-based use cases.

2.2. Model the behaviours of agents of the problem domain by AOR reaction- pattern diagrams (optional).

2.3. Transform goal-based use cases into AOR-activity diagrams.

2.4. Refine AOR-activity diagrams by more detailed behavioural constructs.

2.5. Complement goal-based use cases by the elements corresponding to the behavioural constructs introduced.

3. Domain information viewpoint:

3.1. Model object types and instances of the problem domain and relationships between them by using an AOR-agent diagram.

4. Design interaction, behaviour, and information viewpoints:

4.1. Turn AOR interaction-frame diagrams into AOR reaction-frame diagrams.

4.2. When needed, complement AOR-reaction frame diagrams with user-interface design models and security models.

4.3. Turn AOR interaction-pattern diagrams into AOR reaction-pattern diagrams (optional).

4.4. Turn AOR-activity diagrams into internal AOR-activity diagrams.

4.5. Refine AOR-agent diagrams of the conceptual domain-modelling level into AOR-agent diagrams of the computational-design level.

In large development projects, there will be a team for each of the seven disciplines stated above.

The Phases of a RAP/AOR Development Project

Jacobson, et al. (1999) and Kruchten (1999) have pointed out that a software development project has two dimensions: the temporal sequence of project

Table 4. Project phases with predominant software engineering disciplines

Project Phase	*Predominant Disciplines*
Inception	domain modelling & requirements engineering
Elaboration	design & simulation
Construction	implementation & test
Transition	deployment

phases, and the composition of the overall development process of analysis, design, simulation, implementation, and test activities carried out iteratively in each phase. In RAP/AOR, we follow the distinction between four project phases: *inception*, *elaboration*, *construction*, and *transition*, each of which has one or two predominant software engineering disciplines according to Table 4. An important topic of our future work is aligning the phases of the RAP/AOR development project with the Software Process Engineering Metamodel Specification (SPEM, 2002) so that a phase would be a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria.

Case Study

Our case study is about the business process type of quoting in business-to-business electronic commerce that is based on the RosettaNet standard (http://www.rosettanet.org/). The RosettaNet's "Request Quote" Partner Interface Process® (PIP) enables a buyer to request a product quote from a provider and a provider to respond with a quote. The prices and product availability reflected in a quote may be influenced by an existing or potential relationship between a buyer and provider. We now discuss the conceptual modelling of this problem domain in terms of the RAP/AOR viewpoint modelling framework. We also briefly address the computational design of a business-to-business process automation system for the problem domain.

The Interaction Aspect

At the level of conceptual domain modelling (Row 1 in Table 1), the interaction aspect (Column 1 in Table 1) is captured by *conceptual AOR- interaction models* that are represented by using the following views: AOR-agent diagrams,

UML use-case diagrams, AOR interaction-frame diagrams, and AOR interaction-sequence diagrams.

At the level of computational design (Row 2 in Table 1), the interaction aspect is captured by AOR reaction-frame diagrams.

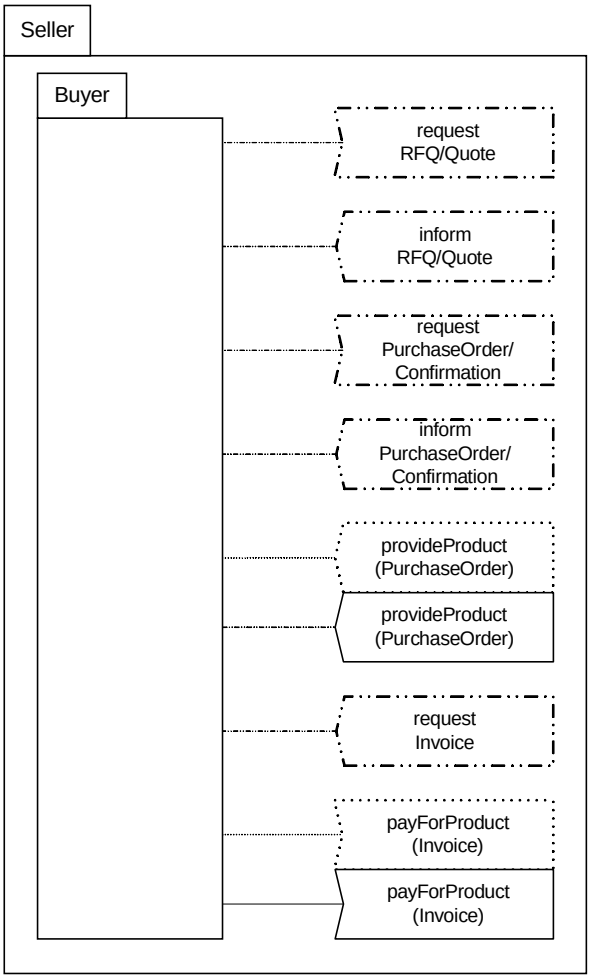
The agent diagram of Figure 5 represents the agent role types Buyer and Seller of the case study, with their respective internal agent types Clerk and SoftwareAgent. For simplicity, other relevant agent types like Bank have been omitted in the figure. There is the isBenevolentTo relationship between instances of the role types Seller and Buyer in Figure 5. This relationship typically appears between a service provider and requester.

A use-case diagram as depicted in Figure 6 summarizes the types of interactions between two agents of the type Buyer and Seller, which act as a service requester and provider, respectively. Interactions between a service requester and provider are further refined in the interaction-frame diagram in Figure 7, which covers the business process types of quoting, ordering, and invoicing between instances of the agent role types Buyer and Seller The interaction frame represented in Figure 7 contains types of communicative action events (messages) such as request PurchaseOrder/Confirmation, which is used for ordering, non-communicative action event types like provideProduct, and types of commitments/claims coupled with them. Each communicative action event type is prefixed by one of two *functions* of message types[5]: request, by which a sender requests the receiver to perform a communicative or physical action or both, and inform, which identifies a communicative action performed in reply to a request or independently.

Interaction-frame diagrams may be complemented by interaction-sequence diagrams, which depict prototypical instances of interaction processes. The interaction- sequence diagram in Figure 8 represents instances of the business process types of quoting, ordering, and invoicing between the agents *BuyerA* and *SellerB*.

An interaction-frame diagram of the conceptual domain-modelling level can be turned into a reaction-frame diagram of the computational-design level by the process of internalization, which was briefly described in an earlier section,"Viewpoints of Design." In that process, the external AOR model is transformed into an internal AOR model for the (type of) focus agent for which an agentified information system is to be developed (typically an organization or an organizational unit). For example, Figure 11 represents the reaction frame created for an instance of the agent role type Seller. This model has been obtained from the interaction frame shown in Figure 7.

Figure 11. The reaction frame for an agent of the type Seller



The Information Aspect

At the level of conceptual-domain modelling (Row 1 in Table 1), the information aspect (Column 2 in Table 1) is captured by *conceptual AOR information models*, which are represented by AOR agent diagrams.

The conceptual-information model of the problem domain of business-to-business electronic commerce is represented in the agent diagram in Figure 5. The figure shows that the object types PurchaseOrder/Confirmation, RFQ/Quote, and Invoice are *shared* between agents of the types Buyer and Seller, while the

object types ProductItemOfSeller and ProductItem are represented exclusively within agents of the types Buyer and Seller, respectively. It is important to emphasize here that a shared object type does not imply that all the agents whose types are connected by an association to it have the same beliefs about it, or, in other words, that there is a common extension of it shared by all agents. For example, different instances of the agent role type Buyer in Figure 5 hold different sets of instances of RFQ/Quote.

Instances of the object type Seller within an instance of Buyer represent information about the corresponding instances of the agent type Seller. An object of the type QuoteLineItem in Figure 5 satisfies one of the following *status predicates*: isBid, isNoBid, and isPending, while an object of the type ProductLineItem represented in the same figure is characterized by the status predicate isAccept, isReject, or isPending.

At the level of computational design (Row 2 in Table 1), the information aspect is captured by detailed AOR-agent diagrams that include, for example, the types of attributes and the identifiers for agent and object types.

The Behaviour Aspect

While the interaction aspect deals with the communication and interactions between the agents, the behaviour aspect (Column 3 in Table 1) addresses the agents' reactions to the communicative and non-communicative action events and non-action events perceived by them. At the level of conceptual modelling (Row 1 in Table 1), the behaviour aspect is captured by the *conceptual AOR behaviour models* that are represented by means of the following views: AOR interaction-pattern diagrams, goal-based use-case models and AOR-activity diagrams.

At the level of computational design (Row 2 in Table 1), the behaviour aspect is captured by AOR reaction-pattern diagrams and AOR internal-activity diagrams.

Figure 4 serves as an example of an interaction-pattern diagram specifying the reaction rule R1 for a Seller's behaviour in response to perceiving a communicative action event of the type request PurchaseOrder.

A goal-based use case can be triggered by an internal or external actor. For example, the use case "Issue a request for a quote" with the buyer in focus is triggered by the buyer's internal actor "clerk," while the use case "Process the request for a quote" presented as an example in Table 3 is triggered by receiving a request from a buyer for a quote. The use case ("Process the request for a quote") in Table 3 is modelled from the perspective of a buyer with the seller in focus (*scope*), which means that the goal of the use case is the so-called *user*

goal, the goal of the actor (i.e., a buyer) trying to get work (*primary task*) done (Cockburn, 1997a). The buyer is therefore the external primary actor of the use case. Since the goal of the primary actor is "internalized" by the actor in focus, unlike Cockburn, we also include the goal of the focus actor in a goal-based use case, as is reflected by Table 3.

As shown in Taveter (2004a), a goal-based use case can be straightforwardly turned into sequences and hierarchies of activity types whose instances are performed by an agent. For example, an activity diagram as shown in Figure 10 corresponds to the use case represented in Table 3. In the figure, an activity of the type "Manage quoting," which is visualized as a roundtangle, is started by reaction rule R1 in response to receiving a message containing a request for a quote. As shown in Figure 10, an activity of the type "Manage quoting" consists of sequential subactivities of the types "Process product items" and "Confirm quote."

For each activity type represented in Figure 10 can be defined the goal that its instances try to achieve. The goal defined for the outermost activity type, which is "Manage quoting" in Figure 10, corresponds to the focus actor's goal of the respective goal-based use case. For example, the goal of an activity of the type "Process product items" is represented informally as "For each product item included by the request for a quote is known whether it is to be bid or not." This goal can be formalized by means of OCL in terms of the input parameter declared for the corresponding activity type, as is represented in Figure 10. Input parameters defined for activity types represent the dataflow through the activities.

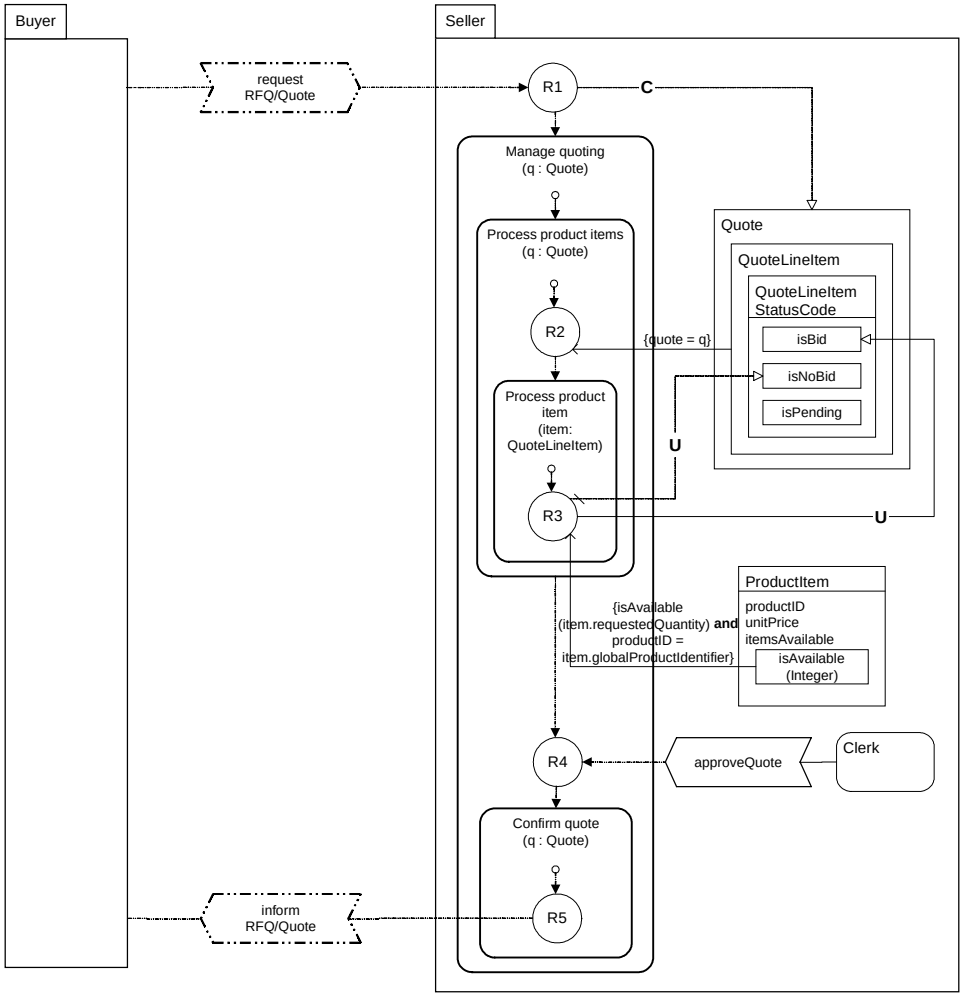
Next, activity diagrams obtained from goal-based use cases are elaborated by introducing into them *behavioural constructs* by means of reaction rules. Taveter (2004a) has shown that AORML extended by activity modelling allows the representation of 16 out of 19 behavioural workflow patterns as defined in the benchmark proposal of Workflow Patterns (2003). The complete behavioural model of the quoting business process type from the perspective of the agent role type Seller is represented in Figure 12. As the figure shows, the behavioural model shown in Figure 10 has been complemented by the behavioural constructs of the types "For-each loop" and "Starting an activity by means of two (or more) events." In addition, elementary epistemic, physical, and communicative actions that make up the activity types "Process product item" and "Confirm quote" have been specified.

According to the behavioural construct of the type "For-each loop" represented in Figure 12, upon the start of an activity of the type "Process product items," its subactivity of the type "Process product item" is performed for each instance of the object type QuoteLineItem for which the pre-condition quote = q evaluates to true. The pre-condition limits the set of QuoteLineItems for which the subactivity

is performed to those belonging to the instance of Quote that is identified by the value of the input parameter q. When all subactivities of the type "Process product item" have ended, the enclosing activity of the type "Process product items" also ends.

The subactivity "Process product item" in Figure 12 checks the availability of the given product item that is specified by the input parameter item of the type QuoteLineItem. If the product item corresponding to the instance of QuoteLineItem is available in the quantity requested, the status of the QuoteLineItem is set to isBid. In the opposite case, the status of the QuoteLineItem is changed to isNoBid. In both cases, the attributes of the QuoteLineItem are updated accordingly.

Figure 12. The complete behaviour model of the quoting business process type from the perspective of Seller



The behavioural construct of the type "Starting an activity by means of two (or more) events" represented in Figure 12 specifies that, upon the end of an activity of the type "Process product items," if the agent perceives an approval of the quote by an internal human agent of the type Clerk, an activity of the type "Confirm quote" is started. An activity of the given type is thus started only after events of *all* the specified types have occurred. In Figure 12, such events are the ending event of the previous activity and an approval of the quote by a human agent instance of the type Clerk.

Since goal-based use cases also serve to document business process types modelled by activity diagrams, in Table 5, the use case "Process the request for

Table 5. A goal-based use case for the business process type "Process the request for a quote"

USE CASE 2	Process the request for a quote.	
Goal of the Primary Actor	To receive from the seller the quote.	
Goal of the Focus Actor	To provide the buyer with the quote.	
Scope & Level	Seller, primary task.	
Success End Condition	The buyer has received from the seller the quote.	
Primary Actor Secondary Actors	Buyer.	
Triggering event	A request for a quote by the buyer.	
DESCRIPTION	**Step**	**Action**
	1	*For each product item included in the request for a quote*: process product item (**Use Case 3**).
	2	*The quote has been approved by the clerk*: send the quote to the buyer.

Table 6: The subfunction "Process product item"

USE CASE 3	Process product item.	
Goal of the Primary Actor		
Goal of the Focus Actor	To decide bidding of the product item.	
Scope & Level	Seller, subfunction.	
Pre-conditions		
Success End Condition	The bidding of the product item has been decided.	
Primary Actor Secondary Actors	Buyer.	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	*The product item is available in the quantity requested*: the product item is to be bid which is registered in the quote.
EXTENSIONS	**Step**	**Branching Action**
	1a	*The product item is not available in the quantity requested*: the product item is not to be bid which is registered in the quote.

a quote," which was originally represented in Table 3, has been complemented by the <*condition*> elements corresponding to the behavioural constructs introduced. Please note that a subfunction[6] of the primary task "Process the request for a quote" presented in Table 6 contains the *main (success) scenario* (Cockburn, 2001), in which the primary actor's goal is delivered and all the stakeholders' interests are satisfied, and one *extension scenario* (Cockburn, 2001) that starts by stating where it picks up in the main scenario and what conditions are different.

Simulation

Taveter (2004a) demonstrated that AORML extended by activities is the first agent-oriented modelling language where partially specified behaviour models by activity diagrams can be executed. For example, in our case study, an incomplete conceptual AOR behaviour model of the quoting business process type represented in Figure 10 can be executed, let alone the complete behaviour model represented in Figure 12. This facilitates iterative business modelling that is the state-of-the-practice. Using executable process models jointly with an Agent-Object-Relationship Simulation (AORS) system, as described in Wagner and Tulba (2003), permits the creation of powerful simulation environments. For this purpose, as well as for the creation of actual agentified information systems, the approach of transforming models into implementations described in the"Implementation and Tools" section can be employed.

Implementation and Tools

The RAP/AOR methodology complies with the principles of the MDA framework of OMG (http://www.omg.org/mda/). According to the overview of MDA provided in Klasse Objecten (http://www.klasse.nl/english/mda/mda-introduction.html), the MDA process defines three steps:

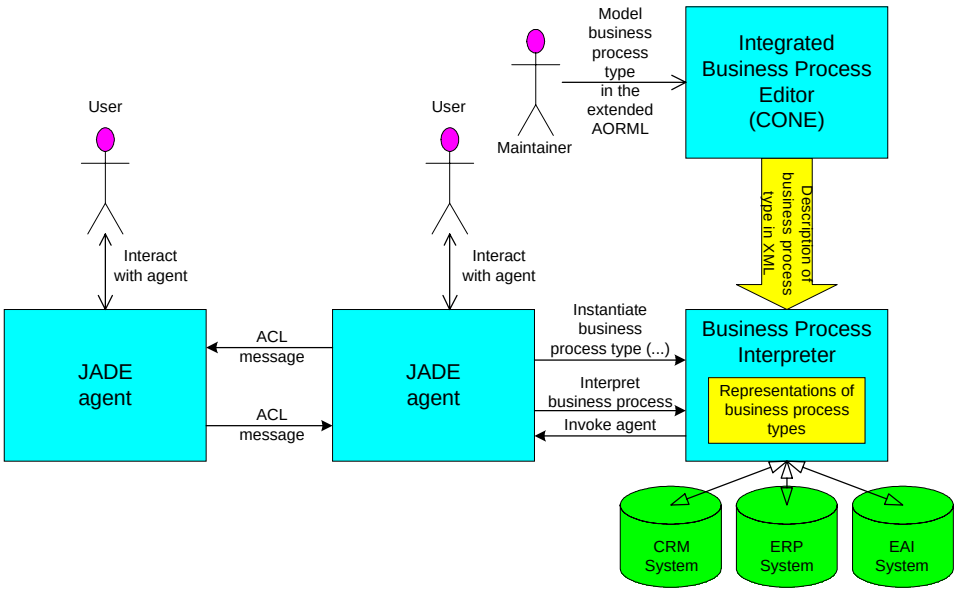
1. First, a model at a high level of abstraction that is independent of any implementation technology is built. This is called a Platform Independent Model (PIM). In the RAP/AOR methodology, the models listed in Row 2 of Table 1 form the PIM.
2. Next, the PIM is transformed into one or more Platform Specific Models (PSM). A PSM is tailored to specify the PIM in terms of the implementation constructs that are available in one specific implementation technology.

3. The final step is to transform a PSM to code. Because a PSM fits its technology very closely, this transformation is rather trivial. The complex step is the one in which a PIM is transformed into a PSM.

In principle, the transformation from a PIM into a PSM can be performed for each software system to be built (e.g., an agentified information system) separately. However, in the RAP/AOR methodology, we advocate an approach where executable AOR models are transformed into equivalent XML-based representations that are then interpreted and executed by software agents. In the case study of automating business-to-business processes, which we described in our case study, this is crucial because new business process types emerge and old ones frequently change, as a result of which specifications of business process automation systems are in constant flux.

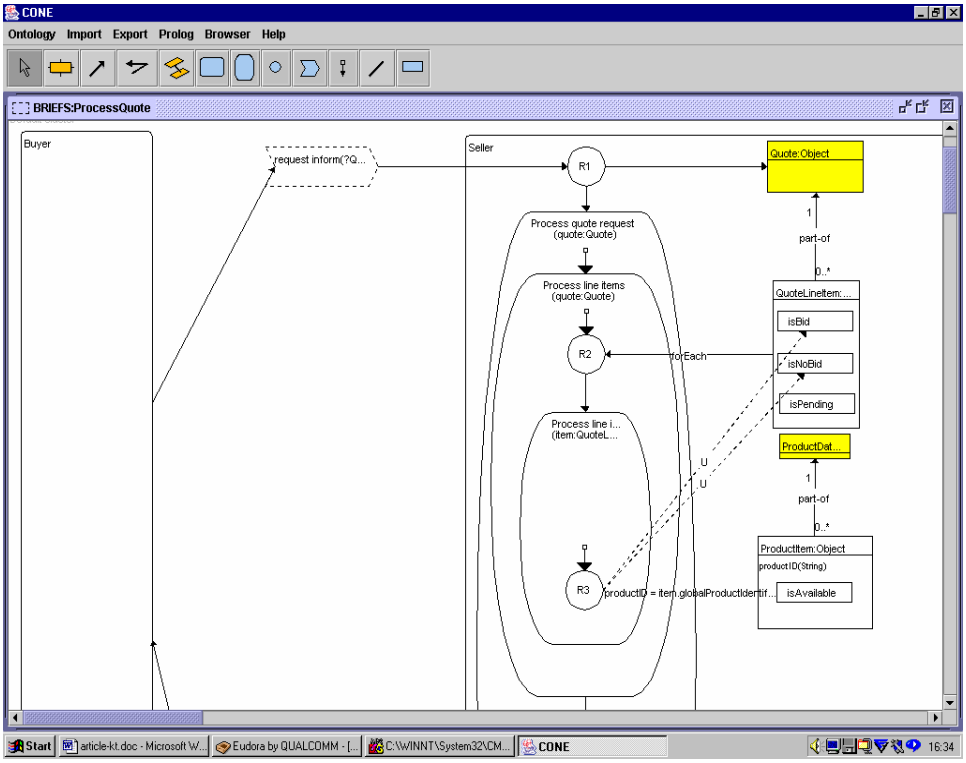
In order to facilitate generation of XML-based representations of business process models, we have developed the corresponding XML Schema (http://www.w3.org/XML/Schema) whose instances describe business process types in a machine-interpretable way. By using the schema, it is possible to represent business process types from different perspectives. For example, the models of the quoting business process type created in our case study are transformed into two XML-based representations that describe the quoting business process type from the perspectives of the agent role types Seller and Buyer.

Figure 13. The business process automation system



In the prototype application represented in Figure 13, inter-organizational business process types are described in the (extended by activities) AORML by means of the Integrated Business Process Editor. The latter has been developed as an extension to the COnceptual Network (CONE) Ontology Editor of VTT Information Technology. Figure 14 shows a snapshot of the model of the RosettaNet-based quoting business process type that has been created by using the editor from the perspective of Seller. Please note that since the older version of AORML was originally defined on the basis of UML 1.5 (OMG, 2003a), Figure 14 uses the old graphical notations for agents and activities in AORML. Note also that the models represented in Figure 14 are *external* AOR models – the process of internalization briefly described in section, "Viewpoints of Design," is performed implicitly when *subjective* XML-representations are generated from an *objective* model of a business process type. In other words, due to the implicit internalization, CIM serves as the starting point instead of PIM.

Figure 14. The quoting business process type from the perspective of Seller modelled by the Integrated Business Process Editor



The Integrated Business Process Editor allows graphical descriptions of business process types expressed in AORML to be transformed into their representations in XML. Even though the XML-based representations of a business process type are generated automatically, as a rule, some manual tailoring of process-specific interfaces must be performed. This occurs between the Business Process Interpreter and enterprise systems of the company. Enterprise systems can include the Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), and Enterprise Application Integration (EAI) systems shown in Figure 13. After the generation and tailoring have been accomplished, a business process type is ready to be interpreted by the Business Process Interpreter, which works in cooperation with the software agent representing the corresponding party. The latter has been implemented using the Java Agent DEvelopment Framework (JADE) (http://jade.cselt.it/) agent platform. Agents communicate with each other using messages in the Agent Communication Language (ACL) defined by FIPA (http://www.fipa.org/), which is based on speech acts (Austin, 1962). As Figure 13 illustrates, an agent representing a party first invokes the Business Process Interpreter to read the description of the business process type, as requested by the agent's human user, and to create its internal representation of the business process type. Thereafter, when the agent receives a message or "perceives" an input by a human user through the Graphical User Interface (GUI), the agent invokes the Business Process Interpreter to act according to the process type description. When the Business Process Interpreter acts, it, in turn, invokes the JADE agent and displays messages through the agent's GUI.

Strength and Weakness of the RAP/AOR Methodology

At present, it is difficult to evaluate the RAP/AOR methodology, mainly because some of its components—in particular, certain model transformations and related tools—have not yet been fully developed. Consequently, a lack of tool support and field testing reports is a weakness of the proposed methodology at the time of writing this chapter. On the other hand, the experience gained in the case study discussed and in implementing the tool support discussed in "Implementation and Tools" section have been encouraging and lead us to believe that RAP/AOR is a natural extension of RUP+UML offering higher level modelling constructs.

Not supporting the design of proactive behaviour may be seen either as a weakness or as a strength. It is certainly a weakness from an AI point of view; however, we consider it to be a strength since the agents we want to design as components of large-scale cross-enterprise distributed information systems do not need proactive behaviour, nor do they need any other form of sophisticated human-like intelligence. Rather, such agents typically have a complicated structure of beliefs and commitments and a reactive behaviour based on them, both of which can be captured by the RAP/AOR methodology.

Two particular strengths of the proposed methodology are its ontological foundation and its use of simulation for achieving more agility. Another strength is the possibility to represent the models of the interaction, information, and behaviour viewpoint aspects in just one integrated diagram. This overcomes the *model multiplicity problem* (Peleg & Dori, 2000), which is that, to understand the system being studied and the way it operates and changes over time, the reader must concurrently refer to various models.

An important open issue for RAP/AOR is the potential for an operational commitment concept for designing and controlling business-to-business interactions. The basic assumption of AOR modelling that, in addition to beliefs and perceptions, commitments are the third mental state component that is important for understanding and designing agent-to-agent interactions, has not yet been validated in novel technologies and practical applications.

Conclusion

In this chapter, we have introduced the RAP/AOR methodology for agent-oriented information systems engineering. Unlike many other agent-oriented methodologies, RAP/AOR is not intended to be used in the development of AI agent systems; rather, it targets the development of large-scale distributed and cross-enterprise business information systems that may include or even consist of software agents. RAP/AOR is an agent-oriented extension of RUP+UML by adding the mental state structure modelling constructs of agents, events, actions, commitments, and claims, and by adding the behaviour modelling constructs of reaction rules and activities. Additionally, these modelling constructs fit well with MDA. In particular, they yield a higher level PIM language that allows a more direct mapping of CIM elements to PIM elements.

References

Austin, J. (1962). *How to do thing with words*. Urmson Editor, Oxford, UK: Clarenson Press.

Beck, K. (1999). *Extreme programming explained: Embrace change*. Indianapolis, IN: Addison-Wesley Professional.

Bernon, C., Glaizes, M-P., Peyruqueou, S., & Picard, G. (2002). ADELFE, a methodology for adaptive multi-agent systems engineering. In P. Petta, R. Tolksdorf, & F. Zambonelli, F. (Eds.), *Engineering Societies in the Agents World III, Third International Workshop* (ESAW 2002), Madrid, Spain, September 16-17. Revised Papers (pp. 156-169). LNAI 2577. Berlin: Springer-Verlag.

BPML (2002). Business Process Modeling Language 1.0 and Business Process Modeling Notation 0.9. (2002). Retrieved September 4, 2004, from *http://www.bpmi.org*.

Capera, D., Georgé, J-P., Gleizes, M-P., & Glize, P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the 1st International Workshop on Theory and Practice of Open Computational Systems* (TAPOCS03@WETICE 2003), Linz, Austria, June.

Cockburn, A. (1997a). Goals and use cases. *Journal of Object-Oriented Programming*, September.

Cockburn, A. (1997b). Using goal-based use cases. *Journal of Object-Oriented Programming*, November/December.

Cockburn, A. (2001). *Writing effective use cases*. Reading, MA: Addison-Wesley.

Eshuis, R., Jansen, D. N., & Wieringa, R. J. (2002). Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Engineering Journal, 7*(4), 243-263.

Evans, R., Kearney, P., Stark, J., Caire, G., Garijo, F. J., Gomez Sanz, J. J., Pavon, J., Leal, F., Chainho, P., & Massonet, P. (2001). *MESSAGE: Methodology for Engineering Systems of Software Agents*. EURESCOM Technical Information, 2001. Retrieved August 31, 2004, from *http://www.eurescom.de/~pub-deliverables/P900-series/P907/TI1/p907ti1.pdf*

Fowler, M. (2003). The new methodology. Retrieved September 4, 2004, from *http://martinfowler.com/articles/newMethodology.html#N400315*

Guizzardi, G. & Wagner, G. (2004). On the ontological foundations of agent concepts. In *Proceedings of International Workshop on Agent-Ori-*

ented Information Systems (AOIS-2004) at CAiSE 2004, Riga, Latvia, June.

Jacobson, I. (1992). *Object-oriented software engineering: A use-case driven approach*. Reading, MA: Addison-Wesley.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *Unified software development process.* Reading, MA: Addison-Wesley.

Kruchten, P. (1999). *Rational unified process – An introduction*. Reading, MA: Addison-Wesley.

Kueng, P. & Kawalek, P. (1997). Goal-based business process models: Creation and evaluation. *Business Process Management Journal, 3*(1), 17-38.

Lamsweerde, A. van. (2003). From system goals to software architecture. In M. Bernardo & P. Inverardi (Eds.), *Formal methods for software architectures*, LNCS, 2804 (pp. 25-43). Berlin: Springer-Verlag.

Luin, J., van, Tulba, F., & Wagner, G. (2004). Remodelling the beer game as an agent-object-relationship simulation. In *Proceedings of Workshop 2003: Agent-Based Simulation 5*, Lisbon, Portugal, 3-5 May. SCS European Publishing House.

OMG (2003a). Unified modeling language specification. March 2003, Version 1.5. Retrieved September 28, 2004, from *http://www.omg.org/cgi-bin/doc?formal/03-03-01*

OMG (2003b). Unified modeling language: Superstructure. Version 2.0, August 2003. Retrieved September 25, 2004, from *http://www.omg.org/cgi-bin/doc?ptc/2003-08-02*

Peleg, M. & Dori, D. (2000). The model multiplicity problem: Experimenting with real-time specification methods. *IEEE Transactions on Software Engineering, 26*(8).

Putman, J. & Boehm, B. (2001). *Architecting with RM-ODP*. Upper Saddle River, NJ: Prentice Hall.

Rao, A. S. & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, & E. Sandewall. (Eds.), *Proceedings of Knowledge Representation 91* (KR-91). San Mateo, CA: Morgan Kaufmann.

Searle, J. R. (1995). *The construction of social reality*. New York: Free Press.

Sowa, J. F. & Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Systems Journal, 31*(3).

SPEM: Software Process Engineering Metamodel Specification. November 2002, Version 1.0. Retrieved October 20, 2004, from *http://www.omg.org/docs/formal/02-11-14.pdf*

Taveter, K. (2004a). *A multi-perspective methodology for agent-oriented business modelling and simulation*. Ph.D. Thesis, Tallinn University of Technology, Estonia. (ISBN 9985-59-439-8).

Taveter, K. (2004b). From business process modelling to business process automation. In J. Cordeiro & J. Filipe (Eds.), *Computer-supported activity coordination – Proceedings of the 1st International Workshop on Computer Supported Activity Coordination* (CSAC 2004). In conjunction with ICEIS 2004, Porto, Portugal, April (pp. 198-210). Setubal , Portugal: INSTICC Press.

Wagner, G. (2003a). The agent-object-relationship meta-model: Towards a unified view of state and behavior. *Information Systems, 28*(5), 475-504. Available online at *http://aor.rezearch.info/*

Wagner, G. (2003b). A UML profile for external AOR models. In F.Giunchiglia, J. Odell, & G. Weiss (Eds.), *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002. Bologna, Italy, July 15, Revised Papers and Invited Contributions*. LNCS, Vol. 2585. Berlin: Springer-Verlag.

Wagner, G. & Tulba, F. (2003). Agent-oriented modeling and agent-based simulation. In P. Giorgini & B. Henderson-Sellers, (Eds.), *Conceptual modeling for ‚novel application domains*. LNCS, Vol. 2814. Berlin: Springer-Verlag.

Workflow Patterns. (2003). Retrieved September 25, 2004, from *http://tmitwww.tm.tue.nl/research/patterns/*

W3C (2003). Web Services Architecture (WSA). W3C Working Draft, 8 August 2003. Retrieved September 29, 2004, from *http://www.w3.org/TR/2003/WD-ws-arch-20030808/*

W3C (2004). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft, 3 August 2004. Retrieved September 27, 2004, from *http://www.w3.org/TR/2004/WD-wsdl20-20040803/*

Yu, E. (1995). *Modeling strategic relationships for process reengineering*. Ph.D. Thesis, Department of Computer Science, University of Toronto, Canada.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini & M. Wooldridge (Eds.), *Agent-oriented software engineering*. LNCS 1957 (pp.127-141). Berlin: Springer-Verlag.

Endnotes

1 Strictly speaking, the RAP defines a *process type family* whose members are *process types* that can be instantiated by different *process individuals*. It is common practice, though, to use the term "process" ambiguously both at the level of types and at the level of instances.

2 The RAP/AOR methodology has, for example, been used for creating a prototypical system for the automation of business-to-business processes described in Taveter (2004b).

3 A *role* can be understood as an "abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavor" (Yu, 1995), such as the role Seller.

4 We use the terms "actor" and "agent" as synonyms.

5 These functions are actually *speech acts* (Austin, 1962).

6 A *subfunction* is a use case that is below the main level of interest of the primary actor.