Agent-Based System Design for B2B Electronic Commerce

Anthony Karageorgos, Simon Thompson, and Nikolay Mehandjiev

ABSTRACT: Agent-based systems are increasingly used to support business-to-business (B2B) electronic commerce and other Internet-based transactions. The design complexity resulting from the multiple interconnected systems in these domains has to be managed in order to reduce costs and time to market.

This paper introduces the Role-Algebraic Multi-Agent System Design (RAMASD) approach. RAMASD utilizes role models as reusable system-building blocks and a role algebra to capture the basic relations of roles. A two-sorted algebra is used to define the role algebra's semantics. RAMASD reduces the complexity of designing agent-based B2B e-commerce systems by enabling designers to work at a high level of abstraction and by automatically allocating roles to agents according to applicable role models and design constraints.

A case study concerning a B2B electronic market for the automotive industry demonstrates the applicability of RAMASD. The advantages and disadvantages of the proposed approach are discussed, and comparisons with relevant work are made.

KEY WORDS AND PHRASES: Agent-oriented software engineering, agent organizations, agent-based e-business.

Agent technology has come to the forefront in the software industry because of the advantages that agents have in complex, distributed environments. It is increasingly used in Internet-based transactions, including electronic commerce and cross-organizational workflow management [7, 24].

Designing an agent-based B2B electronic commerce system is a complex process in which it is necessary to define both the structural relationships between agents and individual agent behavior. Many authors view agent-based systems as organized societies of individual computational entities and thus see designing an agent-based system as designing an *agent organization* (e.g., [14, 38, 43]).[1] Since the criteria affecting agent organization design decisions are numerous and highly dependent on factors that may change dynamically, there is no standard best organization for all situations [34, 38]. Furthermore, to reduce cost and time to market, agent system design decisions should be reusable while both functional and nonfunctional aspects are considered [6].

Existing approaches to agent-based system design fail to address design complexity in that they require the designer to make most of the decisions without the assistance of a software tool [29, 39, 43]. This can be a serious drawback when designing large and complex agent-based B2B e-commerce systems for the real world. Therefore, many authors argue that collective behavior as well as social and organizational abstractions should be considered as first-class design constructs, enabling the agent system designer to reason at a high level of abstraction (e.g., [22, 28]).

This paper proposes an approach that provides semiautomatic support for the high-level design of agent-based B2B electronic commerce systems. It uses role models as basic building blocks, and formalizes the rules and constraints

International Journal of Electronic Commerce / Fall 2002, Vol. 7, No. 1, pp. 59–90.

1086-4415/2002 $9.50 + 0.00.

governing their combinations. This enables semiautomatic tool support for the agent organization designer. The approach has been incorporated in an experimental version of the Zeus agent-building toolkit [27].

Designing Agent-Based Systems

Early research prototypes of agent-based systems were built in an ad hoc manner. The need to engineer agent systems solving real-world problems, however, has given rise to a number of systematic methodologies for agent-oriented analysis and design, such as MESSAGE, Gaia, and SODA [9, 28, 43]. All of these methodologies involve analytic and design submodels that emphasize particular aspects of analysis and design. Organizational settings are either specified explicitly in an organizational model (e.g., [9]) or are defined implicitly from the functionality that agents are assigned (e.g., [22]). Nonfunctional requirements are not explicitly considered in existing agent-based system engineering methodologies, the only exception being TROPOS, where specific steps in the agent system engineering methodology take nonfunctional requirements into account [8].

Weaknesses of Agent System Design Methodologies

The existing approaches to agent-based system design could be further improved by:

- Introducing a practical way to construct agent system design models from analysis models.

  The existing approaches to agent-based system engineering typically involve a considerable number of analysis and design models. There are five analysis models and three design models in MESSAGE [9], three analysis models and four design models in SODA [34], and two analysis models and three design models in Gaia [46]. The main drawback of these approaches is that after a certain point the design decisions are left solely to the creativity and intuition of the designer. The steps involved in transforming analysis models to design models are not specified in a way that would enable an adequate degree of automation by a software tool.

- Explicitly considering nonfunctional requirements on design time.

  The existing agent-based system engineering approaches do not explicitly model nonfunctional aspects on design time, and when they consider them, they do so by adjusting the agent behavior on runtime. As a result, nonfunctional design decisions cannot be reused and runtime reorganization may result in significant resource consumption and system instability. The aim should be to achieve the best possible agent organization at design time. To achieve this, it is necessary to explicitly model and consider nonfunctional aspects before actually deploying a multi-agent system. This idea follows the spirit of

similar works that model and study the behavior of a multi-agent system before actual system deployment [30, 34].

- Reusing functional, nonfunctional, and organizational settings.

  The reuse of functional knowledge has long been an issue in software engineering, and it is mandatory in order to reduce cost and time to market of Internet-based B2B e-commerce systems [6]. The view regarding reuse of organizational settings was inspired by the concepts introduced by Zambonelli, Jennings, and Wooldridge [46]. Their work can be extended by classifying known organizational patterns and providing rigorous means for selecting them in a particular design context. If organizational patterns are to be of practical use in implementing large-scale, real-world agent applications, a way of easily integrating organizational with functional and nonfunctional design decisions is needed.

Background Concepts

Many modeling approaches use roles as basic building blocks. For example, organizational theory uses roles to represent positions and responsibilities in human organizations [34]. Roles are also used in software engineering [1]. They are particularly suitable for modeling the behavior of software agents (e.g., [22]). Agent roles are defined in a manner similar to organizational roles by referring to a position and a set of responsibilities in an organization [14]. To better represent agent concepts, the agent role definition includes additional characteristics like planning, co-ordination, and negotiation capabilities [22].

Existing role-based approaches to multi-agent system design stress the need to identify and characterize relations between roles [1, 22]. However, only a few approaches attempt to investigate the consequences of role relations for the design of multi-agent systems (e.g., [22]). This is partly because of the lack of formal foundations in role relationships. In this work, role relations that affect multi-agent system design are identified and formalized in an algebraic specification model. Role identification is based on organizational principles and, in particular, on *role theory* [4].

The essence of role theory is that persons are appointed to roles in an organization that are representations of concrete behavior. This behavior is characterized by *authorities* describing things that can be done and *responsibilities* describing things that must be done. For example, the job descriptions of directors, help-desk staff, developers, and test engineers all specify their responsibilities in the organization. Their rights and duties in their respective departments, projects, or groups are further determined by organizational goals, policies, and procedures.

Role theory emphasizes that relations between roles may differ. For example, since an examiner cannot be a candidate at the same time, assigning these two roles to one person at the same time would result in an inconsistency. Role relations can be complex. A university staff member who is also a private consultant, for example, may have conflicting interests. Assigning these roles to the same person is possible but would require appropriate mechanisms to resolve the conflicting behavior.

Main Ideas Behind the Proposed Approach

This paper describes part of a work that attempts to extract role relations from human organizations with an eye to using them to specify agent behavior. The reason for searching for role relations in the human organizations domain was that agent research has traditionally aimed to develop agents that mimic human behavior and can be organized in a manner similar to humans. As roles have been extensively used in human organizations (e.g., [45]), it was natural to examine human organizations to identify role relations. The decision to use abstractions from human organizations to model software agent behavior is in line with the latest trend in software engineering, where there is a move from languages and representation formalisms whose conceptual basis is determined by the underlying machine architecture to languages whose key abstractions are rooted in the problem domain [18]. For this reason, the agent-system design approach proposed in this paper is well suited for designing multi-agent systems to support human activity systems, such as B2B electronic commerce systems.

An issue of major concern in the design of agent-based B2B e-commerce systems is the modeling and consideration of nonfunctional requirements [6]. Treatments of nonfunctional requirements can be classified as product-oriented or process-oriented [10]. *Process-oriented* approaches develop techniques for justifying decisions during the software development process, whereas *product-oriented* approaches deal with nonfunctional issues from the evaluation point of view. Software products may be examined to check whether they fall within their constraints of nonfunctionality. This paper combines elements from both approaches. Nonfunctional aspects can be modeled by appropriate role models and taken into account throughout the design process. In addition, they can be quantitatively modeled as constraints on the characteristics of functional role models and can subsequently be used to drive the allocation of roles to agents.

Modeling with Roles

Roles can be used as building blocks for an approach to agent-based system design addressing the weaknesses described above. This is achieved by extending existing role definitions to allow for the modeling of nonfunctional requirements and by introducing a systematic role-model transformation technique enabling semiautomation of the design process.

Role Characteristics

Following Kendall, a role is defined as a *position* and a set of *characteristics* [22]. Each characteristic includes a set of *attributes*. Countable attributes may further take a range of values. More specifically, a role is considered capable of carrying out certain *tasks* and can aim to achieve various *responsibilities* or *goals*. Roles normally need to interact with other roles, which are their *collaborators*. Interaction takes place by exchanging messages according to *interaction*

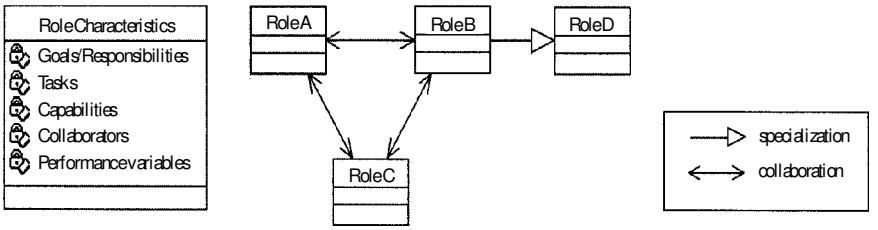

Figure 1. Schematic Representation of a Role Model

protocols. When an entity realizes the behavior represented by a role, it is said to *play* that role.

Roles can be extended to create specialized roles by a process called role *specialization* or *refinement* [1, 22]. Specialized roles represent additional behavior on top of the original role behavior, and thus resemble inheritance in object-oriented systems.

Role Composition

The task of merging several roles into a single composite role is called *role composition*. Role composition occurs when roles are allocated to agents. In role composition, roles may semantically constrain one other. For example, two roles may constrain each other in such a way that a single agent cannot play both roles at the same time. The way that role characteristics and their attributes are merged may be bound to constraints of various kinds. For example, the resource capacity required by the composite role resulting from the merging of two roles may be less than the sum of the capacities required by the two individual roles. In this paper, the constraints arising from relations among roles are referred to as *compositional constraints*. A formal model of the basic relations among roles, the role algebra, will be described further on.

A collection of roles and their interactions constitutes a *role model* (*see Figure 1*). A role model represents the collective behavior required to carry out some activity in the system.[2] An agent application normally consists of more than one activity and thus will involve more than one role model. Role models that occur frequently in some application domain are called *role-interaction patterns*. Role models can be used to represent reoccurring complex behavior based on multiple points of interaction and, therefore, are first-class design constructs—that is to say, they are entities that can be instantiated and given identity. Role models can be used to describe both functional and nonfunctional aspects as well as organizational settings. An agent system designer should be able to reuse role-interaction patterns and specify new role models as required. Therefore, the problem of designing an agent-based system refers to selecting and instantiating suitable role models.

Role Model Types

Role models can be used to describe various types of organizational, functional, and nonfunctional behavior. By using compositional constraints, one

can specify how different types of behavior will be merged and allocated to agents.

The following types of role models can be identified:

- *Functional role models* describe behavior specific to the application domain. For example, the collective behavior that carries out negotiation in a B2B e-commerce context can be described by a functional role model.
- *Nonfunctional role models* model behavior that implements nonfunctional aspects of the application. For example, to increase the security of business-to-business transactions only registered partners should be able to access the pricing information, and any transactions should be carried out using a secure communications protocol. This could be modeled by representing that "nonfunctional" behavior by two roles: *Registered_Partner* and *Secure_Protocol _Trader,* and by requiring that agents play those roles in order to interact with other agents in the agent-based system.
- *Organizational role models* specify organizational patterns (i.e., reusable organizational settings one would like to impose on the agent system) and agent behavior. For example, an agent requiring assistance in some task may ask its *peer agents* (i.e., agents at the same level in the organizational hierarchy [27]) if they are able to provide it. Organizational role models can also be used to impose organizational rules and to introduce social relations among agents in a multi-agent system [28, 46].

Modeling Nonfunctional Requirements

In order to represent realistic behavior in an application domain, roles need to model issues relevant to nonfunctional requirements in that domain. In the approach proposed in this paper, this can be done by using appropriate role models to explicitly represent nonfunctional aspects (i.e., a process-oriented approach) and by modeling nonfunctional requirements as constraints on role characteristics (i.e., a product-oriented approach).

Representing nonfunctional aspects by explicit modeling constructs is an approach followed by many business systems engineering methodologies (e.g., [5, 8]). The underlying principle is the *separation of concerns* [22, 28]. Role models can be used to represent nonfunctional aspects. For example, the security of an agent-based B2B e-commerce system can be ensured by requiring that the agents carrying out the important financial transactions (e.g., payments) play the *SecurityCompliant* role, where *SecurityCompliant* is a role representing the necessary behavior to achieve security compliance.

Using explicit role models to represent nonfunctional aspects is not always the best solution, however. This is obvious when the designer would like to take quantitative nonfunctional constraints into account. For example, if the response time of a transaction should be less than 1 second, it is neither intuitive nor practical to represent this fact by a separate role to be played by the

agent responsible for the transactions. Therefore, the role definition given above is extended to include *performance variables*—parameters whose value defines the runtime behavior represented by a role. For example, if the behavior of a role requires using some resource like memory, the resource capacity can be modeled by a performance variable. Performance variables can also be defined at an agent level. In that case, their value is a function of the values of the respective performance variables of all the roles the agent is capable of playing. This makes it possible to apply design heuristics by imposing constraints on the values of the agent performance variables that must be observed when allocating roles to agents, as illustrated in the case study below.

A Role Algebra for Multi-Agent System Design

Based on role theory and on case studies of human activity systems, six basic role relations have been identified [4, 20]. They will now be formally defined in a model referred to as *role algebra*. Using relations from the role algebra, we can specify the constraints driving the assignment of roles to agents, and in consequence the agent organization design process can be partially automated.

The aim in designing the role algebra was to keep it as simple as possible so that it could be used pragmatically in real-world agent applications. This section will formally define the role relations and then use intuitive examples to informally describe their meaning. Subsequently, a two-sorted algebra is used to give a formal description of the semantics of the role relations.

Relations in the Role Algebra

Let R be a set of roles. For any $r_1, r_2 \in R$, the following binary relationships may hold:

1. **Equals (eq)**. This means that r_1 and r_2 describe exactly the same behavior. For example, the terms *Advisor* and *Supervisor* can be used to refer to people supervising Ph.D. students. When two roles are equal, an agent playing the first role also plays the second at the same time. The relation $Equals \subseteq R \times R$ is an equivalence relation, since it is reflexive, symmetric, and transitive:

$$\forall r : R \ \ (r \text{ eq } r)$$

$$\forall (r_1, r_2) : R \times R \ (r_1 \text{ eq } r_2 \Rightarrow r_2 \text{ eq } r_1)$$

$$\forall (r_1, r_2, r_3) : R \times R \times R \ ((r_1 \text{ eq } r_2) \wedge (r_2 \text{ eq } r_3) \Rightarrow (r_1 \text{ eq } r_3))$$

2. **Excludes (not)**. This means that r_1 and r_2 cannot be assigned to the same agent simultaneously. For example, in a conference-reviewing agent system, an agent should not be playing the roles of paper author and paper reviewer at the same time. Furthermore, a role cannot exclude itself, for if it could, then no agent would ever play it. Therefore, the relation $Excludes \subseteq R \times R$ is antireflexive and symmetric:

$$\forall r : R \ (\neg(r \text{ not } r))$$

$$\forall (r_1, r_2) : R \times R \ (r_1 \text{ not } r_2 \Rightarrow r_2 \text{ not } r_1)$$

3. **Contains (in)**. This means that a role is a subcase/specialization of another role. Therefore, the behavior represented by the first role completely includes the behavior of the second role. For example, a role representing *Manager* behavior completely contains the behavior of the *Employee* role. When two roles such that the first contains the second are composed, the resulting role is the first role. Therefore, the relation *Contains* $\subseteq R \times R$ is reflexive and transitive:

$$\forall r : R \ (r \text{ in } r)$$

$$\forall (r_1, r_2, r_3) : R \times R \times R \ ((r_1 \text{ in } r_2) \wedge (r_2 \text{ in } r_3) \Rightarrow (r_1 \text{ in } r_3))$$

4. **Requires (and)**. The *Requires* relation can be used to show that when an agent is assigned a particular role, then it must also be assigned some other specific role as well. This is especially applicable in cases where agents need to conform to general rules or play organizational roles. For example, in a university application context, in order for an agent to be a *Library_Borrower*, it must be a *University_Member* as well. Although the behavior of a *Library_Borrower* could be modeled as part of the behavior of a *University_Member*, this would not be convenient, since this behavior could not be reused in other application domains where being a *Library_Borrower* is possible for everyone. Furthermore, each role requires itself. Intuitively, the roles that some role r requires are also required by all other roles that require r. Therefore, the relation *Requires* $\subseteq R \times R$ is reflexive and transitive:

$$\forall r : R \ (r \text{ and } r)$$

$$\forall (r_1, r_2, r_3) : R \times R \times R \ ((r_1 \text{ and } r_2) \wedge (r_2 \text{ and } r_3) \Rightarrow (r_1 \text{ and } r_3))$$

5. **Addswith (add)**. The *Addswith* relation can be used to express the notion that the behaviors represented by two roles do not interfere in any way. For example, since the *Student* and *Football_Player* roles describe nonexcluding and nonoverlapping behaviors, they can be assigned to the same agent without any problems. The relation *Addswith* $\subseteq R \times R$ is reflexive and symmetric:

$$\forall r : R \ (\neg(r \text{ add } r))$$

$$\forall (r_1, r_2) : R \times R \ ((r_1 \text{ add } r_2) \Rightarrow (r_2 \text{ add } r_1))$$

6. **Mergeswith (merge)**. The *Mergeswith* relation can be used to express the idea that the behaviors of two roles overlap to some extent or that different behavior occurs when two roles are put together. For example, a *Student* can also be a *Staff_Member.* This refers to cases

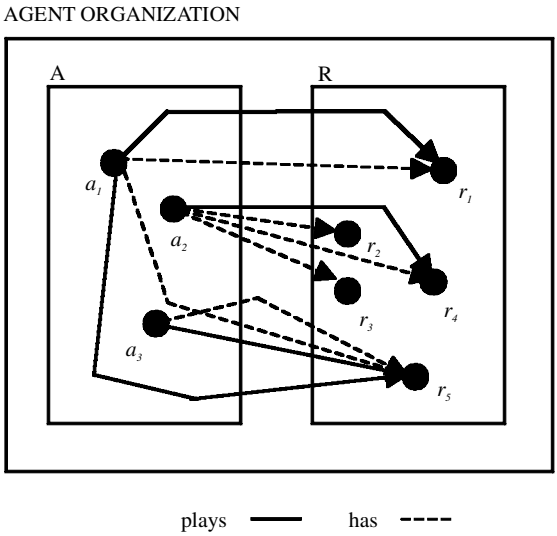

Figure 2. Semantics of Role Relations

where Ph.D. students start teaching before they complete their Ph.D. Although staff members, they cannot access certain information (e.g., future exam papers) or have full staff privileges because of their student status. Also, their salaries are different. In cases like this, although the two roles can be assigned to the same agent, the characteristics of the composed role are not exactly the characteristics of the two individual roles put together. The relation *Mergeswith* $\subseteq R \times R$ is symmetric:

$$\forall (r_1, r_2) : R \times R \; ((r_1 \text{ merge } r_2) \Rightarrow (r_2 \text{ merge } r_1))$$

Semantics of Role Relations

To describe the semantics of role relations, an agent organization is represented by a two-sorted algebra (*see Figure 2*). The algebra includes two sorts, A representing agents and R representing roles.

Let *Has*: $A \rightarrow R$ be a relation mapping agents to roles. The term "has" means that a role has been allocated to an agent by some role-allocation procedure or tool. It is possible for an agent to have roles that do not contribute to defining the agent behavior (e.g., when roles merge with other roles). For each $a \in A$, let *a.has* be the set of roles that the agent *a* maps to in the relation *Has*. In other words, *a.has* denotes the relational image of the singleton $\{a\} \subseteq A$ in the relation *Has*.

Let *Plays*: $A \rightarrow R$ be a relation mapping agents to roles again. The term "plays" means that the behavior a role represents is actively demonstrated by the agent (e.g., the role does not merge with other roles that are also played by the agent). For each $a \in A$, let *a.plays* denote the set of roles that the agent *a*

maps to in the relation *Plays*. In other words, *a.plays* denotes the relational image of the singleton {a} $\subseteq A$ in the relation *Plays*.

By definition, all agents must have the roles they play:

$\forall\ a : A, r : R \cdot\ (r \in a.plays \Rightarrow r \in a.has)$

The meaning of the relations between roles previously introduced can now be described as follows:

- **Equals**. An agent has and plays equal roles at the same time.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ eq } r_2 \Leftrightarrow ((r_1 \in a.has \Leftrightarrow r_2 \in a.has) \wedge (r_1 \in a.plays \Leftrightarrow r_2 \in a.plays)\ ))$

- **Excludes**. Excluded roles cannot be assigned to the same agent.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ not } r_2 \Leftrightarrow \neg(r_1 \in a.has \wedge r_2 \in a.has))$

- **Contains**. Contained roles must be assigned and played by the same agent as their containers.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ in } r_2 \Leftrightarrow ((r_2 \in a.has \Rightarrow r_1 \in a.has) \wedge (r_2 \in a.plays \Rightarrow r_1 \in a.plays)\ ))$

- **Requires.** Required roles must be played by the same agent as the roles that require them.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ and } r_2 \Leftrightarrow (r_1 \in a.plays \Rightarrow r_2 \in a.plays))$

- **Addswith**. There is no constraint in having or playing roles that add together.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ add } r_2 \Leftrightarrow (r_1 \in a.has \Rightarrow ((r_2 \in a.has \vee r_2 \notin a.has) \wedge (r_2 \in a.plays \vee r_2 \notin a.plays))))$

- **Mergeswith**. When two roles merge, only the unique role that results from their merger is played by an agent.

$\forall a : A, (r_1, r_2) : R \times R \cdot\ (r_1 \text{ merge } r_2 \Leftrightarrow \exists 1\ r_3 : R \cdot ((r_1 \in a.has \wedge r_2 \in a.has) \Rightarrow (r_1 \notin a.plays \wedge r_2 \notin a.plays \wedge r_3 \in a.has)))$

For example, let us assume that roles r_2 and r_3 merge, resulting in role r_4. Based on the above semantic definition, if an agent has r_2 and r_3 then it must also have r_4 and it must not play r_2 and r_3 (the agent may or may not play r_4 depending on the relations of r_4 with its other roles). The example of a *Mergeswith* relation between roles r_2, r_3, and r_4, assigned to agent a_2, is depicted in Figure 2. The fact that agent a_2 has all three, r_2, r_3, and r_4, is represented by a dotted line corresponding to the relation *Has*. The fact that agent a_2 can possibly play r_4 but definitely cannot play r_2 and r_3 is represented by a solid line corresponding to the relation *Plays*.

Using the above semantic axioms, it is trivial to verify that the properties of role relations introduced in the preceding section hold.

Finally, relations between more than two roles can be defined in a similar

manner. In that case, a predicate notation is more convenient for representing role relations. For example, when three roles, r_1, r_2, and r_3, merge to r_4, this can be noted by merge(r_1, r_2, r_3, r_4). In this paper, no formal definitions of relations among roles with arity greater than 2 are provided.

Role-Algebraic Multi-Agent System Design (RAMASD)

Role relations, as defined in the above algebra, restrict the way in which roles can be allocated to agents. Therefore, the agent organization design problem is transformed to a constraint satisfaction problem that must be solved for roles to be allocated to agents. The problem can be constrained further by including constraints based on general design heuristics. These constraints are expressed on the performance variables of the agents. For example, the system designer should be able to define the maximum number of roles an agent could play or an upper limit to the resource capacity an agent would require. Role-allocation heuristics could also be specified. For example, roles requiring access to similar resources could be assigned to the same agent.

The manual and automatic steps in the semiautomatic approach to role-based agent-organization design are the following:

1. *Select role models*. There are many ways to carry out role-based analysis. The most common approach is to start from use cases and for each identify roles and their interactions [1]. Many role-interaction patterns can be used directly from existing role-pattern libraries like the one documented at BT [22]. Selection of role-interaction patterns or definition of appropriate role models is a manual step that must be carried out by humans.
2. *Retrieve default role characteristics and compositional constraints*. This is an automatic step concerning only reused role models, since default role characteristics and interrole relations are expected to be stored in a role library from which the agent system designer will be selecting role models. After the designer selects existing role models, role characteristics and role compositional constraints are automatically retrieved.
3. *Refine role models*. The agent system designer is expected to manually specify role characteristics and role relations for any newly defined role models. These new role models should be stored in the role-model library for later use. At this step, additional characteristics of currently reused role models (e.g., additional performance variables) should also be specified.
4. *Specify general design constraints*. This is also a manual step where various requirements are modeled as constraints on the performance variables of roles and agents. For example, in the case study discussed next, the fact that each agent should have access to at most one information source is modeled by requiring that the value of the agent performance variable *database* is at most 1.
5. *Assign roles to agents*. Solving the constraint-satisfaction problem and allocating roles to agents can be done automatically. The solution to

a constraint-satisfaction problem consists of agent types. The term *agent type* refers to a collection of roles an agent can be assigned according to a particular role-allocation solution. Agent types can thus be considered as high-grain conceptual representations of agent behavior. After an agent type has been defined, it can be instantiated to create and deploy as many identical agents components as required. For example, assuming that an agent type is assigned only the role *Customer*, a customer agent component can be created and deployed for each registered customer of the business the agent-based system supports. A heuristic algorithm will be presented below that returns the first-found role-allocation solution, trying to allocate all available roles to as few agent types as possible. The algorithm is demonstrated in the case study example.

Example: An Automotive Industry B2B Exchange

For the purposes of this paper, an example extracted from a large case study concerning an automotive industry B2B exchange is considered. The example is based on a simple B2B electronic commerce model involving the three business phases of quotation, negotiation, and order fulfillment. The use of RAMASD to design a multi-agent system implementing the B2B exchange services involved in the example is demonstrated.

Case Study Overview

Automotive industry B2B exchanges are electronic business service providers offering a variety of services, including business directories, auctions, supply-chain management, and asset redeployment and disposal [26]. The idea of such efforts is to bring companies from the automotive industry together and enable them to carry out their business in a more cost-effective and convenient manner using Internet technology. Automotive industry manufacturers are able to interact with their suppliers without having to interface different information technology systems. Apart from effectively transacting with their customers, suppliers are also able to conveniently consolidate their efforts, thus maximizing their enterprise capability and the ability to pursue other business opportunities. All parties benefit from the utility applications available to B2B exchange participants, such as corporate services and customer relationship–management software [13].

A representative example of an automotive industry B2B exchange is Covisint (COllaboration, VIsion, and INTegration), initiated by Daimler-Chrysler, Ford, and General Motors to create an optimized digital supply chain for the automotive industry [11]. Additional drivers for the creation of Covisint were the expected cost savings and improved product lifecycle management based on sophisticated software support. Currently, fourteen automotive industry key players have joined Covisint together with two technology partners, Commerce-One and Oracle. Covisint supports supply-chain

management, collaboration among automotive market business parties, procurement, quality control, and corporate financial processes. The functionality it offers is the basis for the case study considered in this paper.

An important issue in business-to-business transactions is the underlying electronic commerce model. Early attempts to characterize e-commerce transactions were based on the standard consumer buying behavior (CBB) model [15], which includes six stages: Need Identification, Product Brokering, Merchant Brokering, Negotiation, Purchase and Delivery, and Product Service and Evaluation. In the *Need Identification* phase, the customer conceptualizes the need for a product or service. In the *Product Brokering* and *Merchant Brokering* phases, the customer decides which product or service is needed and selects a suitable supplier or service provider. In the *Purchase and Delivery* phase, the product is delivered or the service is provided, and in the *Product Service and Evaluation* phase the customer advises of its satisfaction with the process, products, or services provided.

More recent work on electronic commerce models produced models that capture specific issues of e-commerce. One such model is the electronic service marketplace model developed at HP labs [12]. The HP e-commerce model includes five phases: Creation, Discovery, Negotiation and Contracting, Monitor and Management, and Fulfillment and Settlement. In the *Creation* phase, the electronic marketplace is established by appropriate bodies that will be responsible for coordinating the market operation. The potential trading parties discover one another and explore possibilities for trading in the *Discovery* phase. The actual trading process results in service contracts and product orders, and takes place in the *Negotiation and Contracting* phase. In the *Monitor and Management* phase, the legitimate operation of the marketplace is monitored and evaluated by some appropriate inspection body, and appropriate actions are taken where required. Finally, in the *Fulfillment and Settlement* phase, trading parties fulfill the terms of the agreements, and products are shipped or service provision starts.

To better illustrate the approach discussed in this paper, here is a simple B2B electronic commerce model abstracted from Durante et al. [12] and Guttman et al. [15]. The model has three phases:

- *Quotation Phase*. Potential trading parties discover one other. Quotations about automotive manufacturing industry parts and supporting services are issued.
- *Auction/Negotiation Phase*. Potential buyers establish an auction. Buyers and sellers negotiate and reach agreements regarding supplying products and providing services. The agreements are examined by an appropriate inspection body (e.g., the Federal Trade Commission) in regard to legal, ethical, and social issues. Appropriate action is taken where required.
- *Fulfillment Phase*. The contracts agreed to in the negotiation phase are executed (e.g., the shipping orders of purchased products are submitted to the appropriate departments and the provision of hired services commences). This phase includes all communication

events relevant to customer input about the quality of the products received and the services provided.

Since the complete automotive industry case study is quite large, an example is extracted to demonstrate the concepts and approach discussed in this paper. The example assumes that it is necessary to design a multi-agent system to support routine automotive manufacturing supply-chain management tasks. In the example scenario, automotive industry manufacturers first identify their needs for automotive manufacturing parts and services using their proprietary, possibly legacy systems. Subsequently, they search product catalogs and business directories for suitable potential suppliers and service providers. For example, it is common for car manufacturers to reduce costs and increase focus on their main tasks by outsourcing the manufacturing of seats and exhausts and the redeployment or disposal of used assets to specialized companies. A potential buyer that identifies suitable potential suppliers or service providers initiates an auction and invites them to participate. Invited trading partners conduct the auction, and the potential buyer may accept or reject the outcome based on whether its personal business goals are satisfied. A representative of the inspection body checks the auction process and outcome. If they are approved, the signed contracts are ready for execution. Finally, the suppliers and service providers fulfill their contracts by shipping the relevant products and starting to provide the relevant services.

Role Identification

In order to model the above system in terms of roles, the first thing to do is to identify the roles involved in the case study example. According to Kendall and Zhao, a way to identify roles in an application domain is to identify use cases that are each associated with a goal, then create a goal hierarchy from the use-case hierarchy, and finally, coalesce semantically relevant goals to roles [23]. For the purpose of the automotive industry example, let us consider three use cases, each corresponding to a phase in the simple B2B e-commerce model described before:

- *Trading partner discovery and request for quotation (Quotation Phase).* This activity involves extensive information exchange among potential trading partners. Each side must sift through large amounts of data for relevant information to make decisions, proposals, and counterproposals. The outcome of this activity is a number of potential suppliers or service providers for each potential buyer.
- *Auction Initiation, Negotiation, and Monitoring (Auction/Negotiation Phase).* This involves initiation and establishment of an auction from each potential buyer, negotiation between the trading parties, and monitoring of the auction process and results by some external inspection body.
- *Order Fulfillment (Fulfillment Phase).* In this activity, all interactions regarding execution of contracts, shipment of products, and provision of services take place.

Each use case has a number of high-level goals (*see Figure 3*). The behavior leading to achieving these goals can be modeled by appropriate roles. Thus the following roles can be identified (*see Figure 4*):

1. *Potential_Buyer (goal Q_1).* This role describes the generic behavior of automotive industry manufacturers interested in purchasing manufacturing parts, outsourcing some of their business processes, or selecting collaborators for co-design projects regarding sophisticated automotive manufacturing parts. Potential buyers communicate with potential suppliers or service providers and request quotations and relevant information. Suppliers that have submitted attractive quotations are invited to participate in an auction.
2. *Potential_Trader (goal Q_2).* Potential traders are suppliers or service providers that communicate with potential buyers providing them with quotations and further information. Potential traders also communicate with one other in attempts to establish coalitions and submit more attractive offers to potential buyers.
3. *Auction_Operator (goal A_1).* This role describes the generic behavior of members of auction operation support groups (e.g., accessing common auction information including bidding history and trading participants status).
4. *Auction_Coordinator (goal A_{11}).* This role describes the behavior required to coordinate the operation of an auction. This includes informing the trading parties about the auction regulations, providing participant profiles, and gathering statistical data on their bidding histories. The performance of auction participants and the efficiency of the auction mechanisms are monitored.
5. *Auction_Inspector (goal A_{12}).* This role ensures the smooth operation of the auctioning process. The *Auction_Inspector* accesses the auction data gathered by the *Auction_Coordinator* and verifies that the process followed is legitimate. This is achieved by comparing auction process data with the auction rules and regulations obtained by communicating with the *Legislation_Interface*, defined below.
6. *Legislation_Interface (goal A_2).* This role maintains a database of rules and legislation that govern the auction operations. Auction inspectors interact with this role and submit queries regarding auction procedures, receiving answers in various data formats.
7. *Invited_Auction_Participant (goal A_3).* This is a utility role providing access to auction operations to selected suppliers and service providers. This involves authorization codes to participate in the auction business and provides access to profiles of other participants and to historical auction data.
8. *Auction_Initiator (goal A_4).* This role is responsible for initiating and running the auction. Its duties include selecting the auction type, the bid limits, and the starting price. Subsequently, it participates in the

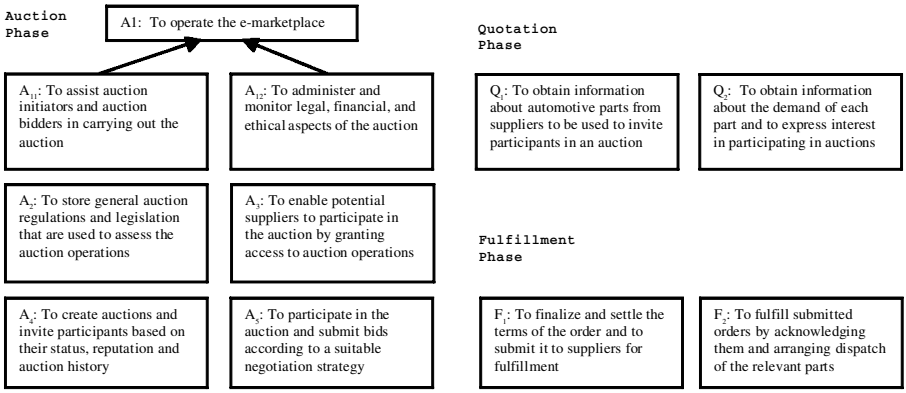

Figure 3. Use Case Goals for an Automotive Industry B2B Exchange Case Study

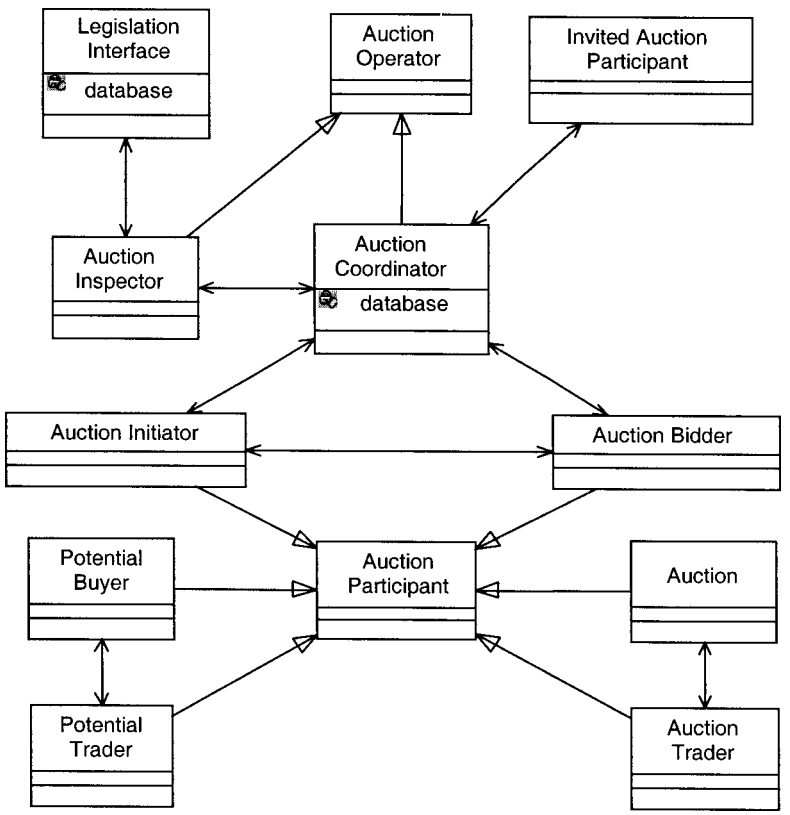

Figure 4. Role Models for the Automotive Industry B2B Exchange Case Study

negotiation with auction bidders by accepting bids and establishes a product-purchasing or service-provision agreement.

9. *Auction_Bidder (goal A_5).* This role participates in the auction and submits bids aiming to achieve a business contract at a beneficial price.
10. *Auction_Buyer (goal F_1). Auction_Buyers* are active in the order fulfillment phase and interact with suppliers and service providers to finalize the details of product shipment and service provision start. They also ensure receipt of products and smooth utilization of the contracted services by interacting with inventory and proprietary workflow-management software.
11. *Auction_Trader (goal F_2).* This role is also active in the order-fulfillment phase and interacts with buyers to confirm receipt of shipped parts and prompt initiation of agreed services. Its responsibilities include notifying the shipping departments to execute shipment orders, informing the service departments to start provision of the contracted services, and interacting with the logistics and accounting departments to ensure that appropriate payment is received.
12. *Auction_Participant.* As noted by Andersen [1], it is a good practice in role-modeling to extract any common behavior from a number of roles and model it separately in a new role. From the roles identified above, *Auction_Initiator*, *Auction_Bidder*, *Potential_Buyer*, *Potential_Trader*, *Auction_Buyer*, and *Auction_Trader* have some behavior in common. They all represent automotive industry parties that interact in the B2B exchange environment. This common behavior is modeled by a separate role that facilitates understanding of the resulting role models and specifies necessary constraints among roles.

Modeling Nonfunctional Aspects

In the case study example, two nonfunctional aspects are taken into account in designing the multi-agent system: *security* and *privacy*. The security strategy implemented in the designed multi-agent system distributes access to different information sources to different software agents. Privacy is ensured by intermediation of trading interactions.

Security Issues

Creating effective electronic business software security strategies and infrastructures is one of the biggest challenges in the electronic business software industry. IDC predicts that the U.S. information security services market will grow from $2.8 billion in 1999 to more than $8.2 billion in 2004 [16].

Common B2B software security requirements include identification/authentication of users to enter the system, authorization to enable them to access the permitted software functionality, user accountability, administration,

and, most important, asset protection. Security strategies typically balance the degree of support for each requirement according to the general policies of the business. For example, higher access privileges for users result in lower software system security.

A security strategy initially requires high-level recognition of the business security concerns, which can be described as simple statements. Examples of high-level security concerns include monitoring user activity, forbidding access to unneeded data, and promoting security awareness among employees. Based on high-level descriptions of security concerns, more specific descriptions of security policies are introduced. Security policies are meant to address security issues when implementing business requirements. Examples of security policies are using out-of-band communication when responding to an incident alert, employing encrypted data exchange techniques, and maintaining a central transaction log server. Security policies lead to specific security actions, for example, disabling telnet and ftp in all externally accessible computers, validating html form data on both the client and server side, and adding an extra authorization level for very sensitive and important data.

There have been many approaches to classifying security strategies for possible reuse. One common approach applies the *limited view* security strategy discussed by Yoder and Baraclow [44]. According to the limited view strategy, users see only what they are allowed to access. Another typical strategy to strengthen the security of a distributed application is to provide a *secure access layer* combining both application and low-level network security [32, 44].

Based on these two security strategies, agents should not only exchange information using secure protocols and over a secure communication medium, but should not have access to information resources relevant to the operation of incompatible roles. The reason is that agents are highly flexible and configurable software components that can alter their behavior on runtime. For example, the goals of the *Auction_Coordinator* role could lead to attempts to modify an auction legislation database if it had access to it. In the example here, a very simple security policy for role-based access control will prohibit the access of any agent to more than one database, where *database* refers to the set of all data sources relevant to a single role.

Privacy Issues

Consumers and organizations that do business over the Internet want assurance that their transactions will remain private and outside parties cannot access sensitive personal data. This is especially true in the emerging automotive industry B2B models that include vendors and external partners early in the business process, from product design through delivery and support. Competitors sometimes cooperate to complement one another's capabilities. For example, in the defense automotive sector, multiple manufacturers collaborate on contracts because of size, complexity, and the need for specialized services. Or a manufacturer may team up with a parts supplier that is also collaborating with its competitors. For example, an automotive supplier pro-

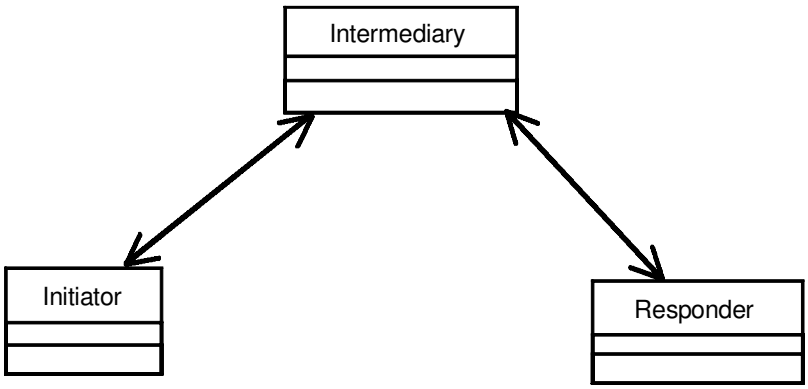

Figure 5. The Mediator Pattern

ducing seats may be working with several competing auto manufacturers on future designs. As an organization increases the size of its network, the variety of its markets, inputs, and outputs increases, and this too increases its need for privacy [25].

Although the technology exists to ensure privacy in personal and business communications and data, many companies that acquire private data from customers do not apply the necessary privacy practices. Therefore, software design solutions that ensure privacy among trading parties are required. It has been suggested that to support privacy Internet-based software should be based on a centralized data model, with nonpublic information disseminated to interacting parties by a trusted third party [2]. Intermediation has been successfully used in many application domains to enforce privacy, including electronic stock markets, manufacturing, and mobile workforce management [21, 35, 41].

Intermediation can be modeled by the *mediator role interaction* pattern.[3] This pattern involves three roles (*see Figure 5*):

1. *Initiator.* This role is active in the order-fulfillment phase and interacts with buyers to confirm receipt of shipped parts and prompt initiation of agreed services. Its responsibilities include notifying shipping departments to execute shipment orders, informing service departments to start provision of contracted services, and interacting with logistics and accounting departments to ensure that appropriate payment is received.
2. *Intermediary.* This role has access to all relevant information of both interacting parties. However, it does not just filter and selectively communicate information to initiators and responders. It can also reduce the costs of many information-intensive tasks by integrating customer-based functionality with privacy and security issues. For example, the intermediary can maintain a database of previous interactions among the same or relevant participants and can provide aggregated results considering any privacy limitations.

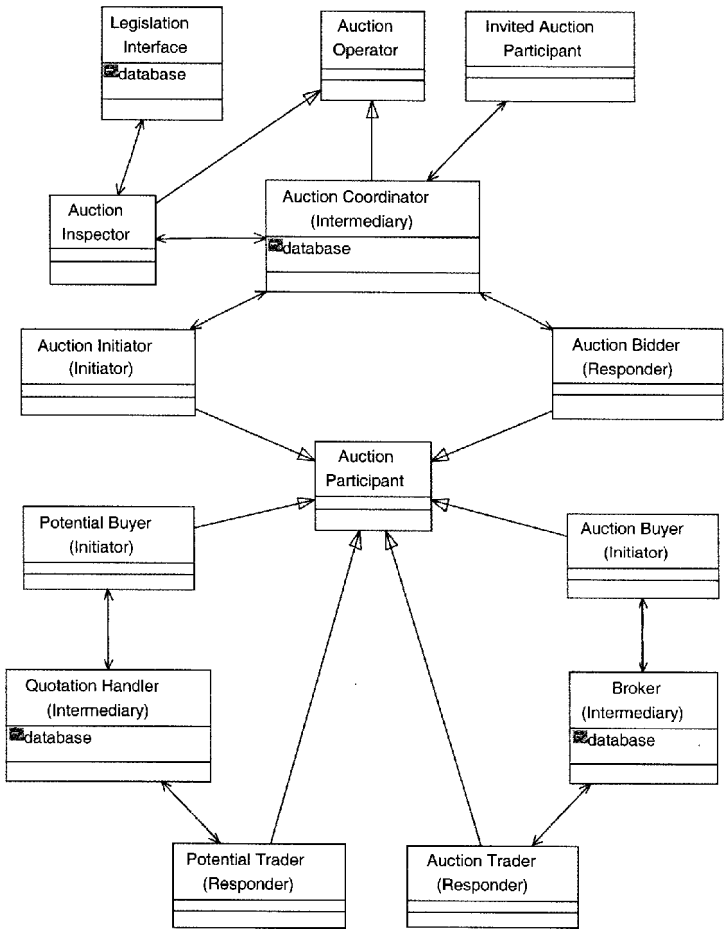

Figure 6. Updated Role Models Based on the Mediator Pattern

3. *Responder.* This role is similar to the *Initiator* role except that the *Responder* role responds and continues an interaction previously started by the *Initiator* role.

Role Composition

Role composition takes place when the mediator role interacts with the roles identified previously. The resulting role models include the mediator pattern. The differences are depicted in Figure 6, which describes the resulting role models. Some roles remain the same, and some are replaced with new roles based on a *Mergeswith* relationship. The new roles are named by combining the names of the roles that contributed to their creation. The role-pairs *Potential_Buyer—Potential_Trader*, *Auction_Initiator—Auction_Bidder*, and *Auction_Trader—Auction_Buyer* are replaced because the new roles do not interact directly but only through intermediaries. The new roles can inform the

intermediaries about their privacy requirements and make use of the utility services (e.g., statistical analyses) provided by the intermediaries. The *Auction-Coordinator* role is also replaced, and the new role acts both as an *Intermediary* and an *Auction_Coordinator*.

Two new roles are needed at this stage:

1. *Quotation Handler.* This role provides a means to enhance the communication between potential buyers and potential suppliers. Apart from intermediation, *Quotation_Handler* offers various services to both parties (e.g., electronic document management, analysis of data gathered throughout similar automotive sourcing requests for quotation). The role has access to a central repository of service sourcing documentation and product price lists.
2. *Broker.* This role intermediates between buyers and suppliers or service providers in the fulfillment phase. It provides the functionality for many utility tasks (e.g., bills of materials, order management, shipping management, returns and status tracking) and maintains a database of order fulfillment critical information (e.g., inventory levels, usage history and patterns, receipts) to help eliminate excess inventory and premium transportation charges.

Specifying Design Constraints

After identifying roles in the application domain and modeling nonfunctional aspects using well-known role interaction patterns, the next step is to model the remaining functional and nonfunctional aspects using constraints on roles and agent and role characteristics. This can be done with Role Constraint Language (RCL), a simple declarative specification language introduced to represent design constraints on roles and agent and role characteristics. The RCL syntax is simple and intuitive in order to facilitate the specification of constraints among roles. The underlying RCL semantic model is based on the algebraic semantics of the role algebra presented earlier. The basic concepts of RCL will be illustrated using the specification of the compositional constraints for the roles identified above, part of which is presented in Figure 7. For clarity, the role names in Figure 7 have been abbreviated when necessary.

An RCL specification contains sections corresponding to role definitions, role constraints, and general constraints. The role definitions section specifies the names of the roles that will be considered in the design of the multi-agent system. More than one role name can be specified at the same specification statement. Furthermore, role characteristics (e.g., role collaborators and performance variables) can be associated with role names. This is illustrated in Figure 7 by specifying that the roles *Legislation_Interface*, *Auction_Coordinator*, *Quotation_Handler*, and *Broker* have associated the integer performance variable *database*. Role characteristics are assigned values in the same section. The syntax for referring to the characteristics of each role is similar to that of common programming languages, using a '.' to link the role name and the role characteristic name.

```
/* ROLE DEFINITIONS */

Role a_operator, a_participant,
     a_inspector, a_coordinator,
     a_initiator, a_initiator_i,
     a_bidder, a_bidder_r,
     p_buyer, p_buyer_i,
     p_trader, p_trader_r,
     a_buyer, a_buyer_i,
     a_trader, a_trader_r,
     ia_participant, initiator,
     intermediary, responder;

Role l_interface, a_coordinator,
     q_handler, broker {
     int database;
}
     l_interface.database = 1;
     a_coordinator.database = 1;
     q_handler.database = 1;
     broker.database = 1;

/* ROLE CONSTRAINTS */

in(a_coordinator, a_operator);
in(a_inspector, a_operator);
in(a_initiator, a_participant);
in(a_bidder, a_participant);
in(p_buyer, a_participant);
in(p_trader, a_participant);
in(a_buyer, a_participant);
in(a_trader, a_participant);

and(a_bidder, ia_participant);

merge(a_coordinator, intermediary,
      a_coordinator_int);

merge(p_buyer, initiator, p_buyer_i);
merge(a_initiator, initiator,
      a_initiator_i);
merge(a_buyer, initiator, a_buyer_i);

merge(p_trader, responder, p_trader_r);
merge(a_bidder, responder, a_bidder_r);
merge(a_trader, responder, a_trader_r);

not(a_participant, a_operator);
not(l_interface, a_coordinator);
not(l_interface, a_participant);

not(a_coordinator, a_inspector);
not(a_coordinator, a_participant);
not(q_handler, a_participant);
not(q_handler, a_inspector);
not(broker, a_participant);
not(broker, a_inspector);

not(a_initiator, a_bidder);
not(p_buyer, p_trader);
not(a_buyer, a_trader);

/* GENERAL CONSTRAINTS */

Constraint Y {
      forall a:Agent {
         a.database <= 1
      }
}
```

Figure 7. Compositional Constraints for the B2B Exchange Case Study

In the role constraints section, any constraints between roles are specified in prefix form. For example, `in(a_coordinator, a_operator)` specifies that an `a_coordinator` contains the `a_operator` role. By using appropriate constraints between roles, it is possible to specify how different roles should be allocated to agent types. For example, the *Excludes* relation is used to specify that an agent that is *Auction_Coordinator* cannot also be *Auction_Inspector*. The *Excludes* relation is also used to specify that interacting roles in the request for the quotation, auction negotiation, and order fulfilment processes should be played by different agents.

In order for an agent to be an *Auction_Bidder* and participate in an auction, it must have been previously invited by *Auction_Initiator*. This is modeled using the *Requires* relation to specify that the *Auction_Bidder* role must be played together with the *Invited_Auction_Participant* role. Changes in the behavior of roles when the mediator pattern is applied are modeled using the *Mergeswith* relation. For example, an *Auction_Coordinator* merges with the *Intermediary* role resulting in the *Auction_Coordinator_Intermediary* role, which combines the behavior of the *Auction_Coordinator* and *Intermediary* roles.

For security reasons, neither *Auction_Coordinator* nor *Auction_Participant* can coexist with the *Legislation_Interface* role. For the same reason, *Auction_Coordinator* cannot coexist with *Auction_Inspector,* and *Auction_Coordinator,*

Quotation_Handler, and *Broker* cannot coexist with *Auction_Participant*. These constraints are specified using the *Excludes* relation.

Constraints on agent and role characteristics are specified in the general constraints section. For example, to increase security, one would like agents to have access to not more than one information source. This is modeled by constraining the *database* performance variable to be at most 1 for all agent types. The value of the *database* variable of an agent type is equal to the sum of the values of the *database* variables of the roles the agent plays. This calculation only considers roles that have the *database* variable defined. For example, an agent that plays the *Quotation_Handler*, *Auction_Coordinator,* and *Auction_Operator* roles is automatically associated with the performance variable *database*, since at least one of the roles it plays is associated with this variable. Furthermore, the value of the agent *database* variable would be 2, since the values of the *Auction_Coordinator* and *Quotation_Handler database* variables are 1 each and the *Auction_Operator* role is not associated with a *database* variable.

Allocating Roles to Agent Types

The constraints on roles and on agent and role characteristics define a constraint satisfaction problem that must be solved to allocate roles to agent types. Since search problems are hard to address optimally, suboptimal solutions are widely adopted. This section gives an overview of a heuristic algorithm for solving the role-allocation problem and discusses its application to allocate roles to agent types in the case study example.

A Role-Allocation Algorithm

The algorithm aims to minimize the number of agent types produced. Therefore, it tries to allocate as many roles to an agent type as possible before moving to the next one. Merging roles are processed first, and the algorithm moves to the remaining roles only when all the roles involved in a *Mergeswith* constraint have been allocated to an agent type. Every role is allocated to an agent type at least once. An overview of the algorithm is given in Figure 8.

The algorithm starts with an empty agent type and randomly allocates it to a group of merging roles involved in a *Mergeswith* relation. Then it checks whether the merging roles are involved in any other role constraints. If they are, the additional roles involved in those role constraints are also allocated to the agent type. Subsequently, the algorithm checks whether the agent type is consistent, which involves examining whether all the constraints concerning the roles so far allocated to the agent type are satisfied.

If these role constraints are satisfied, the next step is to check whether any general constraints concerning role characteristics of this agent type are also satisfied. If they are, the roles involved are considered part of this agent type and the two steps given above are repeated for as long as there are still merging roles to allocate. If allocating any further merging roles to the agent type results in constraints that are not satisfied, then a new agent type is created and the process is repeated. If there are constraints that are not satisfied and

```
1. Create a new agent type t.
2. While there are remaining unprocessed merging
   relationships:
      a. Create agent type t' = t.
      b. Allocate roles r_i involved in a merging
         relationship m to agent type t'.
      c. If t' is consistent for some allocation
         combination of r_i to t':
            i. Check any constraints on the performance
               variables of agent type t'.
           ii. If they are satisfied:
                  1. t = t'.
                  2. Goto step 2.
      d. If agent type t is empty then error.
3. While there are remaining unallocated roles:
      a. Create agent type t' = t.
      b. Allocate a role r to the agent type t'.
      c. If t' is consistent for some allocation
         combination of r to t':
            i. Check any constraints on the performance
               variables of agent type t'.
           ii. If they are satisfied:
                  1. t = t'.
                  2. Goto step 3.
      d. If agent type t is empty then error.
4. Terminate with success
```

Figure 8. A Simple Search Algorithm for Allocating Roles to Agent Types

the agent type was initially empty, the algorithm stops with an error message.

When all the merging roles are finished and there are still roles available, the algorithm attempts to allocate the remaining roles to the current agent type. In case of failure, a new agent type is created and the process continues until all available roles have been allocated to agent types.

This simple and intuitive algorithm is currently being used as a baseline for comparisons in ongoing research about role-allocation algorithms. The algorithm works reasonably well for case study examples involving approximately 40 roles and having, on average, 10 merging role constraints, 20 other role constraints, and two general constraints. However, the algorithm becomes inefficient when the total number of roles increases, the number of merging role constraints decreases, or the total number of constraints increases.

Role-Allocation Results

An application of the algorithm to the RCL specification will now be presented. The results of the role allocation are summarized in Figure 9. The algorithm is assumed to start randomly from the `merge(a_coordinator, intermediary, a_coordinator_int)` constraint. These three roles are allocated to the first agent type. Since `a_coordinator` contains `a_operator`, the `a_operator` role is also allocated to this agent type. All the constraints so far involving the

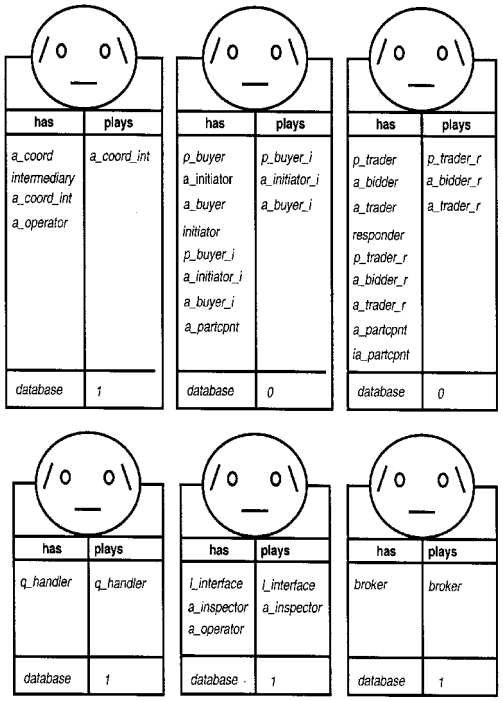

Figure 9. Agent Types for the B2B Exchange Case Study

allocated roles are satisfied. Subsequently, a check of the general constraint `a.database <= 1` is done, and it is successful because `a.database = a_coordinator_int.database = 1`.

The next step is to attempt to allocate roles from the `merge(p_buyer, initiator, p_buyer_i)` constraint. Since the role `p_buyer` contains `a_participant`, an attempt to also allocate `a_participant` is made. However, `a_coordinator` excludes `a_participant`, and therefore this allocation attempt fails. Subsequent attempts to allocate roles participating in the remaining merging constraints fail for the same reason.

The algorithm then proceeds to create a second agent type. In the second agent type, the roles involved in the next three merging constraints are successfully allocated. Since `p_buyer`, `a_initiator`, and `a_buyer` contain `a_participant`, the `a_participant` role is also allocated to this agent type and the general constraint is satisfied. The series of successful allocation steps is interrupted when the algorithm attempts to allocate roles from the `merge(p_trader, responder, p_trader_r)` constraint. The reason is that a `p_buyer` cannot be a `p_trader` in the auction type considered, which is specified in Figure 8 using an *Excludes* relation.

The next step is to create a third agent type where the roles involved in the remaining merging constraints are successfully allocated. Since `p_trader`, `a_bidder`, and `a_trader` contain `a_participant`, the `a_participant` role is also allocated to this agent type. The `ia_participant` role is also allocated because `a_bidder` requires `ia_participant`. As there are no

merging constraints left, the algorithm attempts to allocate the remaining roles. This attempt fails for this agent type because `q_handler`, `broker`, `a_operator`, and `l_interface` cannot coexist with `a_participant`, as specified in Figure 7.

A fourth agent type is therefore created, and the `q_handler` role is randomly allocated to it. The algorithm fails to allocate `l_interface` to this agent type because, as explicitly specified in Figure 7, `q_handler` cannot coexist with `l_interface`. The algorithm also fails to allocate `a_inspector` because in that case it would also have to allocate `a_operator`, contained by `a_inspector`, which is excluded by `q_handler`. Finally, `broker` cannot be allocated, since in that case the agent *database* value would be 2 and the general constraint would be violated.

This leads to the creation of a fifth agent type where the `l_interface` and `a_inspector` roles are successfully allocated. Since `a_inspector` contains `a_operator`, the `a_operator` role is also allocated to this agent type. However, the `broker` role cannot be allocated, since it cannot coexist with `a_inspector`, as specified in Figure 7.

Finally, a sixth agent type containing the remaining role `broker` is created, and the algorithm ends with success.

Comparison with Relevant Work

The existing approaches to agent-based system engineering can be either *dynamic*, where the agent behavior is dynamically specified at runtime, or *static* where the agent system is engineered statically at design time. Static approaches can be ad hoc, *formal*, or *informal*. RAMASD is a static, semiautomatic approach that formalizes some of the modeling primitives, the relations among roles, thus enabling automatic tool support for designer and convenient reuse of design knowledge. Furthermore, RAMASD explicitly models nonfunctional requirements.

An increasing amount of work has been done on the use of software agents for B2B electronic commerce (e.g., [3, 7, 24]) and B2B cross-organizational workflow management (e.g., [19, 31, 36]). However, there have not been many systematic approaches suited for engineering of agent-based B2B systems. A notable exception is the Interaction-Oriented Programming (IOP) approach, which emphasizes engineering agents to that achieve coherence throughout cross-organizational processes [37]. In IOP, multienterprise workflows are flexibly enacted by enabling agents to enter into and behave according to their *commitments* to one another. Considering the commitments they have acquired and the capabilities they possess, agents autonomously adopt and abandon various roles until their commitments are fulfilled. However, with IOP the agent system designer has to work at a low level of abstraction, and IOP does not explicitly model nonfunctional requirements. Furthermore, the reorganization necessary for the agents to decide about the roles they need to play can be resource-consuming and may result in unstable systems because reorganization messages may take a long time to propagate.

Every approach to agent system design that is based on massive reorgani-

zation during runtime suffers from similar weaknesses. For example, Tambe describes a dynamic agent organization approach that assigns task plans to agents that already play the necessary roles and have the appropriate capabilities [40]. This approach assumes that suitable agents already exist in cyberspace and can be allocated task plans as required. The task plan allocation method aims to utilize the capabilities of an agent to the maximum extent before allocating task plans to a different agent. Although this straightforward practice facilitates design of the agent system, it does not explicitly consider nonfunctional requirements (e.g., fully utilized agents could become a system bottleneck, resulting in poor system performance). To mitigate such problems, RAMASD aims to optimize the initial agent-system design so that the need for dynamic reorganization on runtime is reduced.

Several other methodologies also focus on engineering the agent system statically on design time. These methodologies originate in object-oriented analysis and design (e.g., [22, 28, 43]), knowledge engineering (e.g., [9, 17]), formal methods (e.g., [33]), and information systems engineering (e.g., [8]). The notions of role and role-based modeling appear in most of the aforementioned agent-based system engineering approaches. However, their definitions of role are limited to a conceptual level, whereas the role definition presented in this paper is extended to a pragmatic level in order to quantitatively model nonfunctional requirements.

We believe no other agent system engineering methodology explicitly considers nonfunctional requirements, apart from TROPOS, which at some stage includes introducing actors and subactors that contribute positively to nonfunctional requirements [8]. The same result can be achieved in RAMASD by introducing appropriate nonfunctional models. Furthermore, the majority of the aforementioned agent systems engineering methodologies are systematic but informal, which means that they cannot be automated to any extent by a software tool providing automatic support to the designer. In contrast, RAMASD is semiautomatic, because the relations among roles are formally described in the role algebra, and roles can be automatically allocated to agents. RAMASD has been implemented in the Zeus agent-building toolkit [27].

Approaches based on formal methods can be fully automated but require the agent system designer to work at a low level of abstraction [33]. In contrast, RAMASD considers role models as first-class design constructs enabling designers to work at a high level of abstraction and to conveniently reuse functional, nonfunctional, and organizational settings.

Semiautomating agent system design is an approach also followed by the Multiagent Software Engineering methodology (MaSE), which is also based on role models [39, 42]. The MaSE methodology consists of seven analysis and design stages and attempts to formally capture the relationships between models. The design stages are incorporated in an agent system development tool called AgentTool. Although MaSE aims at formally defining the steps from analysis to design of agent systems, there is no notion of organizational relationships between the agents, since agents are viewed as specializations of objects. Nonfunctional aspects are not explicitly supported, and collective behavior cannot be pragmatically reused, since modeling is done only at the conceptual level. In contrast, RAMASD models behavior at a pragmatic level,

including functional, nonfunctional, and organizational aspects, and formalizes relations among roles in a role algebra. This enables semiautomation of the design process and allows the designer to work at a high level of abstraction. As a result, the complexity of the design process is reduced.

Conclusions and Further Work

Existing approaches to agent organization design do not pay enough attention to reducing the complexity of the design process and do not pragmatically consider nonfunctional requirements on design time. The semiautomatic approach described in this paper addresses these concerns by extending the definition of role so that nonfunctional aspects can be pragmatically modeled and by defining a simple role algebra that facilitates the automatic allocation of roles to agents. This enables the practical reuse of functional, nonfunctional, and organizational design patterns by representing them as role models that can be manipulated in line with the proposed role algebra. Thus the complexity of the design process is reduced.

However, there are issues that are not addressed in this paper. For example, since agents can play different roles in different contexts, the possible contexts should be considered when designing agent organizations. In the near future, RAMASD will be extended to consider role playing within some context. As a longer-term research task, the role algebra will be used to study the impact of allocating and deallocating roles to agents on runtime when the system requirements change dynamically. Finally, the specification of multiple constraints on roles and their characteristics can lead to overconstrained problems. To investigate this issue, the efficiency of a number of algorithms for allocating roles to agents under different numbers and hardness of constraints will be examined.

NOTES

1. The terms *agent-based system* and *agent organization* are used interchangeably in this paper.
2. Activity in this context will represent the whole causal sequence of events and actions caused by one triggering event and will correspond to the UML's concept of "use case."
3. More information about the use of the mediator role interaction pattern can be found in E.A. Kendall, Agent analysis and design with role models, vol. 1: Overview (unpublished internal report), BT Exact Technologies, January 1999.

REFERENCES

1. Andersen, E.P. Conceptual modelling of objects: A role modeling approach. Ph.D. dissertation, University of Oslo, 1997.
2. Andrews, P., and Adler, S. *Privacy: Basing Service on Respect*. New York: Center for IBM E-business Innovation, IBM Labs, 2001 (www-1.ibm.com/

services/strategy/files/IBM_Consulting_Privacy_Basing_service_on_respect.pdf).

3. Bartelt, A., and Lamersdorf, W. Agent-oriented concepts to foster the automation of e-business. In A.M. Tjoa, R.R. Wagner, and A. Al-Zobaidi (eds.), *Proceedings of the Eleventh International Workshop on Database and Expert Systems (DEXA 2000)*. Munich: IEEE Computer Society Press, 2000, pp. 775–779.

4. Biddle, B.J. *Role Theory: Expectations, Identities and Behaviors*. London: Academic Press, 1979.

5. Blake, M.B. WARP: An agent-based process and architecture for workflow-oriented distributed component configuration. In H.R. Arabnia (ed.), *Proceedings of the 2000 International Conference on Artificial Intelligence (IC'AI2000) (Session on Software Agent-Oriented Workflow)*. Las Vegas: CSREA Press, 2000, pp. 205–213.

6. Blake, M.B. Innovations in software agent-based B2B technologies. In *Workshop on Agent-Based Approaches to B2B at the Fifth International Conference on Autonomous Agents (AGENTS 2001)*. Montreal: ACM Press, 2001, pp. 1–7.

7. Blake, M.B.; Cornett, T., and Piquado, T. Using intelligent agents in conjunction with B2B interoperability. In *Proceedings of the 2001 International Conference on Artificial Intelligence (IC'AI2001) (Session on Agent-Oriented Software Architectures for B2B)*. Las Vegas: CSREA Press, 2001, pp. 538–545.

8. Bresciani, P.; Perini, A.; Giunchiglia, F.; Giorgini, P.; and Mylopoulos, J. A knowledge level software engineering methodology for agent oriented programming. In E. Andre and S. Sen (eds.), *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS 2001)*. Montreal: ACM Press, 2001, pp. 648–655.

9. Caire, G.; Coulier, W.; Garijo, F.; Gomez, J.; Pavon, J.; Leal, F.; Chainho, P.; Kearney, P.; Stark, J.; Evans, R.; and Massonet, P. Agent-oriented analysis using Message/UML. In M.J. Wooldridge, G. Weis, and P. Ciancarini (eds.), *Agent-Oriented Software Engineering II: Second International Workshop (AOSE 2001), Montreal, Canada*. Berlin: Springer-Verlag, 2001, pp. 151–168.

10. Chung, L.; Nixon, B.A.; Yu, E.; and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*. New York: Kluwer Academic Publishers, 2000.

11. Covisint. *Covisint Solution Suites*. Covisint, LLC., 2002 (www.covisint.com).

12. Durante, A.; Bell, D.; Goldstein, L.; Gustafson, J.; and Kuno, H. *A Model for the E-Service Marketplace*. Palo Alto, CA: HP Laboratories, 2000 (www.hpl.hp.com/techreports/2000/HPL-2000–17.html).

13. Electronic Marketplaces Source Guides. *Business to Business Marketplaces to the Automotive Industry*. Momentum Technologies LLC, 2000 (www.sourceguides.com/markets/byI/auto/byC/B2B/B2B.shtml).

14. Ferber, J., and Gutknecht, O. A meta-model for the analysis and design of organizations of multi-agent systems. In E. Durfee, M. Georgeff, and N.R. Jennings (eds.), *Proceedings of the International Conference in Multi-Agent Systems (ICMAS 98)*. Paris: IEEE Computer Society Press, 1998, pp. 128–135.

15. Guttman, R.H.; Moukas, A.G.; and Maes, P. Agent-mediated electronic

commerce: A survey. *Knowledge Engineering Review, 13*, 2 (July 1998), 147–159.
16. IDC. Strengthening end-to-end e-business security and privacy. IDC Rep No. 23166, 2000 (www-1.ibm.com/partnerworld/pwhome.nsf/vAssetsLookup/IDC_E2E.pdf/$File/IDC_E2E.pdf).
17. Iglesias, C.A.; Garijo, M.; Gonzalez, J.C.; and Velasco, J.R. Analysis and design of multi-agent systems using MAS-CommonKADS. In M.P. Singh, A.S. Rao, and M.J. Wooldridge (eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL '97)*. Berlin: Springer-Verlag, 1998, pp. 313–326.
18. Jennings, N.R. An agent-based approach for building complex software systems. *Communications of the ACM, 44*, 4 (April 2001), 35–41.
19. Jennings, N.R.; Faratin, P.; Norman, T.J.; O'Brien, P.; and Odgers, B. Autonomous agents for business process manage5ment. *Applied Artificial Intelligence, 14*, 2 (February 2000), 145–189.
20. Karageorgos, A. Reducing complexity of agent-based system design. Ph.D. dissertation, University of Manchester Institute of Science and Technology, 2002.
21. Kauffman, R.J.; Subramani, M.; and Wood, C.A. Analyzing information intermediaries in electronic brokerage. In R.H. Sprague, Jr. (ed.), *Proceedings of the 33rd Hawai'i International Conference on System Sciences*. Maui: IEEE Computer Society Press, 2000.
22. Kendall, E.A. Role models: Patterns of agent system analysis and design. *BT Technology Journal, 17*, 4 (October 1999), 46–57.
23. Kendall, E.A., and Zhao, L. Capturing and structuring goals. In P. Bramble, G. Gibson, and A. Cockburn (eds.), *Workshop on Use Case Patterns, Object Oriented Programming Systems Languages and Architectures (OOPSLA)*. Vancouver: ACM Press, 1998.
24. Maes, P.; Guttman, R.H.; and Moukas, A.G. Agents that buy and sell: Transforming commerce as we know it. *Communications of the ACM, 42*, 3 (March 1999), 81–91.
25. McConnell, M., and Hamilton, B.A. Information assurance in the twenty-first century. *IEEE Computer (Supplement on Security and Privacy), 35*, 4 (April 2002), 16–19.
26. Morgenthal, J.P. Which B2B exchange is right for you? *Software* (February/March 2001) (www.softwaremag.com/archive/2001feb/SelectingB2B Exchange.html).
27. Nwana, H.S.; Ndumu, D.T.; Lee, L.C.; and Collis, J.C. Zeus: A toolkit for building distributed multi-agent systems. *Applied Artificial Intelligence Journal, 13*, 1 (January 1999), 129–185.
28. Omicini, A. SODA : Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M.J. Wooldridge (eds.), *Agent-Oriented Software Engineering II: First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer-Verlag, 2000, pp. 185–193.
29. Parunak, H.V.D. Agents in overalls: Experiences and issues in the development and deployment of industrial agent-based systems. *International Journal of Cooperative Information Systems, 9*, 3 (September 2000), 209–227.
30. Parunak, V.; Sauter, J.; and Clark, S. Toward the specification and design

of industrial synthetic ecosystems. In M.P. Singh, A. Rao, and M.J. Wooldridge (eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages*. Berlin: Springer-Verlag, 1998, pp. 45–59.
31. Ricci, A.; Denti, E.; and Omicini, A. Agent coordination infrastructures for virtual enterprises and workflow management. In M. Klush and F. Zambonelli (eds.), *Cooperative Information Agents V: Proceedings of the Fifth International Workshop (CIA 2001)*. Heidelberg: Springer-Verlag, 2001, pp. 235–246.
32. Romanosky, S. Security design patterns v 1.4. *Security Focus* (2001) (www.securityfocus.com/guest/9793/).
33. Sabater, J.; Sierra, C.; Parsons, S.; and Jennings, N.R. Engineering executable agents using multi-context systems. In N.R. Jennings and Y. Lespérance (eds.), *Intelligent Agents VI: Agent Theories, Architectures, and Languages: Sixth International Workshop (ATAL'99)*. Berlin: Springer-Verlag, 2000, pp. 131–148.
34. Scott, W.R. *Organizations: Rational, Natural and Open Systems*. New York: Prentice-Hall International, 1992.
35. Shen, W., and Norrie, D.H. Agent-based systems for intelligent manufacturing: A state-of-the-art survey. *Knowledge and Information Systems, 1*, 2 (May 1999), 129–156.
36. Shepherdson, J.W.; Thompson, S.G.; and Odgers, B. Decentralised workflows and software agents. *BT Technology Journal, 17*, 4 (October 1999), 65–71.
37. Singh, M.P., and Huhns, M.N. Multi-agent systems for workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management, 8*, 2 (June 1999), 105–117.
38. So, Y.-P., and Durfee, E.H. Designing organizations for computational agents. In M.J. Prietula, K.M. Carley, and L. Gasser (eds.), *Simulating Organizations: Computational Models of Institutions and Groups*. Menlo Park, CA: AAAI Press, 1998, pp. 47–64.
39. Sparkman, C.H.; DeLoach, S.A.; and Self, A.L. Automated derivation of complex agent architectures from analysis specifications. In M.J. Wooldridge, G. Weis, and P. Ciancarini (eds.), *Agent-Oriented Software Engineering II: Second International Workshop (AOSE 2001), Montreal, Canada*. Berlin: Springer-Verlag, 2001, pp. 278–296.
40. Tambe, M.; Pynadath, D.V.; and Chauvat, N. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing, 4*, 2 (March/April 2000), 65–73.
41. Thompson, S.G., and Odgers, B.R. Collaborative personal agents for team working. In D. Kitchin, R. Aylett, L. McCluskey, J. Porteous, and S. Steel (eds.), *Proceedings of 2000 Artificial Intelligence and Simulation of Behavior (AISB) Symposium*. Birmingham, UK, 2000, pp. 49–61.
42. Wood, M., and DeLoach, S.A. An overview of the multi-agent systems engineering methodology. In P. Ciancarini and M.J. Wooldridge (eds.), *Agent-Oriented Software Engineering II: First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer-Verlag, 2000, pp. 207–221.
43. Wooldridge, M.; Jennings, N.R.; and Kinny, D. The Gaia methodology for agent-oriented analysis and design. *International Journal of Autonomous Agents and Multi-Agent Systems, 3*, 3 (September 2000), 285–312.

44. Yoder, J., and Barcalow, J. Architectural patterns for enabling application security. The Fourth Pattern Languages of Programming Conference. Technical Report WUCS-97–34. Allerton Park, IL: Washington University, September 1997 (jerry.cs.uiuc.edu/~plop/plop97/Proceedings/yoder.pdf).
45. Yu, L., and Schmid, B.F. A conceptual framework for agent oriented and role based workflow modelling. *CaiSE Workshop Conference on Agent Oriented Information Systems (AOIS´99)*. Heidelberg: AOIS.org, May 1999 (www.mcm.unisg.ch/people/lyu/yuAOIS99.pdf).
46. Zambonelli, F.; Jennings, N.R.; and Wooldridge, M. Organizational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M.J. Wooldridge (eds.), *Agent-Oriented Software Engineering II: First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer-Verlag, 2000, pp. 235–250.

ANTHONY KARAGEORGOS (karageorgos@acm.org) received a B.Sc. in applied mathematics from the Aristotle University of Thessaloniki in 1991 and an M.Sc. in computer science from the University of Essex in 1994. Since then, he has been a lecturer on information technology and a simulation analyst and consultant. He is currently doing Ph.D. research at the University of Manchester Institute of Science and Technology, and his project is sponsored by BT Labs. His research interests are in agent-oriented software engineering, agent-based business applications, and computer simulation. His Ph.D. work has so far resulted in one patent application and a number of academic awards and research papers. He is a member of the ACM, IEEE, and SCS.

SIMON THOMPSON (simon.2.thompson@bt.com) joined the Intelligent Business Systems Research Group at BT in December 1997 from the University of Portsmouth, where he was a research assistant. He has a B.Sc. in computer science from the University of Hertfordshire and a Ph.D. from the University of Portsmouth. His Ph.D. thesis was concerned with structural risk minimization in machine learning. Since 1997, Dr. Thompson has been working on developing commercial applications of intelligent agents. He is also heavily involved in the development of agent systems and is currently the maintainer of the Zeus Agent Toolkit, an open-source project developed by BT and now widely adopted as a leading tool for developing collaborative agent systems. He has authored papers on subjects ranging from machine learning, genetic algorithms, and multiagent systems to business process representation and abstraction. His areas of expertise are Java, distributed computing, intelligent agents, collaborative agents, machine learning, open-source development, community development processes, XML directories, and agent-development toolkits and environments.

NIKOLAY MEHANDJIEV (nikolay@computer.org) is a lecturer in the Department of Computation at the University of Manchester Institute of Science and Technology. He graduated in computer science from Sofia Technical University in 1990 and completed his Ph.D. on "User Enhanceability for Information Systems Through Visual Programming" at the University of Hull. His research interests focus on systems-development models, languages, and tools in the context of rapidly changing business. He has active research projects in the areas of visual languages for end-user development, agent-based architectures, flexible information systems, and multiperspective business modeling.