The Knowledge Engineering Review, Vol. 17:4, 331–358. © 2002, Cambridge University Press
DOI: 10.1017/S0269888903000572 Printed in the United Kingdom

RAMASD: a semi-automatic method for designing agent organisations*

ANTHONY KARAGEORGOS,[1] NIKOLAY MEHANDJIEV[2] and SIMON THOMPSON[3]

[1] *Dept. of Computation, UMIST, Manchester M60 1QD, UK; e-mail: karageorgos@acm.org*
[2] *Dept. of Computation, UMIST, Manchester M60 1QD, UK; e-mail: mehandjiev@acm.org*
[3] *Team Leader, Intelligent Agents Research Group BT Exact Technologies; E-mail: simon.2.thompson@bt.com*

Abstract

Designing realistic multi-agent systems is a complex process, which involves specifying not only the functionality of individual agents, but also the authority relationships and lines of communication existing among them. In other words, designing a multi-agent system refers to designing an agent organisation. Existing methodologies follow a wide variety of approaches to designing agent organisations, but they do not provide adequate support for the decisions involved in moving from analysis to design. Instead, they require designers to make ad hoc design decisions while working at a low level of abstraction.

We have developed RAMASD (Role Algebraic Multi-Agent System Design), a method for semi-automatic design of agent organisations based on the concept of role models as first-class design constructs. Role models represent agent behaviour, and the design of the agent system is done by systematically allocating roles to agents. The core of this method is a formal model of basic relations between roles, which we call *role algebra.* The semantics of this role-relationships model are formally defined using a two-sorted algebra.

In this paper, we review existing agent system design methodologies to highlight areas where further work is required, describe how our method can address some of the outstanding issues and demonstrate its application to a case study involving telephone repair service teams.

1 Introduction

Intelligent agents have a generally accepted property of autonomy, yet we commonly perceive them as operating together with other agents within a system, which we refer to as a *multi-agent system* or an *agent organisation* (Gasser, 2001). Agent organisations have properties, norms and authority relationship structures which transcend individual agents and help them coordinate their actions, regardless of whether the agents are collaborating to achieve a common goal, or acting as competitors, for example when bidding at auctions. The set of properties, norms and structures form the *organisational setting*, which contains lines of communication, lines of authority, rules and patterns of interaction and descriptions of expected behaviour. The agent organisation provides the coordination framework within which individual agents operate (Ferber & Gutknecht, 1998; So & Durfee, 1998; Wooldridge *et al.*, 2000).

* This work has been supported by BT under a grant from the office of the Chief Technologist (No. ML816801/MH354166).

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

Creating an agent organisation is a complex process, and the correct choices at each stage of this process cannot be easily prescribed due to the sheer number of factors which influence such choices in a non-trivial and dynamic manner. This also means that a standard optimal organisation for all circumstances cannot be created (Scott, 1992; So & Durfee, 1998). As a result, most methodologies for engineering multi-agent systems have left the rules for designing agent organisations vague and informal, relying on the creativity and the intuition of the human designer. This, however, can be a serious drawback when designing large and complex real-world agent systems because the scale of the problem creates numerous possibilities for inconsistent decisions, especially when designers work in teams (Grundy, 2000).

One way to counteract the complexity involved in designing agent organisations is to allow agent-system designers to work at a high abstraction level (Kendall, 1999a; Omicini, 2000) by considering both functional and organisational abstractions as first-class design constructs. This also opens up the possibility of reusing domain analysis knowledge during design. Indeed, given appropriate formally defined semantics of these constructs, it becomes feasible to support designers by automatically generating solutions which conform to the constraints and organisational relationships defined during analysis.

These ideas are in the core of RAMASD (Role Algebraic Multi-Agent System Design), our method for high-level design of agent organisations, described in this paper. We focus on the concept of role as "prototypical type of behaviour" (Gasser, 2001), which members of the organisation commit to, expect from others and instantiate by their actions within the organisation. Roles and their interactions in a particular context form *role models*. Our method uses roles and role models as basic building blocks when designing agent organisations, and formalises a basic set of role relations which concern the composition of roles and their allocation to agents. This enables the creation of a tool which can support the designer of agent organisations in the following manner: (a) the designer specifies (or reuses) a set of role models applicable to the problem at hand, where each model contains formalised rules and constraints regarding the ways roles can be assigned to agents; and (b) the tool automatically designs an agent organisation by allocating roles to agents in a manner consistent with these rules. The designer then can accept this design or he can modify the role specification and request an alternative design solution to be produced. To test the method, we have created such a tool – an experimental version of the Zeus agent building toolkit. The original toolkit is described in Nwana *et al.* (1999).

This paper presents our method in the context of existing methodologies for engineering agent organisations. The review of current methodologies is presented in the next section. Section 3 then contains an overview of role modelling, which underpins our semi-automatic method for agent organisation design. In Section 4 we present the role algebra, a formal model of role relations which is the main enabler of our approach. We define its semantics using a two-sorted algebra. The overall method of semi-automatic design of agent organisations is then presented in Section 5. The use of the role algebra to design an agent organisation is illustrated in Section 6 by an example based on a case study of telephone repair service teams. In Section 7 we present discussiong on the limitations of RAMASD. Finally, directions for future work are presented in Section 8.

2 Methodologies for engineering multi-agent systems

Early research prototypes of agent-based systems were built in an ad hoc manner. However, the need to engineer agent systems solving real-world problems has given rise to a number of systematic methodologies for agent system analysis and design such as MESSAGE (Evans, 2000), Gaia (Wooldridge *et al.*, 2000) and SODA (Omicini, 2000). These methodologies range from simple strategies to comprehensive methodologies (Iglesias *et al.*, 1999; Shen & Norrie, 1999; Parunak, 2000; Jennings & Wooldridge, 2001), and they support different software engineering life-cycle stages such as requirements analysis, design, code generation and testing. We are interested in the stage of designing agent organisations. The manner in which different methodologies support this stage can be used to classify them in several broad groups. This classification will be used to structure our discussion of the strengths and weaknesses of each one.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

2.1 Classification of agent-based system engineering methodologies

Extending the classification introduced by Wooldridge (in Wooldridge & Ciancarini, 2000), we propose a classification scheme for agent-based system engineering methodologies, which is summarised in Figure 1. The main levels of classification are:

Level 1: the mode of application The methodologies where agent organisation is designed once before the deployment of the agent-based system are classified as *static*; those where design decisions are taken after the deployment at run time are classified as *dynamic*.

Level 2: the degree of formality present in each methodology This divides methodologies into *formal*, *informal* and *ad hoc*, based on the techniques used to specify the behaviour of each agent. The *ad hoc* methodologies do not contain systematic techniques for analysis and design. *Formal* methodologies use formal techniques for specification and verification of agent behaviour. *Informal* methodologies use structured and systematic techniques, which do not use formal logic and rigorous criteria for taking design decisions but are instead based on heuristic rules and guidelines.

Level 3: the relationship with conventional methodologies The agent methodologies are seen as related to several types of conventional software engineering methodology, from which they mostly originate.

The types of methodology at level 3 are as follows:

- methodologies aligned with *object-oriented software engineering (OOSE) methodologies*, for example Agent/UML (Bauer *et al.*, 2000), Gaia (Wooldridge *et al.*, 2000) and MESSAGE/UML (Caire *et al.*, 2001);
- extensions to *knowledge engineering methodologies*, for example SODA (Omicini, 2000), MAS-CommonKADS (Iglesias *et al.*, 1998);
- methodologies based on *information systems methodologies*, e.g. TROPOS (Bresciani *et al.*, 2001a);
- methodologies highly coupled with specific *agent-based system building toolkits*, e.g. Zeus (Nwana *et al.*, 1999).

Notably, some *OOSE-aligned* methodologies focus on adapting existing software engineering standards and notations to agent-based systems engineering, for example Agent/UML (Bauer *et al.*, 2000), whilst others combine agent theories with existing software engineering methodologies, for example Gaia (Wooldridge *et al.*, 2000).

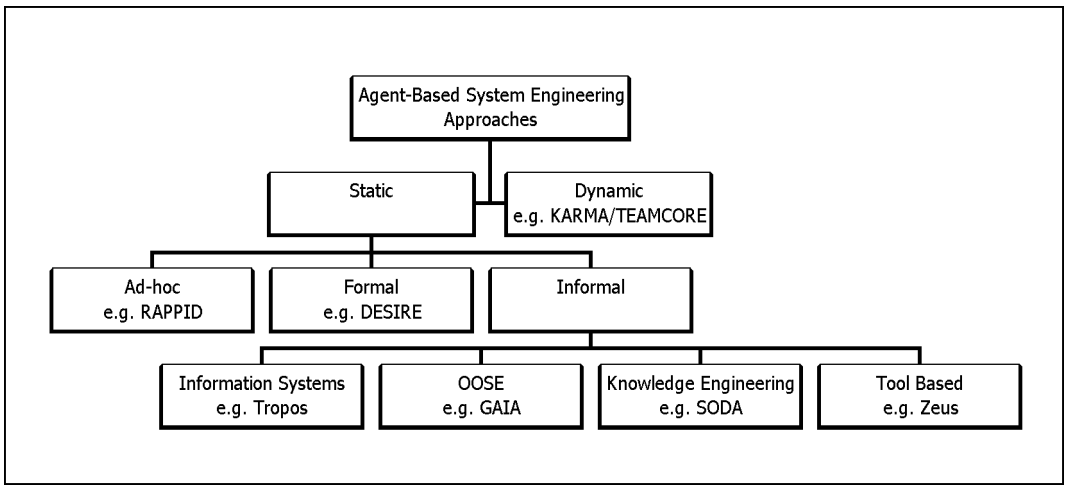


Figure 1 Classification of agent-based system engineering methodologies

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

Having classified existing methodologies for engineering agent organisations, we will now discuss a representative methodology for each of the main groups to identify how they support design activities.

2.2 *Dynamic methodologies*

Dynamic methodologies focus on continuous redesign of the agent organisation after the deployment of the system (at run time), thus allowing adaptation to dynamic changes in requirements and environment. In such systems, agents change their organisational relations and behaviour as a response to environmental stimuli or requirements changes. This is called *self-organisation* (Ishida *et al.*, 1992; Gerber, 1998; Malvile & Bourdon, 1998; So & Durfee, 1998). Self-organisation relies on constant application of re-organisation primitives to optimise the behaviour of the system, for example the *Organisation Self-Design* (OSD) framework (Ishida *et al.*, 1992) uses the primitives of composition and decomposition. Decomposition involves division of an agent into two and can be performed to respond to overwhelming environmental demands. Composition merges two agents into one and can be useful when communication overheads between the two agents are too high. The initial organisation starts with one agent containing all domain and organisational knowledge. Other approaches to self-organisation reassign roles and responsibilities to different organisational nodes (Pattison *et al.*, 1987).

A representative example of this class is KARMA/TEAMCORE (Tambe *et al.*, 2000a), based on dynamic rapid prototyping of agent organisations. It claims to reduce development effort by enabling automatic procurement and integration of existing agents into agent organisations based on a set of application requirements. In this way, even inexperienced users would be able to build large agent organisations for real-world applications. Using automatic procurement and integration, agent organisations can be constantly updated to reflect changes in requirements and the environment.

KARMA/TEAMCORE consists of two subsystems, KARMA and TEAMCORE. KARMA (Knowledgeable Agent Resources Manager Assistant) assists designers in the creation, specification and monitoring of an agent organisation. Each organisation is structured as a hierarchy of teams. Each team is allocated goals decomposed into sub-goals using plans. A *team program* specifies the organisation hierarchy, the plan hierarchy and the capabilities of agents that could execute those plans. The organisation hierarchy consists of roles, which are assigned to plans. All specifications are written in the STEAM specification language (Tambe, 1997). This is called *team-oriented programming*. It aims to abstract away details of coordination, so that designers do not have to write large numbers of coordination plans.

Once the organisation is specified, KARMA searches for agents that match the requirements of the organisation and can play the roles of the specified organisation. It compiles a list of suitable agents together with their properties. From this list, the designer then manually selects the desired agents to include in the organisation. Finally, KARMA monitors the agent organisation to diagnose failures and to evaluate agent performance to be used in forthcoming reorganisations.

The second subsystem, TEAMCORE, focuses on reliable execution of tasks. Robust teamwork among agents is seen as the means for reliability, where agents cover for each other's execution failures and share key information. Interoperability between agents is ensured by a wrapper.

Overall, KARMA/TEAMCORE supports designers of agent organisations in a comprehensive manner. It allows them to work at a high abstraction level using organisational concepts such as roles and teams as first-class design constructs. It also provides automatic support for design heuristics using its specification language STEAM.

One major weaknesses of KARMA/TEAMCORE is that it assumes all necessary agents exist in the cyberspace. In this sense it is an approach for procuring individual agents, not for building them.

The second major weakness is common to all dynamic methodologies. Dynamic reorganisation consumes a lot of run-time system resources as the agents in the agent-based system need to communicate frequently in order to carry out reorganisation acts (Gerber, 1998; So & Durfee, 1998; Tambe *et al.*, 2000a). Also, the time taken by reorganisation signals to propagate through the system

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

may cause the system to behave in a non-predictable manner, or the system performance may deteriorate (Lee *et al.*, 1998).

2.3 Ad hoc methodologies

Static methodologies avoid the problems associated with reorganising since the agent organisation is designed once at design time. They range in formality and structure from ad hoc to fully formal methodologies, and we will first discuss the ad hoc methodologies.

The philosophy underlying ad hoc methodologies is that design decisions should be simple and clear, motivated by agent system properties as autonomy and flexibility. Design in ad hoc methodologies is therefore based on generic guidelines aligned with properties of the agent metaphor. An example guideline is that each user should be paired with a software agent that will be able to act on the user's behalf. An agent is the most suitable software component here, since it can learn and adapt to its user.

RAPPID (Responsible Agents for Product-Process Integrated Design) (Parunak *et al.*, 1999) is a good example of ad hoc methodologies. It targets the domain of collaborative product design (Parunak *et al.*, 1997). In RAPPID, an agent is allocated to each user (designers, manufacturing engineers, marketing and support staff) and to each component of the design itself. Agents trade with one another to optimise design constraints, requirements and manufacturing alternatives.

RAPPID does not support the design of agent organisation in an explicit manner. It only provides a generic approach and guidelines about how the agents and the agent-based system should be built. Decisions about the exact system functionality are left to the designer, who makes them in an ad hoc manner.

This methodology, whilst undoubtedly easy to use for small systems, is clearly not applicable to systems of realistic complexity. Indeed, in complex systems it would be difficult for the designer to act in a consistent manner where the guidelines may not be completely applicable or under conflicting requirements, and this may lead to design errors. This lack of support for designers of agent organisations is common for ad hoc methodologies.

2.4 Formal methodologies

Formal methodologies for agent systems engineering were created to counteract the lack of rigour in ad hoc methodologies (Brazier *et al.*, 1997) and to thus help the application of agent systems in industry. They are used in three primary ways (Wooldridge & Ciancarini, 2000): for the *specification*, *systematic implementation* and *verification* of agent-based systems. Designing agent behaviour and organisation can be seen as belonging to the specification activities.

Specifying an agent system requires a theory describing an individual agent in terms of its structure and internal behaviour. Such a theory is called an *Agent Theory* (Wooldridge & Jennings, 1995), and is commonly based on some temporal modal logic framework such as the Cohen-Levesque *theory of intention* (Cohen & Levesque, 1990) and Rao-Georgeff *belief-desire-intention* model (Rao & Georgeff, 1995). The Cohen-Levesque model is based on two agent attributes: beliefs and goals. Other attributes, for example the notion of intention, are built from those. In contrast, the Rao-Georgeff model takes intentions as primitives, in addition to beliefs and goals.

A specification uses this framework to describe the desirable behaviour of a system in a non-ambiguous manner, and this is the starting point of every formal agent-based system design method, (e.g. Brazier *et al.*, 1997; Hilaire *et al.*, 2000).

Formal approaches to agent-based system design are often based on unrealistic assumptions, for example the *possible worlds assumption* (Wooldridge & Ciancarini, 2000), which impede the efficient mapping of specifications to appropriate implementations. DESIRE (Brazier *et al.*, 1997; Brazier *et al.*, 2001) is a formal methodology that does not suffer from this weakness. It can be used for conceptual specification, behavioural simulation and prototype generation of agent-based systems.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

DESIRE uses two types of model: the *intra-agent* model describes the expertise related to domain tasks, whilst the *inter-agent* model describes the expertise to perform and guide coordination, cooperation and social interaction among agents. The whole system and each individual agent are modelled as sets of interacting hierarchically structured components that represent task-solving units. A task hierarchy is constructed by recursively decomposing the overall task for the system.

The agent organisation in DESIRE is defined by composing primitive component models that are directly related to agent tasks. Reuse is also supported by using existing generic agent models to design a new, more specific agent model, and to specialise tasks (Brazier *et al.*, 1995). During design, components in a generic model are refined by (1) more detailed analysis of the tasks of which such components are comprised and/or (2) inclusion of specific domain knowledge.

Knowledge in DESIRE is represented at three levels: *conceptual*, *detailed* and *operational*, where the operational level is automatically generated from the detailed level. This is possible because of the formal specification language of DESIRE, which also enables a set of automatic tools for verifying and simulating the agent-based system including a graphical editor, a specification compiler that generates Prolog code and various debugging and monitoring tools. DESIRE has been used by a number of companies and research institutes (such as the chemical industry, the financial sector, the software industry, institutes for environmental studies) to develop operational systems for a number of complex tasks (including systems for diagnosis, design, routing, scheduling and planning).

The design process, however, cannot be automated because the agent components should be known before agent behaviour is specified. Furthermore, rules to support design can be modelled using the DESIRE specification language, but there is no support for this at the methodology level, and this needs to be done manually by the designer.

Different levels of abstraction are possible, but only in the task and component dimension; there is no explicit modelling of organisational constructs, collective behaviour and non-functional aspects.

In conclusion, formal methodologies allow automatic support for various activities in the life cycle of agent-based systems, for example validating specifications and simulating behaviour. Some design and implementation activities can also be automated within both static and dynamic methodologies, as in KARMA/TEAMCORE (Tambe *et al.*, 2000a) and Concurrent METATEM (Fisher & Wooldridge, 1997). The general approach of formal methodologies, where detailed agent specifications are automatically synthesised, is theoretically appealing, but it is limited in two important aspects (Jennings & Wooldridge, 2001): First, as the expressive power of the agent specification language grows, as for example in a language based on first-order calculus, the synthesis problem becomes intractable (Wooldridge & Ciancarini, 2000) since no algorithm guarantees a solution. Even at the lower levels of expressive power of propositional logic, the complexity of theorem-proving can be exponential and hence not practical for real design of agent organisations. Second, the application of formal notations based on mathematical theories is often found to be a difficult task by the average software engineer (Wooldridge & Ciancarini, 2000).

These two drawbacks are in the core of argument that at least some steps in methodologies for agent-based systems engineering should be left informal. This argument is supported by many agent researchers, for example Petrie (2000) and Jennings (Jennings & Wooldridge, 2001).

2.5 *Informal methodologies*

Informal methodologies for engineering agent systems express knowledge about the domain and the system using a number of models. For example, an interaction model can describe interactions that take place between agents and an organisational model can describe the organisational structure of the actors in the business organisation. A common characteristic in informal methodologies is that the design of agent-based systems cannot be automated to any extent and is carried out manually by the designers based on informal guidelines and their experience.

The majority of informal methodologies can be traced back to methodologies from object-oriented software engineering, knowledge engineering and information systems.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

2.5.1 Methodologies based on object-oriented software engineering

As mentioned earlier, these methodologies either start from the agent metaphor and try to adopt aspects from traditional object-oriented methodologies, e.g. Gaia (Wooldridge *et al.*, 2000), or specialise and extend object-oriented concepts so that they are applicable to agent-based system design, e.g. Agent/UML (Bauer *et al.*, 2000) and MASE (Wood & DeLoach, 2000).

Gaia is a methodology which supports the design of both individual agents and the agent organisation as a whole. It includes a number of analysis and design models. The construction of analysis models starts by identifying agent roles in the system, and proceeds to modelling interactions between these roles. A role in Gaia consists of four attributes: responsibilities, permissions, activities and protocols, which together describe roles as generalised patterns of individual behaviour and authority. Gaia has formal operators and templates for representing roles and their attributes; it also has schemas that can be used for the representation of interactions.

Once all the roles and their interactions are captured, the *design* process can start. Designers first map roles into *agent types*, and then create the right number of *agent instances* of each type. Then they determine the *services* needed to fulfil a role in one or several agents, and the final step is to create the *acquaintance model* for the representation of communication between the agents.

Gaia's strength is the simple yet comprehensible set of models, which makes its application quite straightforward. It is also not committed to any particular type agent architecture such as BDI or similar. Its applicability, however, is limited by its focus on cooperative agents acting towards a common goal within systems of no more than about 100 agents. It also produces high-level designs, which have to then be refined into implementations using traditional software engineering methodologies. Also, the way in which organisational settings are modelled is limited to the concept of roles. The lack of formal theoretical basis means that the design process cannot be automated and possibilities for tool support are limited.

2.5.2 Methodologies based on information systems engineering

Methodologies based on information systems engineering (Elammari & Lalonde, 1999; Wagner, 2000; Bresciani *et al.*, 2001a) target organisational systems in areas such as e-business (Bartelt & Lamersdorf, 2000) and e-services (Durante *et al.*, 2000), where semantics of transactions can only be captured if the specific actors behind events and actions are explicitly represented in the information system, in addition to passive business objects (Wagner, 2000). An ontological distinction is thus made between active entities (agents) and passive entities (objects). Dynamically evolving parts of the system are modelled using agents and agent relationships, and agent-based conceptual models are then transformed into traditional software engineering models and designs, for example in the AOR method (Wagner, 2000).

An important perceived advantage of information systems-based methodologies is the correspondence between models of the system itself and its organisational environment (Castro *et al.*, 2000). In Tropos (Castro *et al.*, 2000; Giunchiglia *et al.*, 2002), for example, the software system is represented as one or more actors, which participate in a strategic dependency model, along with other actors from the system's operational environment.

Tropos is a comprehensive methodology, which covers the whole software life cycle including two design stages: system-wide architectural design and detailed design for each individual agent. Tropos is integrated with the agent-building toolkit JACK (Maisano, 2002) in terms of architecture and implementation concepts, and therefore Tropos models can be implemented in a rather straightforward manner.

The support for designing agent organisations is nevertheless quite poor, for example Tropos does not allow designers to work with organisational settings and collective behaviour as design constructs. It also does not provide support for design rules and for reusing design knowledge.

2.5.3 Methodologies based on knowledge engineering

Knowledge engineering methodologies are often considered very appropriate for agent-based systems because of agents' knowledge-intensive nature and cognitive behaviour (Iglesias *et al.*, 1999). This

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

allows reuse of existing knowledge engineering tools and ontology libraries. Conventional knowledge engineering methodologies, however, take a centralised view and do not address the distributed or social aspects of the agents, nor their reflective and goal-oriented attitudes. To alleviate these problems, a number of agent-based system engineering methodologies such as CoMoMAS (Glaser, 1997) and MAS-CommonKADS (Iglesias *et al.*, 1998) extend knowledge engineering methodologies such as CommonKADS (Schreiber *et al.*, 2000). The focus of all such methodologies is the building of a number of separate models, but they do not address the links between these models and the implementation of the agent-based system.

Societies in Open and Distributed Agent spaces (SODA) (Omicini, 2000) is a good representative of this class of methodology. The emphasis of SODA is on providing agent-based system engineers with specific, ad hoc modelling abstractions and tools, which focus on inter-agent aspects such as *societies* and *agent spaces* (an agent space is the environment where agents live and interact). Both societies and agent spaces are first-class system components, and are modelled using specific abstractions so that they can be taken into account at every stage of the engineering process.

In particular, SODA exploits coordination models, languages and infrastructures to address social issues, by showing how to choose a coordination model, how to exploit it to design social laws, how to embed them into a coordination medium and how to build a suitable social infrastructure based on coordination services. Coordination models are used to decide which part of social behaviour should be embedded in the individual agent's behaviour, and which should be realised using appropriate social structures

SODA involves three analysis models (*role*, *resource* and *interaction models*) and three design models (*agent*, *society* and *environment models*):

- The *role model* contains roles which have tasks assigned to them. Tasks can be individual or social, and are expressed in terms of the responsibilities they involve, the competences they require and the resources they depend upon. Roles belong to social groups which carry out social tasks.
- The *resource model* includes the abstract resources that need to be used by the agent-based application. Each resource represents an actor in the agent environment and provides a number of *services* to the agent application, for example recording data, querying a sensor or verifying an agent identity.
- The *interaction model* includes definitions of *interaction protocols* (for roles and resources) and of *interaction rules* (for social groups). Interaction rules determine behaviour that all members in a social group should exhibit. A social group would also have a set of social roles, its own social task and specific permissions to access system resources.
- The *agent model* is the first of design models. It contains agent components, which are defined as groups of application and social roles.
- The *society model* describes agent societies, which consist of a set of social groups and their roles, tasks and permissions. The main issue in the society model is how to design interaction rules so as to make societies to accomplish their social tasks.
- In the *environment model*, resources from the resource model are mapped to infrastructure classes. The model also includes a topological description of the distribution of infrastructure classes. Other topological abstractions can also be used, for example places, domains and gateways (as defined by the TuCSoN model for the coordination of Internet agents (Cremonini *et al.*, 1999)).

SODA targets open (e.g. Internet-based) agent-based systems (Omicini, 2000) and its underlying philosophy is that insights from the area of coordination models can be used in order to make traditional agent-based system engineering approaches, such as Gaia (Wooldridge *et al.*, 2000), more suitable for developing Internet-based applications (Zambonelli *et al.*, 2001a).

2.5.4 Tool-based methodologies

A large number of agent development toolkits are currently available, the majority of which are in the public domain (Reticular Systems Inc., 2002). Standardisation efforts have made significant progress in terms of unified agent architectures (Dale & Mamdani, 2001; McCabe *et al.*, 2002), but there are

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

still many differences and incompatibilities between tools, for example regarding agent cognitive models and implementation languages.

Every agent development toolkit is tailored to a specific agent theory and a set of implementation assumptions. For example, Zeus (Nwana *et al.*, 1999) uses BDI agents, which know facts, carry out tasks and are driven by goals, while Voyager (Objectspace Inc., 1997) supports only reactive agents. Another example is that in Jade (Bellifemine *et al.*, 1999) a full agent life cycle including creation and removal can be specified while in the current version of Zeus there is no such provision and agents are assumed to execute infinitely.

The Zeus agent development methodology consists of analysis, design and realisation activities, as well as *run-time support* facilities that enable the developer to debug and analyse their implementations. The recommended approach to analysis is *role modelling* (Collins & Ndumu, 1999). The design then involves linking roles to agents by matching role responsibilities (or tasks) to agent characteristics. This creates a conceptual-level design, which is converted to working agents during the *realisation* stage.

The Zeus agent development methodology is closely related to the Zeus agent development toolkit (Nwana *et al.*, 1999). The toolkit provides an extensive set of editors for different aspects of the system, such as organisational relations, coordination and negotiation models. Once detailed agent-based system designs are constructed, they can be transformed to Java source code within the *Agent Generator* tool.

The Zeus toolkit and the related development methodology are freely available, and provide a comfortable and flexible framework for the development of agent-based systems. They are, however, limited to the BDI agent architecture. The design process is informal and cannot therefore be automated, nor can designers work at different levels of abstraction.

2.5.5 Summary of informal methodologies

Informal methodologies require the designer to address most of the system complexity based on creativity and intuition alone. This can be a serious problem in engineering large, real-world agent-based systems. Furthermore, informal methodologies lack a semantic framework and notation that would allow any verification of the design decisions. This may result in error-prone designs and it does not allow any automation of the design process. This contrasts with the view supported by many authors in agent-based system design (Tambe *et al.*, 2000a; Sparkman *et al.*, 2001) and software design in general (Lowry & McCartney, 1991), that to reduce development effort the design process must be automated to a certain extent.

2.6 Issues in supporting the design of realistic agent organisations

The review of existing agent-based system engineering methodologies has revealed weaknesses in the support they provide for the design of agent organisations. They do not provide formal support for design decisions and do not allow designers to work with organisational settings and collective behaviour as first-class design constructs. A method to effectively support designers should provide the features discussed below.

2.6.1 Systematic support for constructing large agent system design models from the analysis models

The main drawback of existing methodologies such as Tropos is that after a certain point the design decisions are left solely to the creativity and the intuition of the designer. Even when a methodology includes some guidance as to how to transform the analysis models to design models, the steps involved are not specified in a sufficient level of detail, and their interpretation is left to the designers.

This lack of detail also prevents the building of a software tool, which can provide automated support for design. To enable effective support for designers, we therefore need to provide a formally specified method for transforming analysis models into design models. This view is similar to that described in Sparkman *et al*. (2001) where agent architectures are automatically derived from analysis specifications.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

2.6.2 Considering non-functional requirements at design time

Existing dynamic methodologies use run-time reorganisation to reflect non-functional requirements such as speed and memory limitations. To improve the system stability and to avoid performance losses associated with this type of run-time reorganisation, we should aim to optimise, as much as possible, the agent organisation against its non-functional requirements at design time.

Considering non-functional requirements before actually deploying a multi-agent system is therefore necessary for effective support of designing agent organisations of realistic complexity. This is confirmed by some methodologies which attempt to model and study the performance of a system before its actual deployment (Scott, 1992; Parunak *et al.*, 1998).

2.6.3 Organisational settings as reuseable first-class design constructs

A powerful approach to reduce complexity in agent organisation design is to increase the level of abstraction by considering organisational settings as first-class design constructs. We can then reduce complexity further by reusing organisational settings from different applications or domains. This vision concurs with the ideas of Zambonelli and Jennings (Zambonelli *et al.*, 2000).

Turning such a vision into a reality requires a suitably formal model of organisational settings and a rigorous technique for combining both organisational and application design decisions in a systematic manner.

3 Roles and role models

We consider the concepts of *role*, *role relationship* and *role model* as fundamental enablers of our method for designing agent organisations, which addresses the issues described above. This is achieved by extending an existing definition of *role* to allow for modelling of non-functional requirements, and by introducing a systematic technique for transforming *role models* into agent organisation designs, thus enabling a semi-automated design process.

3.1 Roles as behavioural abstraction in organisations

Roles are basic building blocks for a number of modelling approaches. For example, roles are used in organisational theory (Scott, 1992) to represent positions and responsibilities in human organisations. Roles are also used in conventional software engineering (Andersen, 1997). Furthermore, roles are considered particularly suitable for modelling the behaviour of software agents due to their ability to represent generalised behaviour in organisational context (Kendall, 1999a). Roles in multi-agent systems are defined in a manner similar to organisational roles referring to a position and a set of responsibilities in an organisation (Ferber & Gutknecht, 1998). Some additional characteristics are also included in agent roles, such as capabilities to conduct planning, coordination and negotiation (Kendall, 1999a).

Some multi-agent system engineering methodologies use roles as basic behavioural abstractions. These methodologies acknowledge the need to identify and characterise relations between roles (Andersen, 1997; Kendall, 1999a), but only a very few of them consider the consequences of role relations on the design of multi-agent systems (e.g. Kendall, 1999a). This is partly due to lack of formal foundations of role relationships.

In our method, we recognise this deficiency and propose formal algebraic definitions for a set of role relations that affect agent organisation design. To identify the set of role relations, we have used the *role theory* (Biddle, 1979). Role theory studies human organisations, and sees people as appointed to roles within an organisation, where a role is a representation of behaviour. This behaviour is characterised by *authorities* describing things that can be done and *responsibilities* describing things that must be done. For example, directors, help-desk staff, developers and test engineers are all associated with job descriptions specifying their responsibilities in the organisation. Organisational goals, policies and procedures further determine their rights and duties within the departments, projects or groups of which they are members.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

Role theory emphasises *relations* that may exist between roles. For example, an examiner cannot be a student at the same time and therefore appointing these roles to a person at the same time results to inconsistency. Role relations can be complex. For example, a university staff member who is also a private consultant may have conflicting interests. In this case, appointing these roles to the same person is possible but it would require appropriate mechanisms to resolve the conflicting behaviour.

Apart from role theory, the fundamental position of role as a prototypical behavioural type at the centre of each organisation is also accepted by theoreticians in the field of distributed artificial intelligence (Gasser, 2001).

3.2 Role characteristics

Following Kendall (1999a), a role is defined as a position in an agent-based system associated with a set of *characteristics*. Along the lines of role theory (Biddle & Thomas, 1979) roles describe some particular expected behaviour within some *social context*. This expectation can be linked to an agent within the social system, and we then say that the agent *owns* the role. When the agent realises the behaviour represented by a role, then we say that the agent *plays* that role. In agent-based systems roles represent a pragmatic view of agent behaviours, for example an agent-based system is considered to include a specific number of roles at a given time, each one consuming system resources and contributing to the overall behaviour of the system.

In our method each role is associated with five types of characteristic (Table 1): role model, goals/responsibilities, tasks, capabilities/privileges and performance variables.

Role models represent collections of roles and their interactions, and are described in detail below.

Roles have various *responsibilities* and goals which they aim to achieve. This follows the consensus in role-based modelling of social systems (e.g. Biddle & Thomas, 1979 and Kendall, 1999a) that the behaviour of social entities is affected by the goals the entity tries to achieve and by the duties the entity has within the social system. Since roles represent behaviours in certain contexts, they are associated with specific duties that need to be carried out and with goals that need to be achieved in those contexts.

Role behaviour is externalised by carrying out certain tasks. Tasks correspond to actions that social entities take towards fulfilling their duties and achieving their goals. In carrying out tasks, roles normally need to interact with other roles, which are their *collaborators*. Interaction normally takes place by direct exchange of messages according to interaction protocols. It must be noted that not all roles interact with each other in a role model. In the extreme case, we can have a role model consisting of only one role interacting only with passive resources and the environment. For example, this is the case when we have an agent simply handling the temperature valve of a central-heating unit. Such an agent will be playing only one role, which will be monitoring the environment for changes in the temperature, and its only task will be to operate the valve accordingly.

Capabilities or *privileges* refer to properties that enable or facilitate a role to achieve its goals and fulfil its responsibilities. Examples of capabilities or privileges include learning, inferencing and communicating. This view is similar to that of role theory where role functions – particular aspects of role behaviour – have characteristic effects on the social system in connection with the goals of roles.

Each role characteristic includes a set of *attributes*. Attributes represent different aspects of a characteristic property of role behaviour and can take both numeric and non-numeric values. For example, a characteristic of a role could be its capability to negotiate. The negotiation characteristic can have many attributes including the name of negotiation strategy that is followed and maximum and minimum bid values.

In order for roles to pragmatically represent behaviour in an application domain, they need to model issues relevant to non-functional aspects in that domain. Therefore the above role definition is extended to include *performance variables*. Performance variables are parameters whose value defines the run-time behaviour represented by a role. For example, if the behaviour a role represents requires using some resource like memory, the resource capacity can be modelled by a performance variable. Performance variables can also be defined at an agent level. In that case, their value is a function of

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

the values of the respective performance variables of all roles the agent is capable of playing. This allows us to apply design heuristics by imposing constraints on the values of the agent performance variables that must be observed when allocating roles to agents. This is illustrated in the example given in Section 6.

3.3 Role models

A collection of roles and their interactions constitutes a *role model*. A role model represents the behaviour required to carry out some process[1] in the system. An agent application normally consists of more than one activity and hence it will involve more than one role model. Role models that occur frequently in some application domain are called *role interaction patterns*.

In a role model, roles can be *specialised* in a manner similar to inheritance in object orientation (Andersen, 1997; Kendall, 1999a; Biddle & Thomas, 1979). Specialised roles represent additional behaviour on top of the original role behaviour in a manner similar to inheritance in object-oriented systems.

Role models can be used to represent reoccurring complex behaviour based on multiple points of interaction. Therefore they are considered to be first-class design constructs, that is, entities that can be instantiated and given identity. This property is fundamental regarding addressing the open issues raised in Section 2.6. This is because role models can be used to describe both functional and non-functional aspects of the application behaviour as well as organisational 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 organisation refers to selecting and instantiating suitable application and organisational role models.

4 Role algebra: a formal model of role relations

Designing an agent organisation in our method refers to allocating roles to agents. Allocating a number of roles to an agent is semantically equivalent to merging a number of roles into a single composite role played by the agent, and we call this activity *role composition*. In role composition, we have to take into account that roles may semantically constrain each 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. Furthermore, the way that role characteristics and their attributes are merged may be bound to various constraints. 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. To enable automatic software support for the process of role composition, we need to formally specify the set of role relations on which such constraints are based. Such a formal model, termed *role algebra*, is defined in this section.

4.1 Semantics of the role algebra

To describe the semantics of role relations we represent an agent organisation by a two-sorted algebra (Figure 2). The algebra includes two sorts, A representing agents and R representing roles, and two auxiliary relations, *Has* and *Plays*, representing role allocation.

In modelling agent behaviour using roles, we consider two possible modes of role allocation: At mode one, the behaviour specified by a role is allocated to an agent. The agent is aware of the details of the particular behaviour but it does not necessarily actively demonstrate it. For example, in a presidential election all candidates generally know how to act like presidents and hence they are aware of the presidential role. We can represent this as a relation $Has: A \rightarrow R$, which maps agents to 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*.

[1] Process in this context will represent the whole causal sequence of events and actions caused by one triggering event, and will correspond to the UML concept of "use case".

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

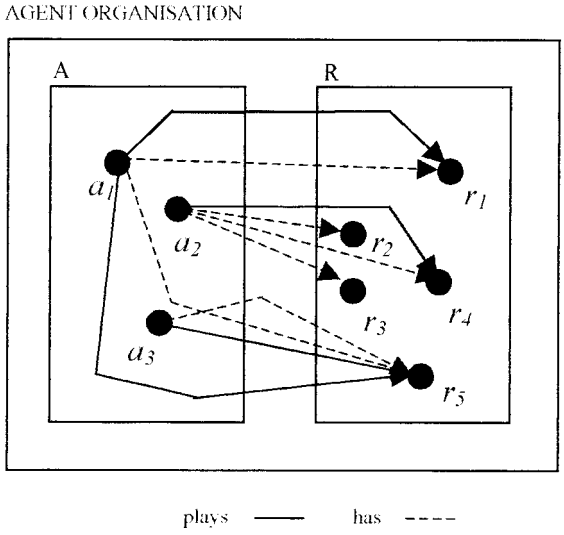


Figure 2 Semantics of role relations

At mode two, the behaviour represented by a role is not only known to but also actively demonstrated by an agent. In the presidential election example given above only the elected candidate will actually act like president. We denote this using another relation *Plays*: $A \rightarrow R$, which maps agents to roles again. 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 $\{a\} \subseteq A$ to the relation *Plays*.

We have to note that $a.plays \subseteq a.has$. In other words, an agent can have allocated roles and not actively demonstrate the behaviour represented by these roles. An example of this will be given in Section 6.

4.2 Role relations

Based on role theory (Biddle, 1979) and on case studies of human activity systems (e.g. Stark *et al.*, 2001), we have identified six binary relations as fundamental for role composition. For any $r_1, r_2 \in$ R (R is a set of roles), the binary relationships discussed below may hold.

4.2.1 Equals

The relation $Equals \subseteq R \times R$ *denoted by* '*eq*' refers to role equality. By r_1 eq r_2 we would mean that r_1 and r_2 describe exactly the same behaviour. 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 one role also plays the other at the same time, and the agent owning one role also owns the other. In the two-sorted algebra,

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

Thus defined, the relation $Equals \subseteq R \times R$ is an equivalence relation since it is reflexive, symmetric and transitive:

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

4.2.2 Excludes

The relation $Excludes \subseteq R \times R$ will be denoted by '*not*'. By r_1 *not* r_2 we mean that r_1 and r_2 cannot be assigned to the same agent simultaneously:

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

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

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 – if it would then no agent would ever play it. Therefore the relation $Excludes \subseteq R \times R$ is anti-reflexive and symmetric:

(a) $\forall r : R(\neg(r \; not \; r))$
(b) $\forall(r_1, r_2) : R \times R \; (r_1 \; not \; r_2 \Rightarrow r_2 \; not \; r_1)$

4.2.3 Contains

The relation $Contains \subseteq R \times R$ will be denoted by '*in*'. By r_1 in r_2 we mean that r_2 is a sub-case/ specialisation of r_1. Therefore the behaviour represented by r_2 completely includes the behaviour represented by r_1. We can say, for example, *Employee in Manager*, and then the role representing *Manager* behaviour completely contains the behaviour of the *Employee* role. If r_1 in r_2 and we compose r_1 and r_2, the resulting role contains the characteristics of r_2 only. In terms of the two-sorted algebra, contained roles must be assigned and played by the same agent as their containers:

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

Therefore the relation $Contains \subseteq R \times R$ is reflexive and transitive:

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

4.2.4 Requires

The relation $Requires \subseteq R \times R$, denoted with '*and*', can be used to describe that when an agent is assigned a particular role, then it must also be assigned some other specific role as well. In terms of the two-sorted algebra, required roles must be played by the same agent as the roles that require them:

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

This is particularly applicable in cases where agents need to conform to general rules or play organisational roles. For example, in a university application context, in order for one to be a *Library_ Borrower* one must be a *University_Member* as well. Although the behaviour of a *Library_Borrower* could be modelled as part of the behaviour of a *University_Member*, this would not be convenient since this behaviour 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:

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

4.2.5 Addswith

The relation $Addswith \subseteq R \times R$ is denoted with '*add*'. It can be used to express that the behaviours two roles represent do not interfere in any way. In the two-sorted algebra, there is no constraint in having or playing roles that add together:

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

For example, the *Student* and the *Football_Player* roles describe non-excluding and non-overlapping behaviours. Hence these roles can be assigned to the same agent without any problems. The relation $Addswith \subseteq R \times R$ is reflexive and symmetric:

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

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

4.2.6 Mergeswith

The relation $Mergeswith \subseteq R \times R$ is denoted with '*merge*'. It can be used to express that the behaviours of two roles overlap to some extent or that different behaviour occurs when two roles are put together. For example, a *Student* can also be a *Staff_Member*. This refers to cases when Ph.D. students start teaching before they complete their Ph.D. or they register for another degree (e.g. an MBA) after their graduation. Although members of staff, these persons cannot access certain information (e.g. future exam papers) due to 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. In terms of the two-sorted algebra, when two roles merge, only the unique role that results from their merge is played by an agent:

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

For example, let us assume that roles r_2 and r_3 merge, resulting to 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 the other roles the agent has). The example of a Mergeswith relation between roles r_2, r_3 and r_4, where r_4 is played by the agent, is depicted in Figure 2.

The relation $Mergeswith \subseteq R \times R$ is symmetric:

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

Relations between more than two roles can be defined in a manner similar to the binary relations, perhaps using the more convenient predicate notation. For example, when the 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, we will not provide any formal definitions for relations among roles with arity greater than two.

5 A synthesis-based agent organisation design method

5.1 The synthesis design process

Role relations, as defined in the above algebra, restrict the way that roles can be allocated to agents. The agent organisation design problem is thus 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 in the performance variables of the agents. For example, the system designer should be able to define the maximum number of roles that an agent could play, or an upper limit to the resource capacity that an agent would require. Furthermore, role allocation heuristics could also be specified. For example, roles requiring access to similar resources could be assigned to the same agent.

A generic approach for this type of constraint-based design problem is synthesis. The term 'synthesis' in engineering disciplines refers to a process in which a problem specification is transformed to a solution by decomposing the initial problem into loosely coupled sub-problems that are independently solved and integrated into an overall solution while various constraints within and among sub-solutions are observed (Samuel & Weir, 1999; Antonsson & Cagan, 2001).

The synthesis problem solving process involves searching solution alternatives in the corresponding solution domain and selecting appropriate solutions based on explicit quality criteria. In agent-based system design, the problem is the application requirements and the possible solutions are agent-based systems that can fulfil them. Therefore synthesis can be used in the agent-based system design as well.

Generally, a synthesis-based design process includes the following phases (Karageorgos, 2002): problem requirements specification, synthesis problem formulation, synthesis problem refinement, division of the main problem to sub-problems as appropriate and synthesis problem solution. The generic synthesis-based design process is depicted in Figure 3.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

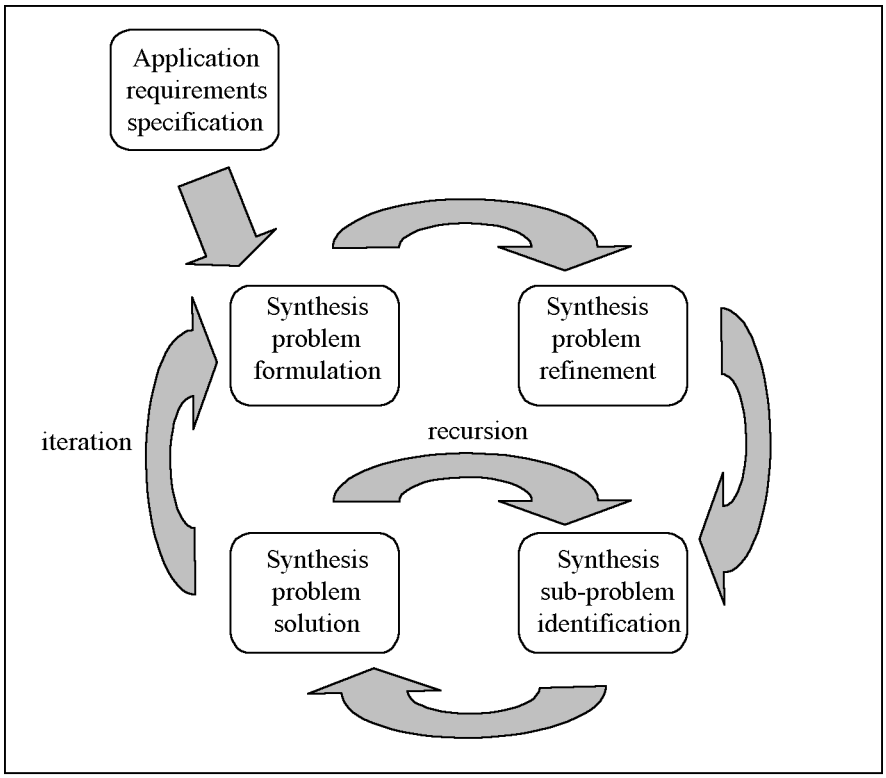


Figure 3 A generic synthesis-based design process

The overall synthesis problem solution often requires iteration and recursion. Iteration refers to repeating phases of the synthesis process in order to improve the synthesis solution results. Iterations start from the synthesis problem definition phase where the problem is redefined seeking a better solution. This may involve neglecting or changing the application requirements. Recursion refers to repeating the sub-problem partitioning and search phases of a synthesis process for a lower abstraction level. This involves decomposing the overall problem into (perhaps different) sub-problems again and relaxing or increasing the problem specification constraints.

Generally, to obtain satisfactory solutions in a synthesis problem, a number of recursions and iterations is required.

5.2 RAMASD: a semi-automatic method for designing role-based agent organisations

Considering the generic synthesis-based design process model, a systematic process for designing agent organisations can be introduced. This process involves both manual and automatic steps where manual steps are carried out by the human designer and automatic steps by some software tool. The manual and automatic steps of the semi-automatic method for role-based agent organisation design are the following (see also Figure 4):

1. *Specify application requirements* The agent-based system design is initiated with the requirements analysis phase in which the basic goal is to understand the stakeholder requirements. The well-known requirements analysis techniques such as textual requirements specifications, use-cases (Jacobson *et al.*, 1994), and even formal requirements specification languages (Greenspan *et al.*, 1994) can be used. In all cases, however, requirements should be specified in such a way to assist role identification, which follows in the next stage.
2. *Identify roles and 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 use-case identify roles and their interactions (Andersen, 1997). Many role interaction patterns can be used directly from existing

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

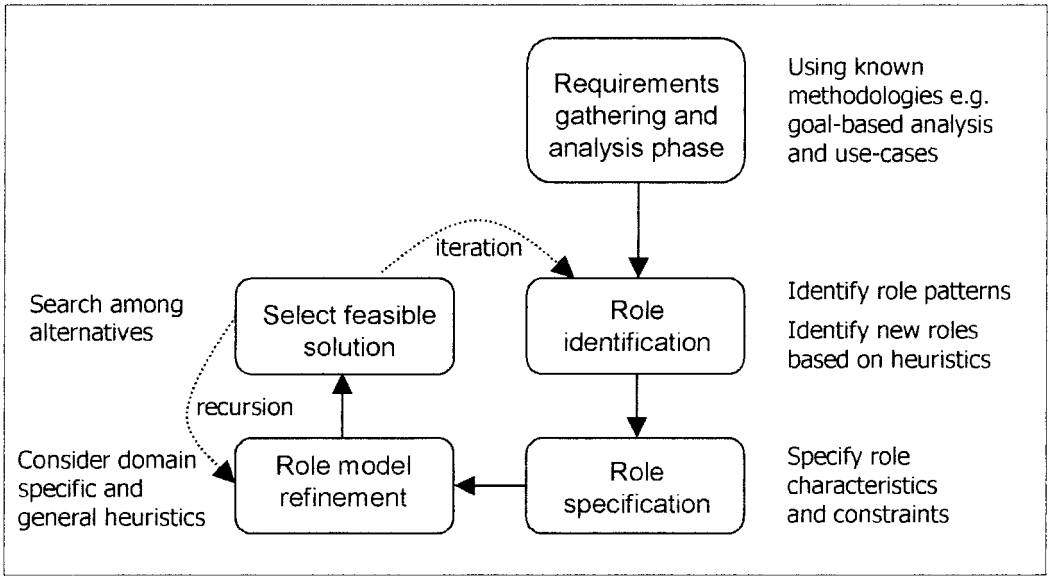


Figure 4 Semi-automatic method for agent organisation design

role-pattern libraries like the one documented at BT (Kendall, 1999a). Selection or definition of appropriate role models is a manual step that must be carried out by humans.

3. *Specify relevant role characteristics and compositional constraints* This is an automatic step since role characteristics and inter-role relations are expected to be stored in some role-model library. After the designer selects existing role models, role characteristics and role compositional constraints are automatically retrieved.
4. *Refine role models* The agent system designer is expected to manually specify role characteristics and role relations for any new role models he or she defines. These new role models should be stored in the role-model library. At this step, additional characteristics of existing role models, for example performance variables, should also be specified. Furthermore, at this step various domain-specific and general design heuristics are specified as constraints on the performance variables of roles and agents.
5. *Assign roles to agents* Performing the search among various alternatives and allocating roles to agents can be done automatically. However, at this point the process may continue to a next recursion or iteration cycle depending on the results obtained. Iteration and recursion possibilities make the design of the agent organisation an interactive process where routine tasks are automated and humans carry out tasks requiring experience and creative decisions.

5.3 Compatibility with agent systems engineering methodologies

An important advantage of our agent organisation design method, which we call RAMASD, is that it can be used in conjunction with existing multi-agent system engineering methodologies. The degree of compatibility between RAMASD and existing methodologies for engineering agent organisations depends on whether they use the role concept for modelling the agent behaviour.

High compatibility exists when the methodologies include the role concept as is the case with the majority of informal approaches, for example MESSAGE/UML, SODA, Gaia and Zeus. As mentioned in Section 2.6, the main weakness of such approaches is that they do not provide any formal underpinnings to drive the design of the agent organisation.

RAMASD can be used after the analysis stage to drive the design of the agent components. This requires formulating the role allocation problem in RAMASD, including the definition of constraints on roles and role characteristics and the possible introduction of additional role models to represent organisational and non-functional aspects. After role allocation, the remaining phases of each existing

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

methodology can be followed, for example in SODA the topological model could be created and considered after designing the agent components using RAMASD.

Currently, RAMASD is not fully compatible with all possible specifications that can be done in methodologies including the notion of role. For example, the methodology described in Zambonelli *et al*. (2001b) involves specifying complex organisational rules in temporal logic and this is not compatible with RAMASD in its present form. The RAMASD philosophy of trying to reduce the design complexity that designers have to handle by automating the role allocation process based on inter-role constraints can still be applied in all cases.

The compatibility of RAMASD is low in methodologies where the role concept is not included, for example DESIRE. In such cases, RAMASD can still be useful but an additional role-modelling phase is required. For example, in DESIRE the tasks the agent organisation needs to carry out identified in the analysis phase should be analysed according to some task analysis method such as Chandrasekaran *et al*. (1992) and grouped to roles following general heuristics.[2] Subsequently, additional roles to represent organisational and non-functional aspects should be introduced as required and role allocation could take place. Finally, the tasks associated with each agent component could be determined from the roles allocated to it and the rest of the DESIRE methodology (task and component specification, code generation) could be applied normally.

6 Example: supporting mobile work teams

For this example a case study concerning telephone repair service teams is considered. The aim is to build an agent system that would assist field engineers to carry out their work. Among the issues involved in such a system are those of *travel management*, *teamwork coordination and knowledge management* (Thompson & Odgers, 2000; Stark *et al.*, 2001).

Travel management is about support to mobile workers for moving from one repair task location to another. It involves finding the position of each worker, obtaining relevant travel information, planning the route to the next repair task location and allocating travel resources as required. Teamwork coordination is about allocating and coordinating the execution of repair tasks in a decentralised manner taking into account the personal preferences and working practices of the mobile workers. Work knowledge management concerns storage and dissemination of work-related expertise.

6.1 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. According to Kendall and Zhao (1998) a way to identify roles in an application domain is to start by identifying use-cases, associating each use-case with a goal, creating a goal hierarchy from the use-case hierarchy and coalescing semantically relevant goals to roles. For the purpose of the telephone repair service teams example, the following use-cases are considered (see Figure 5):

- *Teamwork coordination* In this activity the customer places a request for a telephone repair. This request is placed in a pool of repair request tasks and it is eventually allocated to some mobile field engineer who will be responsible for its execution.
- *Travel management* This involves providing up-to-date travel information to the field engineer, including his current exact location, an optimal plan of the route to the next telephone repair task, as well as traffic information and managerial policy regarding travelling.
- *Work knowledge management* This activity deals with maintaining and storing expertise for complex telephone repair tasks.

[2] For example, such an approach has been followed in Kendall (1999b).

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

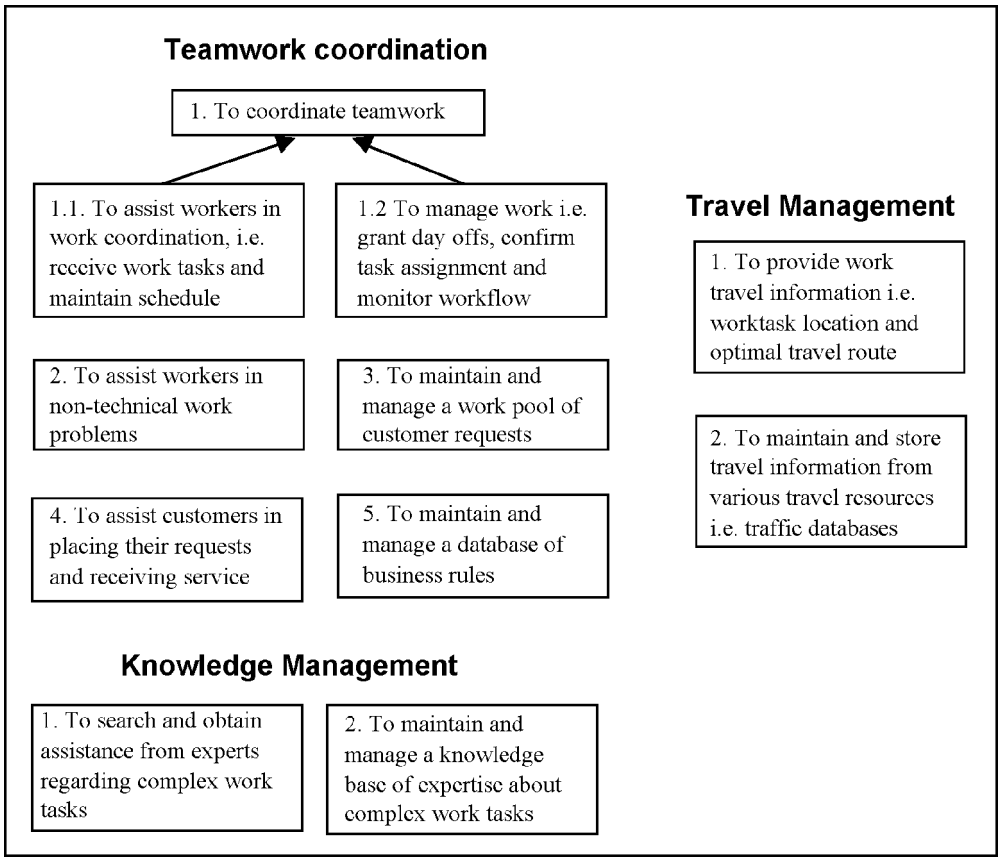


Figure 5 Use case goals for the telephone repair service teams case study

Each use-case has a number of high-level goals depicted in Figure 5. The behaviour leading to achieving these goals can be modelled by appropriate roles. Hence the following roles can be identified (see Figure 6):

1. *Employee* This role describes generic behaviour of the members of the customer service teams. An example of this type of behaviour is accessing common team resources including work practice announcements and business news.
2. *Coordinator* The *Coordinator* role describes the behaviour required to coordinate the work of a field engineer. This includes bidding for and obtaining repair work tasks from a work pool, negotiating with other workers and the team manager as required and scheduling and rescheduling work task execution.
3. *Manager* The *Manager* role models the behaviour of the team manager. This includes confirming task allocation, monitoring work and ensuring that business rules are followed.
4. *Mentor* The *Mentor* role provides assistance to field engineers for non-technical issues.
5. *WorkPool* The *WorkPool* role maintains a pool of telephone repair requests. Customers interact with this role to place requests and engineers interact with this role to select tasks to undertake.
6. *Customer* The *Customer* role models the behaviour of a customer. It involves placing telephone repair requests, receiving relevant information and arranging appointments with field engineers.
7. *Brulebase* This role maintains a database of business rules. It interacts with Manager, providing information about the current work policy of the business.
8. *TravelManager* The *TravelManager* role provides travel information to the field engineer including current location, traffic information and optimal route to next telephone repair task.
9. *TravelInfoBase* This role stores travel information from various travel resources, i.e. GPS and traffic databases.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

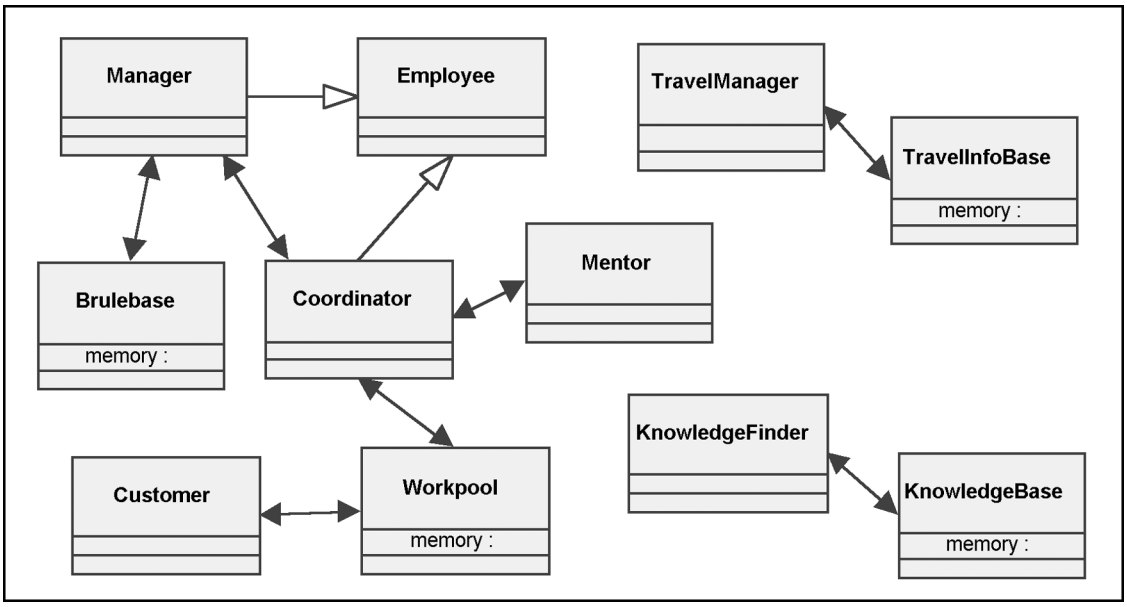


Figure 6 Role models for the telephone repair service teams case study

10. *KnowledgeFinder* This role searches for experts and obtains assistance regarding complex work tasks.
11. *KnowledgeBase* The KnowledgeBase role maintains and manages a database of expertise about telephone repair tasks.

6.2 Specifying design constraints

In Figure 7, compositional constraints for the roles described in Section 6.1 are specified in *RCL* (Role Constraint Language).

```
/* ROLE DEFINITIONS */

Role employee, coordinator, mentor,
     customer, travelmanager,
     knowledgefinder;

Role workpool, brulebase, workerassistant,
     travelinfobase, knowledgebase {
     int memory;
}
     workpool.memory = 1;
     brulebase.memory = 1;
     travelinfobase.memory = 2;
     knowledgebase.memory = 2;
     workerassistant.memory = 1;

Role manager {
     collaborators = {Coordinator,
                      Brulebase};
     protocols = {contracting};
}
```

```
/* ROLE CONSTRAINTS */

in(employee, coordinator);
in(employee, manager);

not(customer, employee);
not(customer, travelinfobase);
not(customer, knowledgebase);
not(mentor, manager);
not(manager, coordinator);

and(mentor, employee);

merge(coordinator, travelmanager,
knowledgefinder,
workerassistant);

/* GENERAL CONSTRAINTS */

Constraint Y {
      forall a:Agent {
      a.memory <= 2
      }
}
```

Figure 7 Compositional constraints for the telephone repair service teams case study

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

RCL is a simple declarative constraint language that was introduced to represent design constraints on agent and role characteristics. The use of *RCL* in Figure 7 is self-explanatory and therefore in this paper no further description of *RCL* will be given. *RCL* is described in detail in Karageorgos (2002).

Roles in *RCL* are specified in a manner similar to programming languages. In the telephone repair service teams example, roles that directly manipulate databases require access to some storage space. This is modelled by the performance variable memory. The memory requirements of each role are different. For example, *TravelInfoBase* and *KnowledgeBase* require twice as much memory as *WorkPool* and *Brulebase*.

Part of the definition of the characteristics of the *Manager* role is shown in more detail in Figure 6. The collaborators of the *Manager* role are the *Coordinator* and *Brulebase* roles and its interaction protocol is the Contract Net. The *Employee* role is contained in both *Manager* and *Coordinator* roles. Furthermore, a *Manager* cannot coexist with *Mentor* or *Coordinator* and for security purposes a *Customer* cannot coexist with *Employee*, *TravelInfoBase* or *KnowledgeBase*. In order for an agent to be *Mentor* it must also be an *Employee*.

When an agent plays all three *Coordinator*, *TravelManager* and *KnowledgeFinder* roles, overheads in synchronising results from the three different activities (travel management, teamwork coordination and knowledge management) may occur. This is modelled as a merge of the *Coordinator*, *TravelManager* and *KnowledgeFinder* resulting in the *WorkerAssistant* role. The *WorkerAssistant* role requires some memory to store intermediate synchronisation results, as specified in Figure 7. This is an illustration of the *Has* and *Plays* relations used to represent the semantics of role relations described in Section 4.1: The agent that is aware of the *Coordinator*, *TravelManager* and *KnowledgeFinder* roles does not actually exercise those behaviours. Instead it is also aware and it actively demonstrates the behaviour of the *WorkerAssistant* role with the only difference in this simple example that the *WorkerAssistant* role requires additional memory.

An example of a non-functional requirement is to limit the memory each agent could occupy. In this case study, agents supporting field engineers should be able to operate in PDAs with limited amounts of memory. This is modelled as a general design constraint on the performance variable memory (Figure 7).

We have encoded the basic role relationships and the example application using DAML-OIL to illustrate how role models can be shared using Semantic Web standards. The DAML-OIL classes, properties and relationships for the case study are shown in Figure 8. Limitations of DAML-OIL do not allow us to encode the full definitions of role relationships, but the current degree of specification is sufficient for publishing and sharing of role models to be reused in different agent system designs.

The current version of RAMASD has been implemented as an extension to the Zeus agent building toolkit (Nwana *et al.*, 1999) developed at BT labs. Currently, the RAMASD extensions to the Zeus version include 4 components: The new *Agent Generator* component (Figure 9) provides an interface for editing roles and role models and storing and retrieving them from a role-model library. The Agent Generator also links the user with the other pre-existing components of the Zeus agent building toolkit. The *Role Constraints Editor* provides an interface for specifying constraints among roles and role characteristics and the *Role Allocator* component allows the user to initiate the role allocation process (Figure 10) and select the search algorithm that will be used to search for a design solution. Furthermore, there is a *Code Generation* component which produces Java source code for the designed agent-based system.

6.3 Allocating roles to agent types

The constraints on roles and agent and role characteristics define a constraint satisfaction problem, which must be solved to allocate roles to agent types. Search problems are hard to address optimally and therefore sub-optimal solutions are widely adopted.

A simple heuristic that can be used for role allocation is to try to minimise the number of agent types produced. Therefore as many roles are allocated to an agent type as possible before moving to the next. Merging roles are processed first and the algorithm moves to the remaining roles only when all roles

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

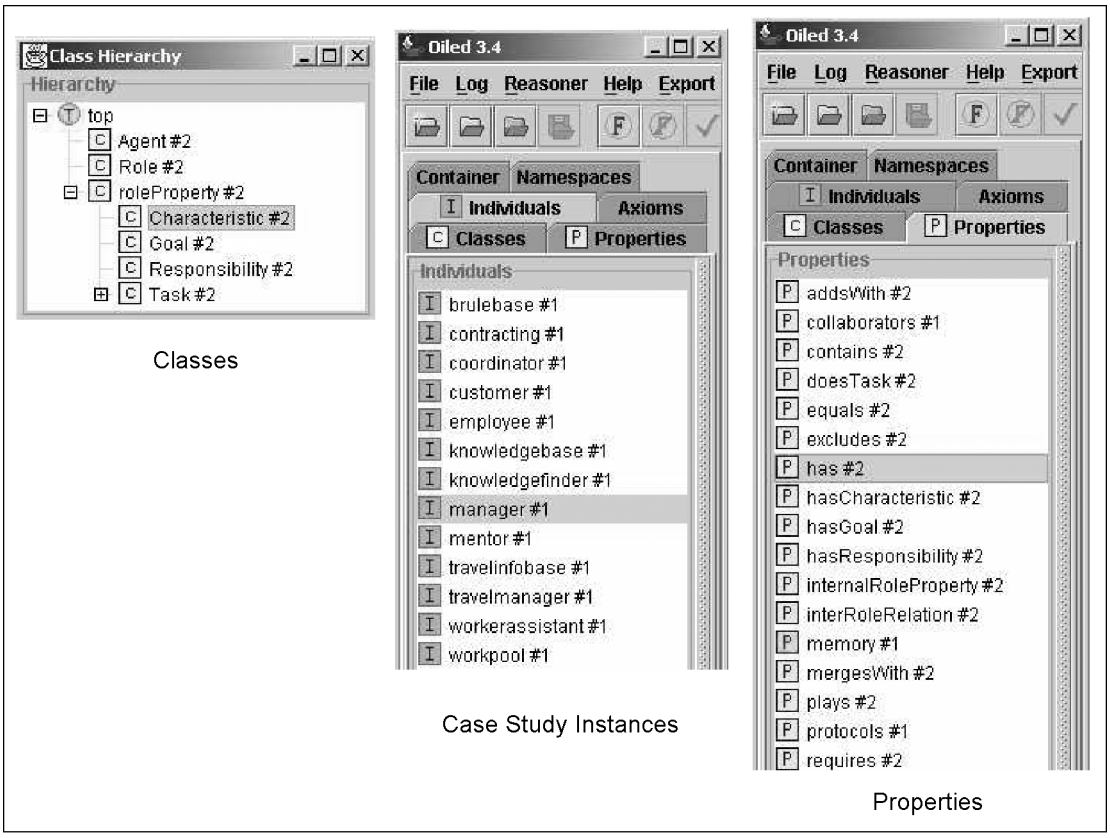


Figure 8 DAML-OIL representation of role relations in the telephone repair service teams case study

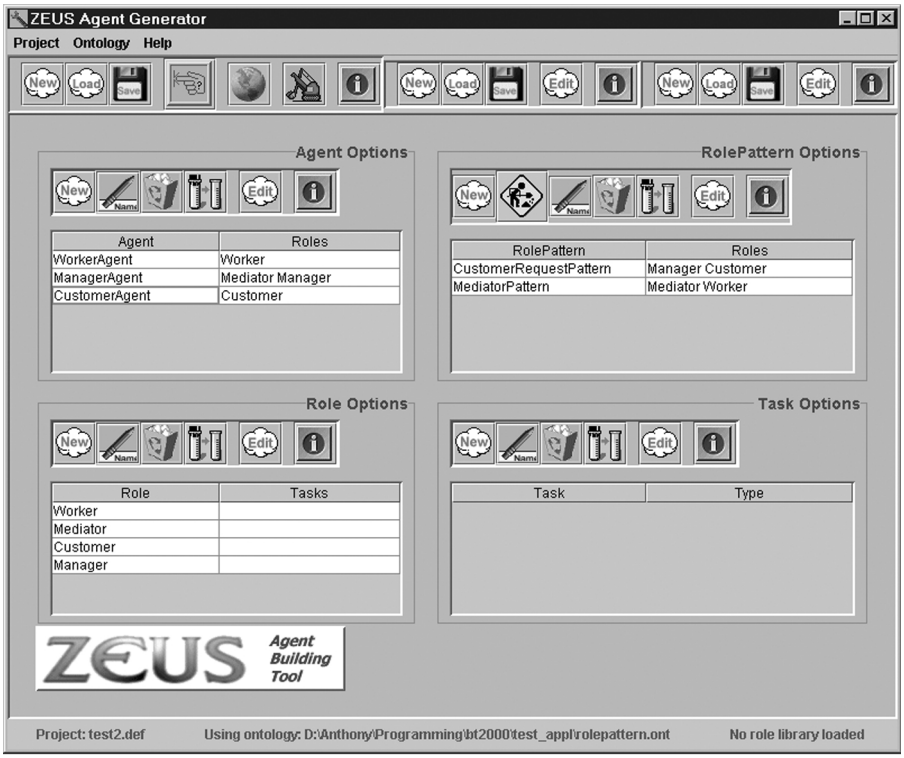


Figure 9 The extended Zeus Agent Generator implementing the RAMASD method

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

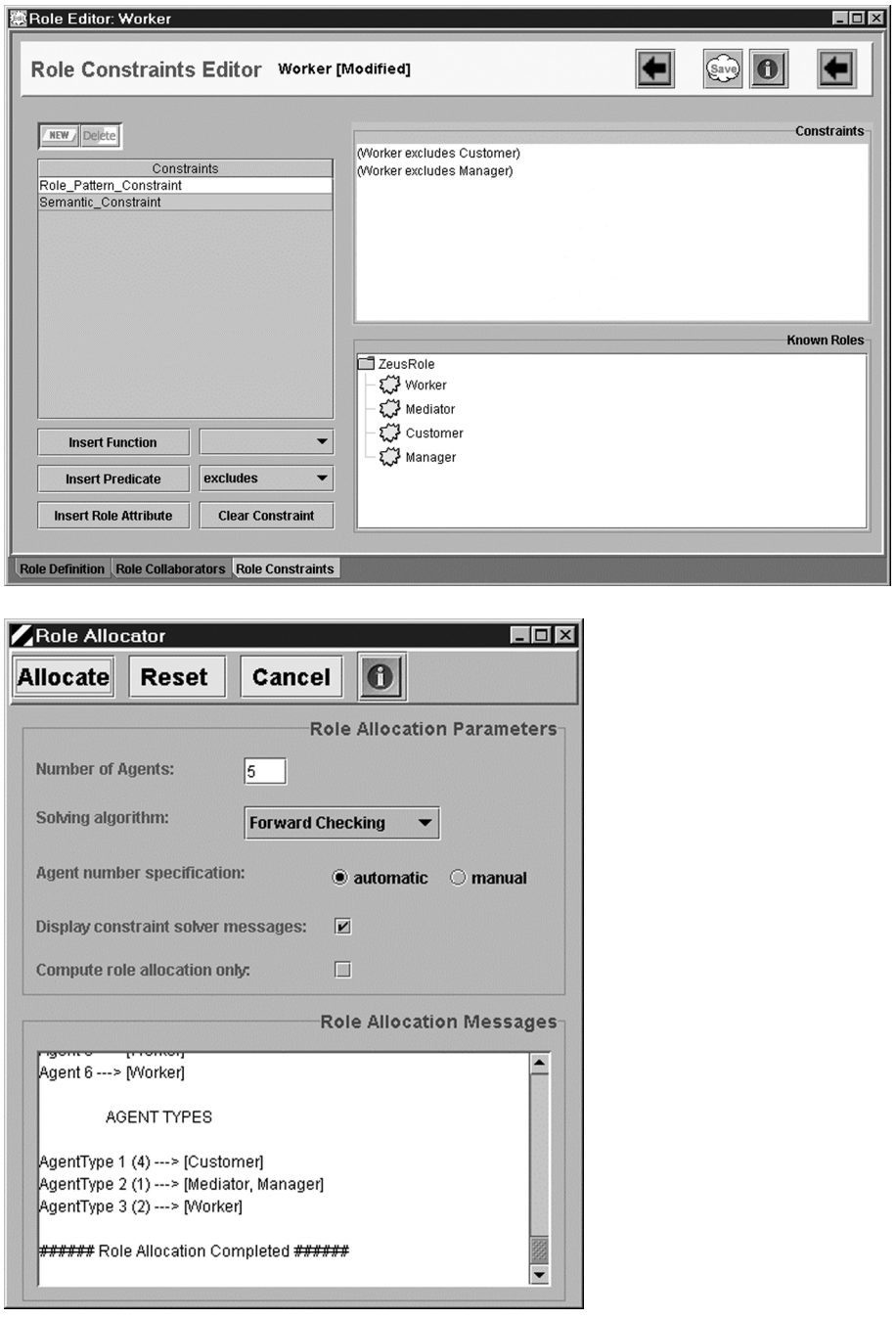


Figure 10 The role constraints editor and the role allocator component of the extended Zeus agent building tool

that are involved in a *Mergeswith* constraint have been allocated to an agent type. All roles are allocated to an agent type at least once.

Applying this heuristic produces an agent organisation satisfying the design constraints described in Section 6.2, which includes five agent types. The agent types produced are depicted in Figure 11.

7 Limitations of RAMASD – discussion

RAMASD is a method primarily aiming at reducing the complexity that agent-based system designers have to handle. Towards this aim, it considers a formal model of role relations concerning role

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

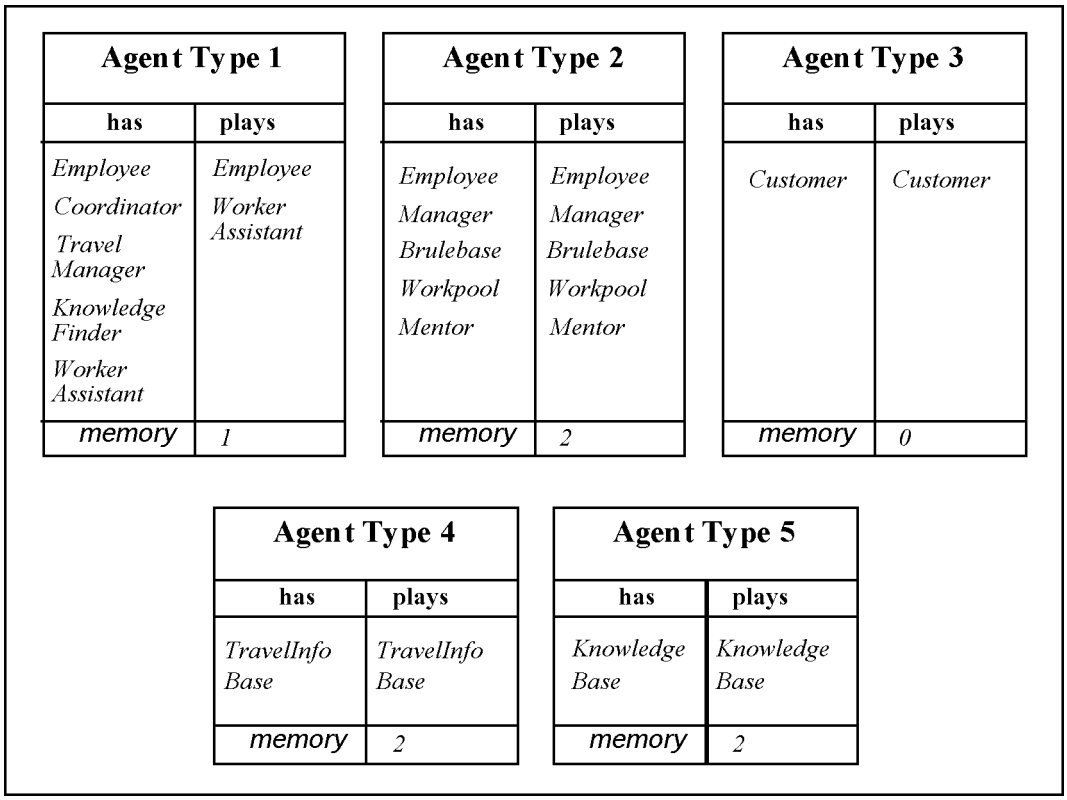


Figure 11 Agent types for the telephone repair service teams case study

composition when roles are assigned to agents and it uses this model to drive the role allocation process.

The benefit of using two relations, namely *Has* and *Plays*, to represent the semantics of the role relationships is twofold: first it assists in allocation of the specified roles to agent types and second it makes it possible to allow modification of agent behaviour (role allocation) at run time. For example, if an agent were to receive a new role dynamically, any constraints between the new role and the roles already owned by an agent would be considered even if the agent does not actively play all these roles. This will allow a decision to be made if the new role can be added to the set of roles played by the agent. In this case, having two set of roles associated with the agent allows modelling of a richer set of behavioural constraints. However, dynamic role allocation is outside the scope of this paper.

Furthermore, as research on RAMASD is still ongoing the version of RAMASD reported in this paper has certain limitations:

- *Organisational rule specification* The current version of RAMASD is restricted as concerns the types of organisational rule that can be specified. As mentioned in Zambonelli et al. (2001b), organisational rules can be quite complex, involving temporal and dynamic relationships among roles and role characteristics in an agent organisation. The current RAMASD modelling framework allows for modelling only simple types of organisational rule and instead emphasises reducing the complexity of the agent-based system design task by providing automatic allocation of roles to agents.
- *Modelling non-functional aspects* Currently, only simple quantitative models of non-functional aspects are considered in RAMASD, for example it has been assumed that the relation between the agent performance variables and the performance variables of the agent roles is linear. In the case study this is reflected by assuming that the memory needed by an agent is equal to the sum of the memories needed by the roles the agent plays. This is not generally the case, however, as some non-functional parameters change dynamically over time. Therefore it is expected that sophisticated models of non-functional aspects need to be considered in later versions of RAMASD.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

- *Scalability of RAMASD* RAMASD has so far been applied in case studies involving a relatively small number of roles (up to 20 roles) and a small number of inter-role constraints (up to 30 role constraints). In those numbers the RAMASD search algorithm as well as standard search algorithms managed to provide design solutions at reasonable time. However, RAMASD has yet to be tested in the design of agent organisations involving large numbers of roles and role constraints. It is envisaged that suitable (most probably heuristic) search algorithms will be required to handle large numbers of roles and design constraints and this is currently the subject of ongoing work.

8 Conclusions and further work

Current methodologies for engineering multi-agent systems do not adequately support the transformation of analysis knowledge to design decisions that take place when designing agent organisations, requiring the designers to manually address the complexity of the design problem. This makes designing large, real-world multi-agent systems a tiresome and error-prone exercise.

To address these concerns, we propose RAMASD: a method for designing agent organisations which reduces design complexity by semi-automating the design process. RAMASD achieves semi-automation based on: (a) an extended definition of the role concept which allows modelling of non-functional aspects as constraints on special role characteristics called performance variables and (b) a formal model of role relationships concerning ways that roles can be assigned and played by agents. This formal relationship model is called role algebra, and it is used to drive a synthesis design process which allocates roles to agents whilst observing any design constraints specified on inter-role relationships and performance variables. RAMASD enables reusing organisational design knowledge by allowing designers to specify and retrieve relevant role models and to manipulate them using the role algebra. Furthermore, RAMASD can be used in conjunction with a number of existing multi-agent system design methodologies which make use of the role concept.

Some issues, however, are not yet addressed by RAMASD, and have been established as directions for future work. One such issue is the influence of context when designing agent organisation, since agents can play different roles in different contexts. A longer-term research task would be to investigate the impact of dynamically allocating and de-allocating roles to agents at run time by using our role algebra and to explore sophisticated models of non-functional aspects. On a more applied level, we would like to design a mechanism where libraries of role models can be published on the Semantic Web and shared using standard ontology description languages such as DAML-OIL.

References

Andersen, E. P. *Conceptual Modelling of Objects: A Role Modelling Approach*. PhD Thesis. Oslo, Norway: University of Oslo, 1997.

Antonsson, E. K., and Cagan, J. (eds.). *Formal Engineering Design Synthesis*. Cambridge: Cambridge University Press, 2001.

Bartelt, A., and Lamersdorf, W. Agent-Oriented Concepts to Foster the Automation of e-Business. In Tjoa, A. M., Wagner, R. R. and Al-Zobaidi, A. (eds.), *Proceedings of the 11th International Workshop on Database and Expert Systems (DEXA 2000)*. Munich, Germany: IEEE Computer Society Press, 2000, 775–779.

Bauer, B.; Muller, J. P., and Odell, J. Agent UML: A Formalism for Specifying Multiagent Software Systems. In Ciancarini, P. and Wooldridge, M. (eds.), *Agent-Oriented Software Engineering*. Berlin: Spriger-Verlag, 2000, 106–119.

Bellifemine, F.; Poggi, A., and Rimassa., G. JADE – a FIPA-compliant agent framework. *PAAM 99*. London, UK, 1999, 97–98.

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

Biddle, B. J., and Thomas, E. J. *Role Theory: Concepts and Research*. Huntington, New York: Robert E. Krieger Publishing Company, 1979.

Brazier, F.; Keplicz, B. D.; Jennings, N. R., and Treur, J. Formal Specification of Multi-Agent Systems: a Real-World Case. *First International Conference on Multi-Agent Systems (ICMAS'95)*. San Francisco, CA., 1995, 25–32.

Brazier, F. M. T.; Eck, P. A. T. V., and Treur, J. Modelling a Society of Simple Agents: From Conceptual Specification to Experimentation. *Applied Intelligence*, **14**(2) (March-April 2001), 161–178.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

Brazier, F. M. T.; Dunin-Keplicz, B.; Jennings, N., and Treur, J. *DESIRE*: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems, Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems*, **5**(1) (June 1997), 67–94.

Bresciani, P.; Perini, A.; Giunchiglia, F.; Giorgini, P., and Mylopoulos, J. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In Andre, E. and Sen, S. (eds.), *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS 2001)*. Montreal, Canada: ACM Press, 2001, 648–655.

Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F., and Mylopoulos, J. Modeling Early Requirements in Tropos: a Transformation Based Approach. In Wooldridge, M. J., Weis, G. and Ciancarini, P. (eds.), *Agent-Oriented Software Engineering II, Second International Workshop (AOSE 2001), Montreal, Canada*. Berlin: Springer Verlag, 2001, 151–168.

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 Wooldridge, M. J., Weis, G. and Ciancarini, P. (eds.), *Agent-Oriented Software Engineering II, Second International Workshop, (AOSE 2001), Montreal, Canada*. Berlin: Springer Verlag, 2001, 151–168.

Castro, J.; Kolp, M., and Mylopoulos, J. Developing Agent-Oriented Information Systems for the Enterprise. *Second International Conference On Enterprise Information Systems*. Stafford, UK, 2000.

Chandrasekaran, B.; Johnson, T., and Smith, J. W. Task Structure Analysis for Knowledge Modeling. *Communications of the ACM*, **33**(9) (September 1992), 124–136.

Cohen, P. R., and Levesque, H. J. Intention is choice with coomitment. *Artificial Intelligence*, **42**, 213–261.

Collins, J., and Ndumu, D. The Zeus Agent Building Toolkit:The Role Modelling Guide.

Cremonini, M.; Omicini, A., and Zambonelli., F. Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In J.Garijo, F. and Boman, M. (eds.), *Multi-Agent Systems Engineering – Proceedings of the 9th European Workshop on Modelling Autonoumous Agents in a Multi-Agent World (MAMAAW '99)*. Berlin: Springer Verlag, 1999, 77–88.

Dale, J., and Mamdani, E. Open Standards for Interoperating Agent-Based Systems. *Software Focus (Wiley)*, **2**(1) (Spring 2001), 1–8.

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

Elammari, M., and Lalonde, W. An Agent-Oriented Methodology: High-Level and Intermediate Models. *CaiSE Workshop on Agent Oriented Information Systems (AOIS99)*. Heidelberg: MIT Press, 1999.

Evans, R. MESSAGE: Methodology for Engineering Systems of Software Agents. *Initial Methodology*, Heidelberg: EURESCOM, (July 2000), EURESCOM research report, 75, `http://www.eurescom.de/~pub-deliverables/P900-series/P907/D1/P907D1.zip`.

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

Fisher, M., and Wooldridge, M. On the formal specification and verification of Multi-Agent Systems. *International Journal of Cooperative Information Systems*, **6**(1), 37–65.

Gasser, L. Perspectives on Organizations in Multi-agent Systems. In Luck, M., Marik, V., Stepankova, O. and Trappl, R. (eds.), *Multi-Agent Systems and Applications, 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001, Prague, Czech Republic, July 2–13, 2001. Selected Tutorial Papers*. Springer Verlag, 2001, 1–16.

Gerber, C. Bottleneck Analysis as a Heuristic for Self-Adaption in Multi-Agent Societies. *Joint Conference on the Science and Technology of Intelligentt Systems ISIC/CIRA/ISAS*. GAITHERSBURG, MARYLAND: IEEE Control Systems Society Press, 1998.

Giunchiglia, F.; Mylopoulos, J., and Perini, A. *The Tropos Software Development Methodology: Processes, Models and Diagrams*. Submitted to AAMAS 2002, Bologna: ITC – IRST, 2002.

Glaser, N. The CoMoMAS Methodology and Enironment for Multi-Agent System Development. In Zhang, C. and Lukose, D. (eds.), *Multi-Agent Systems: Methodologies and Applications, Second Australian Workshop on Distributed Artificial Intelligence, Cairns, Queensland, Australia, August 27, 1996, Revised Papers*. Berlin: Springer Verlag, 1997, 1–16.

Greenspan, S.; Mylopoulos, J., and Borgida, A. On formal requirements modeling languages: RML revisited. *Proceedings of the 16th International Conference on Software Engineering*. IEEE Computer Science Press, 1994, 135–148.

Grundy, J. C. Multi-perspective specification, design and implementation of components using aspects. *International Journal of Software Engineering and Knowledge Engineering*, **10**(6) (December 2000), 713–734.

Hilaire, V.; Koukam, A.; Gruer, P., and Muller, J.-P. Formal Specification and Prototyping of Multi-Agent Systems. In Omicini, A., Tolksdorf, R. and Zambonelli, F. (eds.), *Engineering Societies in the Agents World,*

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

First International Workshop, ESAW 2000, Berlin, Germany, August 21, 2000. Berlin: Springer Verlag, 2000, 114–127.

Iglesias, C. A.; Garrijo, M., and Gonzalez, J. C. A Survey of Agent-Oriented Methodologies. In Muller, J., Singh, M. P. and Rao, A. S. (eds.), *Proceedings of the 5th International Workshop on Intelligent Agents {V}: Agent Theories, Architectures, and Languages (ATAL-98)*. Heidelberg, Germany: Springer-Verlag, 1999, 317–330.

Iglesias, C. A.; Garijo, M.; Gonzalez, J. C., and Velasco, J. R. Analysis and Design of Multiagent Systems using MAS-CommonKADS. In Singh, M. P., Rao, A. S. and Wooldridge, M. J. (eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL '97)*. Berlin, Germany: Springer Verlag, 1998, 313–326.

Ishida, T.; Gasser, L., and Yokoo, M. Organisation Self-Design in Distributed Production Systems. *IEEE Transactions on Knowledge and Data Engineering*, **4**(2), 123–134.

Jacobson, I.; Ericsson, M., and Jacobson, A. *The Object Advantage: Business Process Re-engineering with Object Technology*. Menlo Park, CA: Addison Wesley Publishing Company, 1994.

Jennings, N. R., and Wooldridge, M. Agent-Oriented Software Engineering. In Bradshaw, J. (ed.), *Handbook of Agent Technology*. AAAI/MIT Press, 2001.

Karageorgos, A. *Reducing Complexity in Agent-Based System Design*. PhD Thesis. Manchester, UK: University of Manchester Institute of Science and Technology, 2002.

Kendall, E. A. Role models – patterns of agent system analysis and design. *BT Technology Journal*, **17**(4) (October 1999), 46–57.

Kendall, E. A. Agent Analysis and Design with Role Models. *Volume 1: Overview*, Martlesham Heath, UK: BT Exact Technologies, (January 1999), unpublished Internal BT Report, 89.

Kendall, E. A., and Zhao, L. Capturing and Structuring Goals. In Bramble, P., Gibson, G. and Cockburn, A. (eds.), *Workshop on Use Case Patterns, Object Oriented Programming Systems Languages and Architectures (OOPSLA),*. Vancouver, British Columbia, Canada: ACM Press, 1998.

Lee, L.; Nwana, H.; Ndumu, D., and Philip De Wilde. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, **16**(3) (July 1998), 94–103.

Lowry, M. R., and McCartney, R. D. (eds.). *Automating Software Design*. Menlo Park, CA: AAAI Press, 1991.

Maisano, P. JACK Intelligent AgentsTM User Guide. Melburne, Australia: Agent Oriented Software Pty. Ltd., (March 2002), Software Manual, `http://www.agent-software.com.au/shared/downloads/index.html`.

Malvile, E., and Bourdon, F. Task Allocation: A group self Design Approach. *International Conference on Multi-Agent Systems*. Paris: IEEE Press, 1998.

McCabe, F.; Odell, J., and Thompson, C. OMG Agent Platform Special Interest Group. OMG, (Spring 2002), Standardisation group meetings report.

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.

Objectspace Inc. Voyager 2.0 User's Manual. Objectspace Inc., `http://www.objectspace.com`.

Omicini, A. SODA : Societies and Infrastructures in the Analysis and Design of Agent-based Systems. In Ciancarini, P. and Wooldridge, M. J. (eds.), *Agent-Oriented Software Engineering II, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2000, 185–193.

Parunak, H. V. D. A Practitioners' Review of Industrial Agent Applications. *Autonomous Agents and Multi-Agent Systems*, **3**(4), 389–407.

Parunak, H. V. D.; Ward, A.; Fleischer, M., and Sauter, J. A Marketplace of Design Agents for Distributed Concurrent Set-Based Design. *Fourth International Conference on Concurrent Engineering, Research and Applications*. Troy, Michigan, 1997.

Parunak, V.; Sauter, J., and Clark, S. Toward the Specification and Design of Industrial Synthetic Ecosystems. In Singh, M. P., Rao, A. and Wooldridge, M. J. (eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages*. Berlin: Springer Verlag, 1998, 45–59.

Parunak, V. D.; Sauter, J.; Fleischer, M., and Ward, A. The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Autonomous Agents and Multi-Agent Systems*, **2**(2) (June 1999), 111–140.

Pattison, H. E.; Corkill, D. D., and Lesser, V. R. Instantiating Descriptions of Organisational Structures. In Huhns, M. N. (ed.), *Distributed Artificial Intelligence*. London: Pitman, 1987, 59–96.

Petrie, C. Agent-Based Software Engineering. In Ciancarini, P. and Wooldridge, M. J. (eds.), *Agent-Oriented Software Engineering II, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2000, 58–76.

Rao, A. S., and Georgeff, M. P. BDI-agents: from theory to practice. *Proceedings of the First International. Conference on Multiagent Systems*. San Francisco, 1995.

Reticular Systems Inc. Agent Construction Tools. (Summer 2000), `http://www.agentbuilder.com/agenttools.htm`.

Samuel, A., and Weir, J. *Introduction to Engineering Design: Modelling, Synthesis and Problem Solving*. Oxford: Butterworth-Heinemann, 1999.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press

Schreiber, G.; Akkermans, H.; Anjewierden, A.; Hoog, R. d.; Shadbolt, N.; Velde, W. v. d., and Wielinga, B. *Knowledge Engineering and Management: The CommonKADS Methodology.* New York: MIT Press, 2000.

Scott, W. R. *Organisations: Rational, Natural and Open Systems*. New York, NY: Prentice Hall International, 1992.

Shen, W., and Norrie, D. H. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems, an International Journal*, **1**(2) (May 1999), 129–156.

So, Y.-p., and Durfee, E. H. Designing Organisations for Computational Agents. In Prietula, M. J., Carley, K. M. and Gasser, L. (eds.), *Simulating Organisations: Computational Models of institutions and groups*. Menlo Park, CA: AAAI Press, 1998, 47–64.

Sparkman, C. H.; DeLoach, S. A., and Self, A. L. Automated Derivation of Complex Agent Architectures from Analysis Specifications. In Wooldridge, M. J., Weis, G. and Ciancarini, P. (eds.), *Agent-Oriented Software Engineering II, Second International Workshop (AOSE 2001), Montreal, Canada*. Berlin: Springer Verlag, 2001, 278–296.

Stark, J.; Rodriguez, M.; Leal, F., and Kearney, P. ACSOSS: a case study applying the MESSAGE analysis method. Martlesham Heath: BT Exact Technologies, (January 2000), EURESCOM research report.

Tambe, M. Towards flexible teamwork. *Journal of AI Research*, **7** (July-Dec 1997), 83–124.

Tambe, M.; Pynadath, D. V., and Chauvat, N. Building Dynamic Agent Organisations in Cyberspace. *IEEE Internet Computing*, **4**(2) (March/April 2000), 65–73.

Tambe, M.; Pynadath, D. V., and Chauvat, N. Rapid integration and coordination of heterogeneous distributed agents for collaborative enterprises. *DARPA JFACC symposium on advances in Enterprise Control*. 2000.

Thompson, S. G., and Odgers, B. R. Collaborative Personal Agents for Team Working. In Kitchin, D., Aylett, R., McCluskey, L., Porteous, J. and Steel, S. (eds.), *Proceedings of 2000 Artificial Intelligence and Simulation of Behaviour (AISB) Symposium*. Birmingham, England, ISBN 1-902956-14-6, 2000, 49–61.

Wagner, G. Agent-Oriented Analysis and Design of Organizational Information Systems. *Proc. of Fourth IEEE International Baltic Workshop on Databases and Information Systems*. Vilnius (Lithuania), 2000.

Wood, M., and DeLoach, S. A. An Overview of the Multiagent Systems Engineering Methodology. In Ciancarini, P. and Wooldridge, M. J. (eds.), *Agent-Oriented Software Engineering II, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2000, 207–221.

Wooldridge, M. J., and Ciancarini, P. Agent-Oriented Software Engineering: The State of the Art. In Ciancarini, P. and Wooldridge, M. (eds.), *Agent-Oriented Software Engineering*. Berlin: Spriger-Verlag, 2000, 1–28.

Wooldridge, M. J..; 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.

Wooldridge, M. J., and Jennings, N. R. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, **10**(2) (June 1995), 115–152.

Zambonelli, F.; Jennings, N. R., and Wooldridge, M. J. Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. In Ciancarini, P. and Wooldridge, M. J. (eds.), *Agent-Oriented Software Engineering II, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2000, 235–250.

Zambonelli, F.; Jennings, N. R., and Wooldridge, M. J. Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *International Journal of Software Engineering and Knowledge Engineering*, **11**(3) (June 2001), 303–328.

Zambonelli, F.; Jennings, N. R.; Omicini, A., and Wooldridge, M. J. Agent-Oriented Software Engineering for Internet Applications. In Omicini, A., Zambonelli, F., Klusch, M. and Tolksdorf, R. (eds.), *Coordination of Internet Agents: Models, Technologies and Applications*. Berlin Heidelberg: Springer-Verlag, 2001, 326–346.

https://doi.org/10.1017/S0269888903000572 Published online by Cambridge University Press