Semi-Automatic Design of Agent Organisations

Anthony Karageorgos
PhD Student
Dept. of Computation
UMIST, Manchester M60 1QD, UK
+44-161-2003306
karageorgos@acm.org

Simon Thompson
Team Leader
Intelligent Agents Research Group
BT Exact Technologies
+44-1473 605531
simon.2.thompson@bt.com

Nikolay Mehandjiev
Lecturer
Dept. of Computation
UMIST, Manchester M60 1QD, UK
+44-161-2003319
nikolay@computer.org

ABSTRACT
Multi-agent systems can be viewed as organisations of individual agents. Designing an agent organisation is a complex process involving defining the structural relationships among agents, the lines of inter-agent communication, and the agent functionality. Existing approaches to agent organisation design are difficult to apply in practice since they require designers to make decisions while working at a low level of abstraction.

This paper contributes towards designing agent organisations in a practical and effective manner by proposing to *semi-automate* the organisational design process. The proposed semi-automatic approach enables agent system designers to reason at a high abstraction level and conveniently re-use previous design decisions. This semi-automatic approach to agent organisation design uses role modelling and a *role algebra* which captures a number of basic relations among roles. The role algebra's semantics are formally defined using a two-sorted algebra.

The applicability of the semi-automatic agent organisation design approach is demonstrated by an example drawn from a case study involving telephone repair service teams.

Keywords
Software Agents, Multi-Agent Systems, Agent Organisations, Agent-Oriented Software Engineering.

1. INTRODUCTION
Multi-agent system architectures can be naturally viewed as organised societies of individual computational entities e.g. [5, 13, 17], and hence the problem of designing a multi-agent system refers to designing an agent organisation[1]. The criteria affecting an agent organisation design decision are numerous and highly dependent on factors that may change dynamically. Therefore, there is no standard best organisation for all circumstances [12, 13]. As a result, agent organisation design rules are left vague and informal, and their application is left on the creativity and the intuition of the human designer. This can be a serious drawback when designing large and complex real-world agent systems. Therefore, many authors argue that social and organisational abstractions should be considered as first class design constructs and that the agent system designer should reason at a high abstraction level, e.g. [7,10].

In this paper, an approach providing semi-automatic support for the high-level design of agent organisations is proposed. It uses role models as basic building blocks, and formalises the rules and constraints of their combination. This enables semi-automatic tool support for the agent organisation designer. The approach has been incorporated in an experimental version of the Zeus agent building toolkit [9].

The rest of this paper is structured as follows: in the next section some deficiencies in the current methodologies for agent-oriented software engineering are highlighted, and an overview of a semi-automatic approach to role-based agent organisation design is given. Subsequently, role characteristics and possible relations among roles are discussed in the light of examples. In the next step, a formal model of role relations, the role algebra, is defined and its semantics are discussed using a two-sorted algebra. The use of the role algebra to design an agent organisation is illustrated by an example based on a case study of telephone repair service teams. Finally, directions for future work are presented.

2. DESIGNING AGENT ORGANISATIONS
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 oriented analysis and design such as MESSAGE [4], GAIA [17] and SODA [10]. All these methodologies involve a number of analysis and design sub-models emphasising particular analysis and design aspects. Organisational settings are either specified explicitly in an organisational model e.g. [4] or are defined implicitly from the functionality that agents are assigned e.g. [7].

2.1 Weaknesses of Agent Organisation Design Methodologies
Existing approaches to designing agent systems could be further improved in the following ways:

SAC 2002, March 10-14, 2002, Madrid, Spain.

[1] In this paper the terms *multi-agent system* and *agent organisation* are used interchangeably.

- A more systematic way to construct large agent system design models from the analysis models. The main drawback of existing approaches such as MESSAGE is that after a certain point the design decisions are left solely to the creativity and the intuition of the designer. The steps involved in transforming analysis models to design models are not specified in a detail that would enable an adequate degree of automation by a software tool. This view is similar to the one described in [14] where agent architectures are automatically derived from analysis specifications.

- By considering non-functional requirements on design time. For example, it would be better to avoid massive run-time reorganisation for the sake of system stability and performance. Therefore, the aim should be to achieve as optimum organisation on design time as possible. Consequently, some means for considering non-functional requirements before actually deploying a multi-agent system is needed. This hypothesis is along the lines of similar works [11, 13] where the behaviour of a multi-agent system is modelled and studied before actual system deployment.

- By reusing organisational settings. This view regarding reuse of organisational settings has been inspired by the concepts introduced in [19]. It is believed that that work can be further extended by classifying known organisational patterns, and by providing some rigorous means for selecting them in a particular design context. In order for organisational patterns to be practically useful in implementing large-scale, real-world applications, a way to easily integrate organisational with application design decisions is required.

2.2 Background

Many modelling approaches use roles as basic building blocks. For example, roles are used in organisational theory [12] to represent positions and responsibilities in human organisations. Roles are also used in software engineering [1]. Roles are particularly suitable for modelling the behaviour of software agents, e.g. [3,7]. Agent roles are defined in a manner similar to organisational roles referring to a position and a set of responsibilities in an organisation [5]. To better represent agent concepts, the agent role definition includes additional characteristics, for example planning, co-ordination and negotiation capabilities [7].

Existing role-based approaches to multi-agent system design stress the need to identify and characterise relations between roles [1, 7]. However, only a small number of approaches attempt to investigate the consequences of role relations on the design of multi-agent systems, e.g. [7]. This is partly due to lack of formal foundations of role relationships. In this work, role relations that would affect multi-agent system design are identified and are formalised in an algebraic specification model. Role identification was based on organisational principles and in particular on *role theory* [2].

The essence of role theory is that persons are appointed to roles within an organisation, which are representations of concrete 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.

Role theory emphasises that various relations may exist between roles. For example, an examiner cannot be a candidate 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.

This paper describes part of a work where an attempt to extract role relations from human organisations with an eye on using them to specify agent behaviour has been made. The reason for searching for role relations in the human organisations domain was that agent research traditionally aimed to develop agents that mimic human behaviour and can be organised in a manner similar to humans. As roles have been extensively used in human organisations, e.g. [18], it was natural to examine human organisations to identify role relations.

3. ROLES AND ROLE MODELS

Roles can be used as building blocks for an approach to agent organisation design addressing the weaknesses described in Section 2.1. This is achieved by extending existing role definition to allow for modelling of non-functional requirements, and by introducing a systematic role model transformation technique enabling semi-automation of the design process.

3.1 Role Characteristics

Following [7], a role is defined as a *position* and a set of *characteristics*. Each characteristic includes a set of *attributes*. Countable attributes may further take a range of values. More specifically, a role is considered capable of carrying out certain *tasks* and can have various *responsibilities* or *goals* that aims to achieve. Roles normally need to interact with other roles, which are their *collaborators*. Interaction takes place by exchanging messages according to interaction protocols.

Roles can be extended to create specialised roles by a process called role *specialisation* or *refinement* [1, 7]. Specialised roles represent additional behaviour on top of the original role behaviour in a manner similar to inheritance in object-oriented systems.

In order for roles to pragmatically represent behaviour in an application domain, they need to model issues relevant to non-functional requirements 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 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

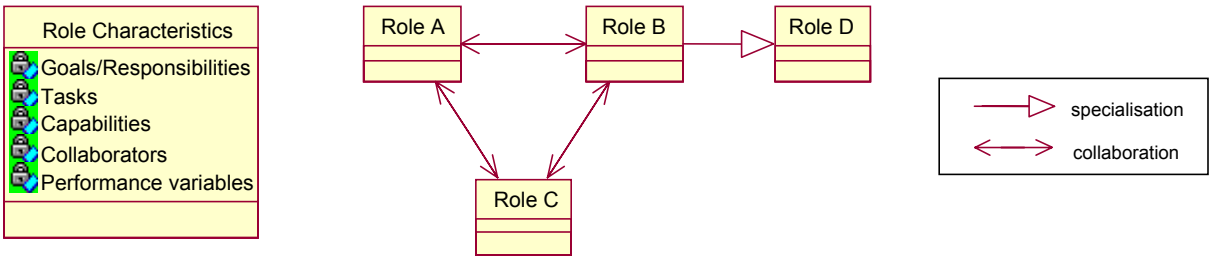

Figure 1: Schematic representation of a role model

must be observed when allocating roles to agents. This is illustrated in the example given in Section 5.

3.2 Role Models

A collection of roles and their interactions constitutes a *role model* (Figure 1). A role model represents the behaviour required to carry out some activity[2] in the system. An agent application normally consists of more than one activity and hence it will involve more than one role models. Role models that occur frequently in some application domain are called *role interaction patterns.* 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 entities that can be instantiated and given identity. Role models can be used to describe both application behaviour and 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. A ROLE ALGEBRA FOR AGENT SYSTEM DESIGN

Based on role theory [2] and on case studies of human activity systems, e.g. [15], six basic role relations have been identified. In this section, a formal model of role relations is defined, referred by the term role algebra. Using relations from the role algebra, constraints driving the assignment of roles to agents can be specified and hence the agent organisation design process can be partially automated.

4.1 Relations of the Role Algebra

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

1) **Equals (eq)** — This means that r_1 and r_2 describe exactly the same behaviour. For example, the terms *Advisor* and *Supervisor* can be used to refer to people supervising PhD students. When two roles are equal, an agent playing the one role also plays the other at the same time. 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) $\forall\, (r_1, r_2, r_3) : R \times R \times R \; ((r_1 \text{ eq } r_2) \wedge (r_2 \text{ eq } r_3) \Rightarrow (r_1 \text{ eq } r_3))$

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

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

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

3) **Contains (in)** — This means that a role is a sub-case/specialisation of another role. Therefore, the behaviour the first role represents completely includes the behaviour of the second role. For example, a role representing *Manager* behaviour completely contains the behaviour of the *Employee* role. When two roles such that the first contains the second are composed, the resulting role contains the characteristics of the first role only. 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) **Requires (and)** — The *Requires* relation 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. 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* it 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:

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

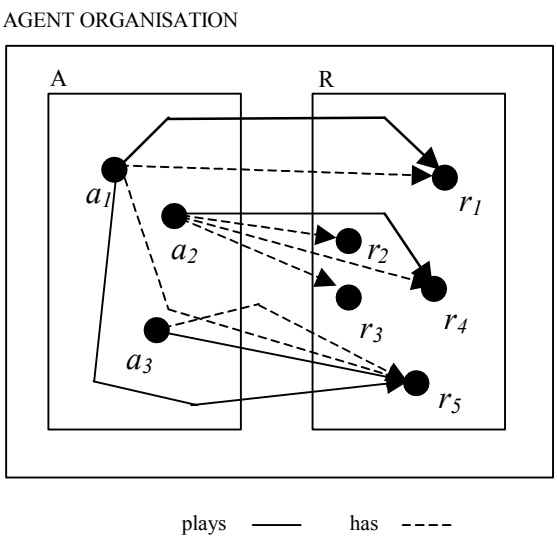

Figure 2: Semantics of role relations

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

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

5) **Addswith (add)** — The *Addswith* relation can be used to express that the behaviours two roles represent do not interfere in any way. 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) $\forall\ (r_1, r_2) : R{\times}R\ ((r_1 \text{ add } r_2) \Rightarrow (r_2 \text{ add } r_1))$

6) **Mergeswith (merge)** — The *Mergeswith* relation can be used to express that the behaviours of two roles overlap to some extend 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 PhD students start teaching before they complete their PhD 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. The relation *Mergeswith* $\subseteq R{\times}R$ is symmetric:

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

4.2 Semantics of Role Relations

To describe the semantics of role relations we represent an agent organization by a two-sorted algebra (Figure 1). The algebra includes two sorts, A representing agents and R representing roles.

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

Let *Plays*: $A \rightarrow R$ be a relation mapping agents to roles again. The term "plays" means that that the behaviour a role represents is actively demonstrated by the agent, for example the role does not merge with other roles that are also played by the agent. For each $a \in A$, let *a.plays* denote the set of roles that the agent *a* maps to in the relation *Plays*. In other words, *a.plays* denotes the relational image of the singleton $\{a\} \subseteq A$ to the relation *Plays*.

The meaning of the relations between roles introduced in Section 4.1 can now be described as follows:

1) **Equals** — An agent has and plays equal roles at the same time.

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

2) **Excludes** — Excluded roles cannot be assigned to the same agent.

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

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

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

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

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

5) **AddsWith** — There is no constraint in having or playing roles that add together.

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

6) **MergesWith** — When two roles merge, only the unique role that results from their merge is played by an agent.

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

For example, let us assume that roles r_2 and r_3 merge resulting 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.

Using the above semantic axioms, it is trivial to verify that the properties of role relations introduced in Section 4.1 hold.

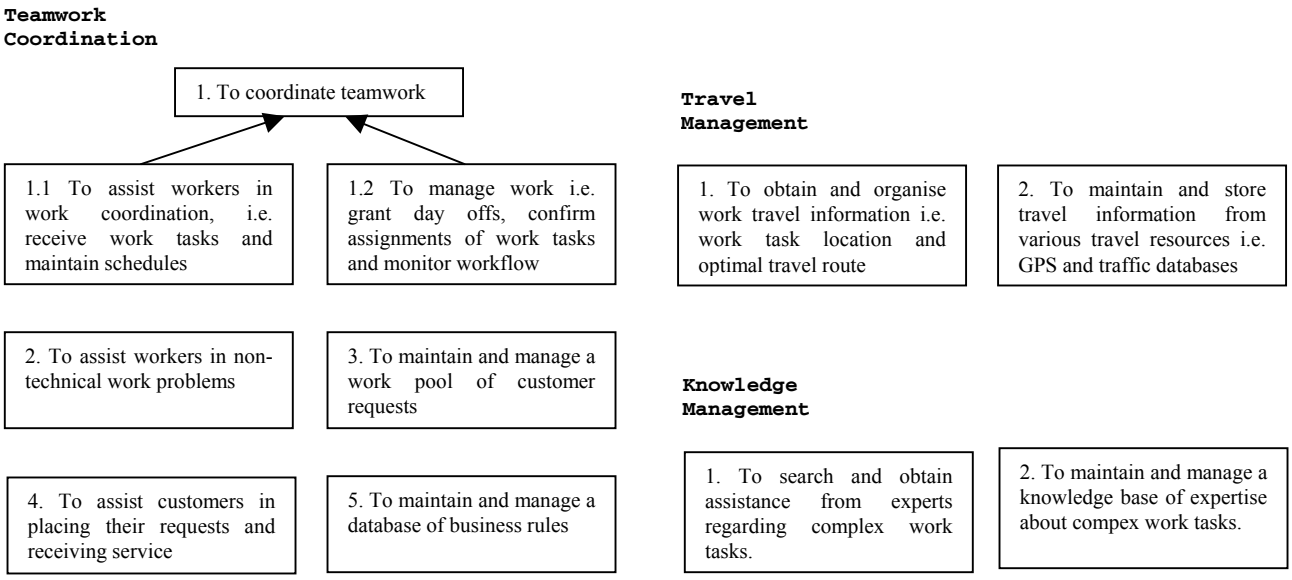

Figure 3: Use case goals for the telephone repair service teams case study

Furthermore, relations between more than two roles can be defined in a similar manner. In that case, a predicate notation is more convenient to represent role relations. For example, when three roles r_1, r_2, and r_3 merge to r_4 this can be noted by merge(r_1, r_2, r_3, r_4). In this paper, we will not provide any formal definitions for relations among roles with arity greater than two.

4.3 A Semi-Automatic Agent Organisation Approach

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 on 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.

The manual and automatic steps of the semi-automatic approach to role-based agent organisation design are the following:

1. *Select role models:* There are many ways to carry out role-based analysis. The most common approach is to start from use cases and for each use case identify roles and their interactions [1]. Many role interaction patterns can be used directly from existing role pattern libraries like the one documented at BT [7]. Selection or definition of appropriate role models is a manual step that must be carried out by humans.
2. *Specify 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.
3. *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.
4. *Specify design heuristics*: This is also a manual step where design heuristics are specified as constraints on the performance variables of roles and agents.
5. *Assign roles to agents*: Solving the constraint satisfaction problem and allocating roles to agents can be done automatically.

5. 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* [15, 16].

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 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.

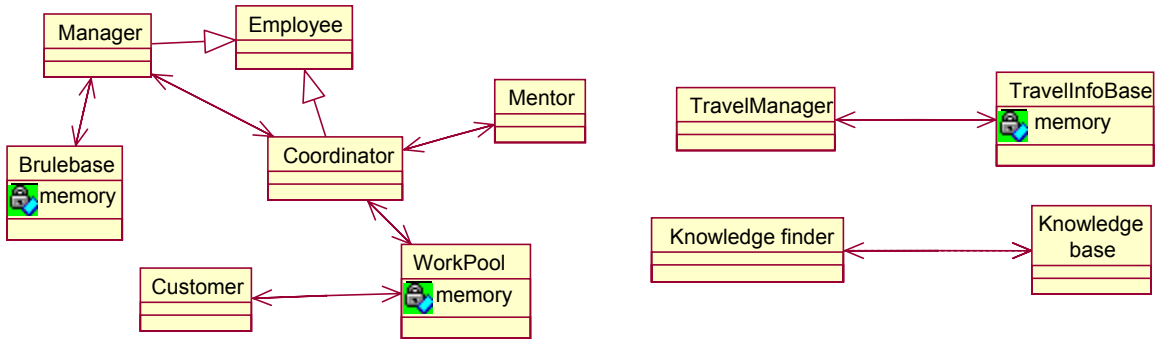

Figure 4: Role models for the telephone repair service teams case study

5.1 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 [8] a way to identify roles in an application domain is to start from 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 (Figure 3):

- *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 of expertise for complex telephone repair tasks.

Each use case has a number of high-level goals depicted in Figure 3. The behaviour leading to achieving these goals can be modelled by appropriate roles. Hence, the following roles can be identified (Figure 4):

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. In 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 store travel information from various travel resources i.e. GPS and traffic databases.
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.

5.2 Specifying Design Constraints

In Figure 5, compositional constraints for the roles described in Section 5.1 are specified in *RCL* (Role Constraint Language).

RCL is simple declarative constraint language that was introduced to represent design constraints on agent and role characteristics. The use of *RCL* in Figure 5 is self-explanatory and therefore in this paper no further description of *RCL* will be give. *RCL* is described in detail in [6].

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*.

```
/* 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 5: Compositional constraints for the telephone repair service teams case study

Part of the definition of the characteristics of the *Manager* role is shown in more detail in Figure 5. 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 to the *WorkerAssistant* role. The *WorkerAssistant* role requires some memory to store intermediate synchronisation results — as specified in Figure 5.

An example of a non-functional requirement is to limit to the memory each agent could occupy. In this case study, agents supporting field engineers should be able to operate in PDAs with limited amount of memory. This is modelled as a general design constraint on the performance variable memory (Figure 5). The agent types of an agent organisation satisfying the above design constraints are depicted in Figure 6.

6. CONCLUSIONS AND FURTHER WORK

Existing approaches to agent organisation design do not pay enough attention to semi-automating the transformation of analysis into design, nor do they consider non-functional requirements on design time. The semi-automatic approach described in this paper addresses these concerns by extending the definition of role to include performance parameters and by defining a simple role algebra to facilitate automatic allocation of roles to agents. This approach enables reuse of organisational design patterns by representing them as role models being able to be manipulated considering the proposed role algebra.

However, there are issues that have not been addressed in this paper. For example, agents can play different roles in different contexts and hence the possible contexts should be considered when designing agent organisations. It is planned to extend the semi-automatic approach to consider role playing within some context in the near future. As a longer-term research task it is planned to use the role algebra to study the impact of dynamically allocating and de-allocating roles to agents on run-time.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] Andersen, E., Conceptual Modelling of Objects: a role modelling approach. PhD Thesis, Dept. of Computer Science, University of Oslo, Oslo, 1997.

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

[3] Depke, R., Heckel, R. and Kuster, J.M. Improving the Agent-oriented Modeling Process by Roles, in Proceedings of Autonomous Agents'01 (Montreal, Canada, 2001), ACM Press.

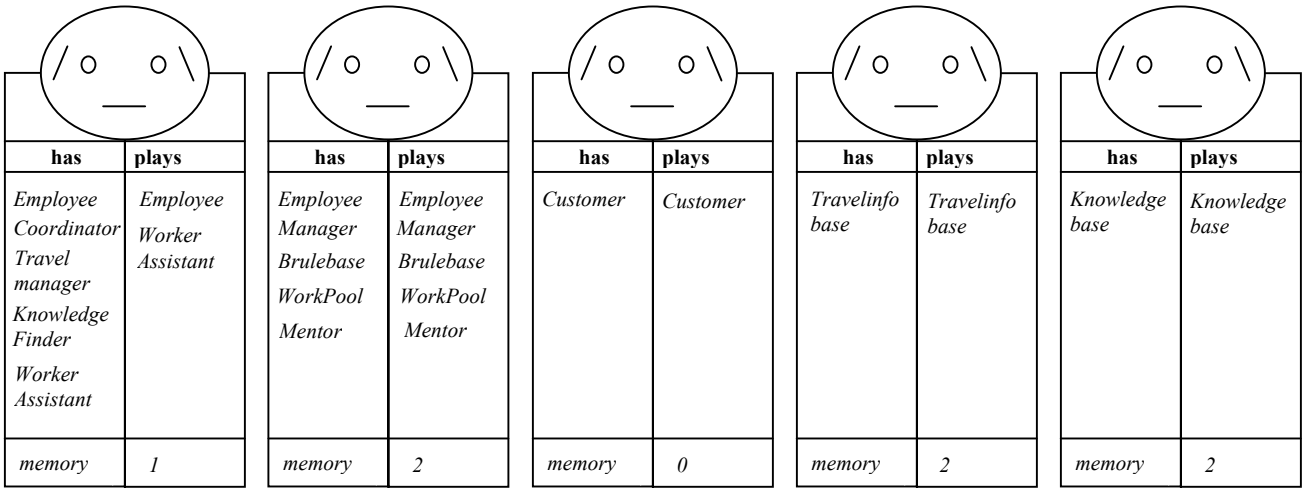

Figure 6: Agent types for the telephone repair service teams case study

[4] Evans, R., MESSAGE: Methodology for Engineering Systems of Software Agents. BT Labs, Ipswich, 2000. http://www.labs.bt.com/projects/agents.htm.

[5] Ferber, J. and Gutknecht, O. A meta-model for the analysis and design of organizations of Multi-Agent systems, in Proceedings of ICMAS'98 (Paris, France, 1998), IEEE Press.

[6] Karageorgos, A., Thompson, S. and Mehandjiev, N. Using Role Algebra for Semi-Automatic Agent System Design: A Case Study. Dept. of Computation, UMIST, Manchester, 2001. http://www.co.umist.ac.uk/~mcaihak2/respapers.html.

[7] Kendall, E.A., Role models - patterns of agent system analysis and design. BT Tech. Journal, 1999. 17(4), 46-57.

[8] Kendall, E.A. and L. Zhao. Capturing and Structuring Goals, in Proceedings of OOPSLA'98 (1998), ACM Press.

[9] Nwana, H.S., et al., Zeus: A Toolkit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence Journal, 1999. 13(1), 187-203.

[10] Omicini, A. SODA : Societies and Infrastructures in the Analysis and Design of Agent-based Systems, in Proceedings of AOSE'00 (Limerick, Ireland, 2000), Springer Verlag.

[11] Parunak, V., Sauter, J. and Clark, S. Towards the Specification and Design of Industrial Synthetic Ecosystems, in Intelligent Agents IV: Agent Theories, Architectures, and Languages, Singh, M.P., Rao, A. and Wooldridge, M.J., (eds.), 1998, Springer Verlag, 45-59.

[12] Scott, W.R., Organizations: Rational, Natural and Open Systems. Prentice Hall International, New York, 1992.

[13] So, Y.-p. and Durfee, E.H. Designing Organizations for Computational Agents, in Simulating Organizations: Computational Models of institutions and groups, Prietula, M.J., Carley, K.M. and Gasser, L. (eds.), 1998, AAAI Press, 47-64.

[14] Sparkman, C.H., DeLoach, S.A. and Self, A.L. Automated Derivation of Complex Agent Architectures from Analysis Specifications, in Proceedings of AOSE'01 (Montreal, Canada, 2001), Springer Verlag.

[15] Stark, J., et al., ACSOSS: a case study applying the MESSAGE analysis method, BT Labs, Ipswich, 2001. http://www.labs.bt.com/projects/agents.htm.

[16] Thompson, S.G. and Odgers, B.R. Collaborative Personal Agents for Team Working, in Proceedings of AISB'00 (Birmingham, England, 2000).

[17] Wooldridge, M., Jennings, N.R. and Kinny, D. The Gaia methodology for agent-oriented analysis and design. International Journal of Autonomous Agents and Multi-Agent Systems, 2000. 3(3), 285-312.

[18] Yu, L. and Schmid, B.F. A Conceptual Framework for Agent-Oriented and Role-Based Workflow Modelling. in AOIS´99 (Heidelberg, 1999), MIT Press.

[19] Zambonelli, F., Jennings, N.R., and Wooldridge, M. J., Organizational Abstractions for the Analysis and Design of Multi-Agent Systems. In Proceedings AOSE'00 (Limerick, Ireland, 2000), Springer Verlag.