A Role-Based Infrastructure for Customised Agent System Development in Supply Networks

Alexander Hämmerle [1], Anthony Karageorgos [2], Michael Pirker [1], Alois Reitbauer [1], Georg Weichhart [1]

[1] Profactor Produktionsforschungs GmbH
Im Stadtgut A2, 4407 Steyr, Austria
alexander.haemmerle@profactor.at

[2] University of Thessaly, Dept. of Computer & Communication Engineering
37 Glavani - 28th October Str
382 21 Volos - Greece
karageorgos@computer.org

Abstract – *This paper presents a methodology for generic role identification and role reuse together with an infrastructure enabling rapid development of role-based multi-agent system (MAS) applications. The infrastructure includes a FIPA compliant runtime environment for role execution as well as a library of generic roles and interaction protocols (IPs) capturing generic agent behaviours and communication abilities. The library has been generated by applying the methodology for generic role identification to a set of real-world scenarios from three different industrial domains, encompassing different aspects of collaboration in supply networks. Developers build their applications for the runtime environment by reusing and extending roles and IPs from this library.*

Keywords: *Next generation infrastructures, collaborative intelligent systems, systems modelling and control.*

1 Introduction

As pointed out by [1, 16, 18], MAS will provide the enabling technology for next generation eBusiness solutions. Application perspectives range from business-to-consumer domains like travel agencies and retailing to full integration of supply networks for virtual and trans-national enterprises.

In the context of MAS analysis and design the role concept has been extensively used. To this aim a number of role catalogues have been suggested [12, 21] and the reuse of existing roles is common in many role-based MAS engineering approaches [15, 8]. However, there are still some open questions in generic role identification and use both at methodological and at technical infrastructure level. This paper aims to contribute towards this direction by proposing a systematic method for generic role identification and customised reuse and a flexible infrastructure for rapid role-based MAS application development.

The remainder of the paper is structured as follows. In section 2 we will outline the industrial grounding of our work. Section 3 will introduce the notion of roles and tasks, followed by the description of a systematic method for generic role selection and reuse. An exemplary description of the results of applying the systematic method to the scenarios from section 2 is given in section 4. The runtime infrastructure for roles is described in section 5, whereas section 6 deals with related work. In section 7 we present our conclusions and an outlook on future research activities.

2 Industrial use cases

The development of the role-based infrastructure is based on real-world scenarios from three different industrial domains, encompassing different aspects of collaboration in supply networks. In the following we outline the different scenarios, highlighting their key requirements.

1st tier supplier - automotive domain

For a 1st tier supplier to survive in the automotive domain it is of paramount importance to keep (or even set) the pace of innovation cycles with the OEMs. Due to the complexity of the product (a car) the 1st tier supplier is not able to provide its module in isolation, but is itself just a node in a complex supply network, where companies provide specific components for the module of the 1st tier supplier. That is to say if we talk about the innovative character of the supplier, we actually mean the innovative character of a whole supply network. The *discovery of innovative suppliers* is thus the key requirement in the automotive domain scenario.

Service provider / broker - logistics domain

The scenarios in the logistics domain give rise to a series of requirements. *Request handling* supports the human user in the processing of customer requests for logistics services. If an offer is accepted by the customer an order is created and dispatched, resulting in the requirements for *order tracking and forecasting delays.* If a service broker is confronted with a customer request, he has to select an appropriate service (provider), based on the customer requirements. Hence *service selection* is the main requirement in the service broker scenario.

Distributed enterprise - manufacturing domain

The key requirement in the manufacturing domain scenario is *resource coordination* (including process planning and resource allocation) across geographically distributed plants. The coordination of internal manufacturing resources and external logistics providers (for transport between plants) facilitates efficient production plans for the distributed enterprise.

3 A systematic method for generic role selection and reuse

We propose a role-based approach for MAS application development, which gives particular emphasis on the reuse of existing roles and protocols. The approach is supported by an agent-based middleware infrastructure, which allows role selection and use both on design and on run-time. The approach includes a method for generic role identification for a given application domain coupled with a method for rapid MAS application development based on reuse and customisation of the generic role components previously identified. These methods are described in turn after providing some background definitions.

3.1 Modelling agent behaviour using roles and tasks

Roles are basic building blocks for a number of modelling approaches. For example, roles are used in organisational theory [23] to represent positions and responsibilities in human organisations and in object-oriented software engineering to represent the functionality of software objects [2]. Furthermore, roles are considered particularly suitable for modelling the behaviour of software agents due to their ability to represent generalised behaviour in organisational context [13]. Roles in multi-agent systems are mainly defined in a manner similar to that of organisational roles referring to a position and a set of associated responsibilities (for example privileges and obligations) in an organisation [9]. Some additional characteristics are also attributed to agent roles, such as capabilities to conduct planning, co-ordination and negotiation [13].

Role characteristics	Description
Context	Describes the application context in which the role is applicable.
Goals/Responsibilities	Refer to what the role aims to achieve within a particular context
Tasks	Represent specific tasks the role can carry out.
Capabilities/Privileges	Properties that enable/ facilitate role behaviour.

Table 1: Role characteristics

Multi-agent system engineering methodologies increasingly use roles as basic behavioural abstractions. These methodologies acknowledge the need to identify and reuse generic role-based components [13, 2], but they do not provide any systematic methods and supporting infrastructures for this purpose.

In this work, we pay particular attention to the notion of *task* as a fundamental representation of atomic behaviour used to compose the role behaviour. We consider roles as task carriers that execute tasks aiming to fulfil their goals. There is a strong relation between roles and tasks and each task is associated with one role. Based on similar notations used for listing class characteristics in object-oriented software construction [11], we can use a table similar to Table 1 to list the characteristics of roles. Role characteristics are described in more detail below:

- *Context*: Context refers to the application context where the behaviour the role represents belongs. For example, the *"AGV_Vehicle_Operator"* role can refer to different behaviours in the contexts of harbour or military operations respectively.

- *Goals/Responsibilities*. This is what the role aims to achieve, for example a goal of the "AGV_Vehicle_Operator" role can be: "To park the AGV at an appropriate place in the service yard when needed".

- *Tasks.* Tasks are units of behaviour that have a purpose and a specific outcome. For example, "Driving an AGV to the unloading track". There is a strong relation between goals and tasks as each task is associated with a goal. However, this is not a one-to-one relation as more than one task can correspond to a single goal. For example, "to operate an AGV" is a goal that can include a number of tasks such as "starting the AGV", "stopping the AGV" and "loading/unloading the AGV". The exact

correspondence between tasks and goals is a modelling decision and it depends on the role designer and the application requirements.

Tasks can invoke other tasks during their execution and this gives rise to a task hierarchy. An example of a composite task hierarchy is given in Fig 1. For the composite task "Register to conference" to be executed, tasks "Register to conference sessions" and "Register to conference accommodation" need to be executed at some point and each one of them in turn needs to invoke the task "pay using credit card" to pay for the respective fee. The latter is also an example of task parameterisation showing that the exact task execution can be depended on a number of constraints and parameters, which are checked and/or initialised upon task invocation.

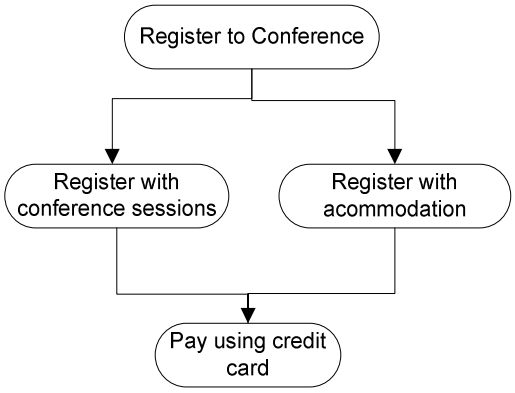

Figure 1: Composite task hierarchy

3.2 Generic role identification

To identify generic roles and interaction patterns in an application domain, we combine techniques from role-based modelling in MAS [13, 8] and role engineering [17] augmented with ideas used in identifying generic patterns of interacting components in software engineering [24, 19]. Our view is to base the generic role identification on the identification of generic tasks, which we can derive from use cases. In particular, for a given application domain we propose the following steps:

1. Describe the application requirements with use cases: This is done in a similar manner as in the analysis phase of many other software engineering methodologies. Requirements are described by use cases (both in diagrammatic and textual form) that are obtained either through common requirements elicitation typical of the object-oriented methods [11]or through the application of a scenario-based method such as GBRAM [3]. Subsequently, use case diagrams are restructured so that any repeated use case functionality will be separated.

2. Identify tasks and their characteristics: This is done using scenarios. For each use case a number of representative scenarios which cover the functionality (the alternative flow paths) described in the use case is selected. Particular attention must be paid to points where tasks invoke other tasks. Normally, this will have been already indicated in the use case model by associating the use case including the tasks under consideration with the appropriate use cases via "uses" or "extends" relationships. At this point, any necessary task hierarchy graphs can be drawn as needed. Furthermore, the interactions and steps relevant with the execution of each task are described in a high-level manner such as by a textual description. For example, for the "Pay using credit card" task mentioned in Section 3.1 all steps that need to be taken, such as amount specification, payment authorisation, card details validation, and actual payment, will be described.

3. Identify suitable roles and associate them with tasks: Tasks are executed by appropriate roles. Furthermore, any interactions related with tasks take place between a number interacting parties. In our view, task executors and protocol interacting parties are represented by roles. Therefore, a natural starting point for candidate role identification is to list the actors associated with the use cases. However, actor names are not sufficient for describing all roles possibly involved in execution and interactions related with a particular task, Therefore, further candidate roles are identified by applying known methods for identifying objects in object oriented programming [11]. For example, we enumerate all nouns in the use case and we further check words with *–er*, *-ist* or *–or* suffix in the requirement specification. In the latter step particular attention should be paid to any candidate roles representing organisational positions or describing organisational relationships in the domain of the application as it is important for the information system to mirror the organisational structure of the business system it supports [12]. For example, in a MAS supporting execution of a business process we most probably need to have the role of *"Business_Manager"* assigned to an agent that oversees the operations carried out by a number of other agents with closely related goals.

4. Create goal tree and task groups: We construct the goal tree based on the use case diagrams. Using the goal tree makes is easier to see which goals are related with each other. Based on the related goals we group the respective tasks to task groups. Tasks belonging to the same task group will be assigned to the same role while repeated goals will give rise to generic tasks and roles.

5. Identify higher-level roles and role relations: Introduce appropriate higher-level roles and associate them with task (and hence goal) groups. To identify higher-level roles we apply the techniques referred in step

3 and we further try to synthesise appropriate role names from the names of the roles corresponding to tasks belonging to task groups. This will give rise to generalisation and aggregation relationships between roles which need to be noted. However, further relations between roles may exist, for example two roles cannot be played simultaneously.

6. Identify generic roles: In this step we examine which roles appear more than once (both low and high level), we characterise them as generic and we store them together with their respective sub-roles and related task hierarchies in the library.

The above process (and particularly steps 4-6) will be repeated as many times as needed to refine the identified set of generic roles and tasks.

3.3 Reuse-based method for rapid MAS development

Considering that the above steps have been applied and a library of generic roles and tasks has been generated, it is straightforward to reuse them for rapid application development. The following steps are proposed:

1. Describe the application domain requirements: This is done in a similar manner as in the generic role identification process described above. Again, the use case diagrams will be refined so that "extended" or "used" use cases will be clearly distinguishable.

2. Create use case goal tree: Starting from the refined use case diagram we construct the goal tree. In this tree, related goals are grouped together to form goal groups.

3. Select and customise generic roles: Select any roles from the role library that have similar goals or goal groups. Subsequently, customise the characteristics of the selected generic roles to fit exactly with the application requirements. This will involve adding, deleting or overriding generic role tasks.

4. Identify further application tasks and roles: Starting from the refined goal tree, we cross out the goals that have been corresponded to generic roles from the role library and for the remaining goals we introduce appropriate application specific tasks to fulfil them. This is done with the help of scenarios drawn from the use cases as was described in Step 2 of the method for generic role identification described in Section 3.2 above. Subsequently, for the application specific tasks identified in the previous step we introduce suitable application roles similarly to the procedure described in Step 3 in Section 3.2.

5. Bind roles to agents: Roles are assigned to agents following heuristic rules such as trying to maintain low coupling and high cohesion between agent components. We can consider that for each role, the appropriate agent may exist (if we have determined an agent structure) or not. Inexistent agents are related with roles and thus can be generated from roles.

4 Generic roles and protocols

When applying the systematic method described in 3.2 to the industrial scenarios from chapter 2 we realised that the resulting roles are applicable not just to the considered domains. Due to their generic nature the roles are suitable for a broad range of business applications, ranging from the operative level (like order tracking) to the strategic level (like the discovery of innovative suppliers). In the following we provide an exemplary description of generic roles, illustrating their level of abstraction and their importance for application development. We avoid a lengthy enumeration of role specifications, which is outside the scope of this paper.

4.1 Roles

In our exemplary description we regard three generic roles. The first role is called *Seeker*. A seeker has only one task which is to seek for an object fulfilling a certain condition. The second role is a *Decision Taker***.** Decision Takers can be either asked to agree on a proposed decision or to select from a set of proposals those which are feasible. For coordination purposes we will use a *Controller* able to execute a plan, which is generated outside the Controller.

To illustrate the importance of the above roles for application development we refer to two simple scenarios from chapter 2. The first scenario deals with process planning and is a subset of resource coordination in the manufacturing domain. When a manufacturing order is received by an employee, he takes the product requirements and searches for possible process plans in a database with historic process planning data. If several process plans are feasible, a decision has to be made about the most preferable one.

The second scenario is the search for innovative suppliers in the automotive domain. In order to find a novel supplier an employee defines a set of keywords which is to be used for a web query. Then pages matching the query are returned. These pages are checked for their relevance with respect to the automotive domain (which involves a decision process) and forwarded to the employee.

At a first glance these two scenarios do not have much in common. However, the above roles allow a generic description. A Controller first contacts a Seeker to retrieve information (process plans or web pages,

respectively). The Controller forwards the collected information to a Decision Taker, which decides upon the most preferable process plan or the relevance of web pages, respectively.

The simple example demonstrates the broad range of applications for the developed generic roles. For a concrete application the generic roles have to be customised for specific implementation requirements (cf. 5.1). Referring to our example a generic Seeker would become a *Process Plan Searcher* in the first scenario and a W*eb Searcher* in the second one.

4.2 Protocols

In the course of generic role identification we also aimed at the identification of generic interaction patterns between roles. The design goal was a set of simple protocols, necessary and sufficient to cover all interaction patterns in the considered scenarios (cf. section 2). On the basis of FIPA [10] following protocols have been identified:

- FIPA Request Protocol
- FIPA Query Protocol
- FIPA Subscribe Protocol
- FIPA Propose Protocol

For the Request protocol a modified version was developed, in which the requestee is not allowed to refuse the requested action. Additionally one protocol consisting of an *Inform* communicative act and another one including a confirmation by the receiver have been introduced.

5 Runtime infrastructure

On the basis of the FIPA compliant agent platform JADE [4] we designed a framework supporting the concepts presented in the previous sections. Each agent has a built in framework, called the Agent Hull, facilitating the execution of the different roles the agent can play. In detail the Agent Hull manages the:

- Life-cycle of the whole agent (e.g. boot up, shutdown, persistence).
- Configuration of the whole agent.
- Life-cycle of all agent-related roles.
- Proper handling of incoming ACL messages (that may contain new concepts).

This way the Agent Hull acts as a "runtime environment" for all the implemented roles.

5.1 Roles, tasks and interaction Protocols

Roles designed for the Agent Hull only form logical clusters of one or more tasks. Therefore tasks are the place where an agent's business logic is implemented. From this point of view the functionality of an agent is defined by the entirety of its tasks, each of which is assigned to a certain role. Figure 2 summarizes the role and task model supported by the Agent Hull. It illustrates the fact that tasks may invoke other tasks, giving rise to task hierarchies (cf. 3.1).

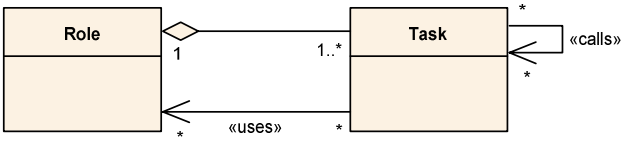

Figure 2: The Agent Hull's role and task model

As tasks implement the business logic, interaction protocols (IPs) are driven by tasks. This means that tasks initiate IPs to perform their work and handle incoming agent communication language (ACL) messages.

Implementing an agent on the basis of the Agent Hull is reduced to the task implementations of all roles the agent can play. Tasks are implemented as JADE behaviours. The implementation of roles and their tasks is supported by pre-defined, generic roles and IPs (as described in section 4). The generic roles are source-code templates in the form of Java abstract classes and interfaces that must be completed by the application developer.

5.2 The Agent Hull

Based on requirements belonging to the role and task centered architecture (as explained throughout this paper), the handling of dynamic ontologies (unknown concepts may appear at any time in incoming ACL messages and must be processed properly by an agent) and the knowledge base centered approach (agent central management of all kinds of agent related information), the Agent Hull architecture shown in Figure 3: The Agent Hull architecture was derived.

In order to increase the scalability (very huge number of agents or constrained devices) of the whole MAS based on the Agent Hull runtime infrastructure we had to change the JADE threading model. In the new model an agent no longer posses a thread of its own, but receives only some processing time within a thread assigned to the agent by an agent platform feature called the agent resource manager. Because an agent owns no single thread the agent (in more detail: the agent's task manager) is called periodically from the JADE agent container in a non-preemptive multitasking way.

The **task manager** provides two main functionalities in the form of a built-in scheduler (accessible for the agent platform via an external interface) and a built-in message dispatcher (a task manager internal functionality). The task manager reimplements the simple JADE scheduler. It maintains the currently active tasks and allocates execution time in a round robin like way to each of them. Each time the task manager is called (by the agent

resource manager) it selects and runs the next task. Additionally the task manager also forwards an incoming ACL message (i.e. a message the agent received) to the appropriate task. First, the task manager actively gets a message from the JADE message queue and processes it by means of a reimplementation of the JADE content manager. Second, the task manager identifies by means of the rule engine an appropriate task that should receive this message.

The **content manager** is a reimplementation of jade.content.ContentManager extended with ACL message scanning capabilities to deal with unknown concepts, partly by means of an ontology agent [5]. The task manager calls the content manager for each received ACL message to get the corresponding message object, which contains both all useful slots and the parsed message content of the original ACL message.

The **rule engine** uses rules of the form "if <left side> then <right side>" to identify the target task for an incoming ACL message. Agent state information and the parsed content contained within the message object are combined to find a rule with a matching <left side>. The <right side> of such a matching rule then references the target task. To avoid ambiguities, rules are ordered within the rule engine according to their priorities.

The **agent life cycle manager** is responsible for the whole agent life cycle and is accessible for the JADE agent container. It initializes and shuts down all other agent hull components (in the following order: knowledge base, rule engine, content manager, role class publisher, task manager, agent administration). It also loads at startup or during runtime agent roles. Loading a role implies loading the rules belonging to the role (to be processed by the rule engine) and eventually registering the role and (some of) its tasks with the DF. At agent shutdown the agent life cycle manager deregisters the roles and tasks. Additionally it controls the persistence functionality of all agent hull components.

The **knowledge base** stores any information belonging to the agent, like agent-global runtime information, role specific data and conversation status of interaction protocols.

The **role class publisher** (de)registers roles and tasks with an agent platform's directory facilitator.

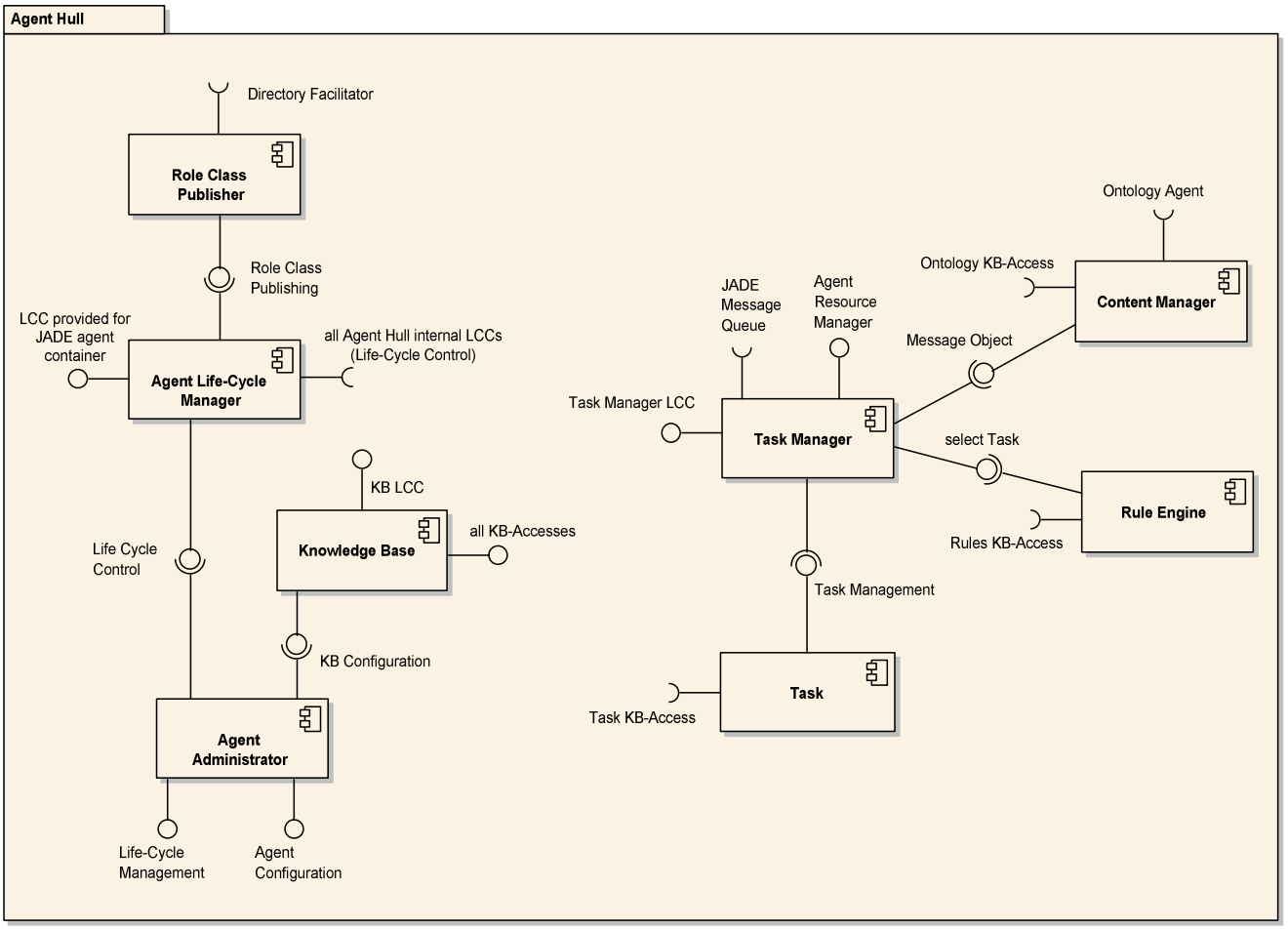

Figure 3: The Agent Hull architecture

The **agent administrator** provides user access to the agent like for example via GUIs.

6 Related work

The concept of reusing generic roles appears in many modelling methodologies in software engineering [20] and particularly in MAS development [13, 8]. However, there are two major differences between these works and the work described here: A systematic method for generic role identification and reuse and an infrastructure supporting storing retrieving and reuse of generic roles and protocols.

With respect to methodological issues most role-based MAS development methodologies do not provide steps for role identification at all [25]. In others, roles are broadly considered as originating from use cases [6] in an ad-hoc manner and reusing them is not explicitly considered as an option. Furthermore there is a problem with specifying the semantics of roles [14] and this impedes developing appropriate tools for generic role management and reuse.

Considering the infrastructural support the situation is even worst. Although there have been efforts to store and reuse roles in relevant areas such as Role-Based Access Control [22], to our knowledge there is currently no existing infrastructure for reusing and customising roles in MAS development except agentTool [7]. However, the MaBE middleware and agentTool have a number of important differences: agentTool focuses purely on a software engineering view of roles, for example it does not support modelling of organisational relations using roles. Furthermore, the approach presented in this paper supports a consistent way of designing, implementing *and* executing role based MAS applications. In contrast to agentTool we have not only concentrated on the analysis and design phase by providing a CASE tool supporting the development of MAS applications. We also provide a runtime infrastructure for MAS applications that have been implemented with (customized) generic roles of the given role library. Additionally our approach strictly complies with FIPA standards by realizing the runtime infrastructure on top of the wide-spread JADE agent platform and by adhering to FIPA defined interaction protocols and communicative acts.

7 Conclusion and outlook

In the course of applying the methodologies described in section 3 to the scenarios from section 2 we realised that our role-based design approach is very intuitive and easy to handle even for people with little background in agent-oriented design. Another lesson learned is the fact that a limited set of simple protocols is sufficient to model interaction patterns occuring in a rich set of business scenarios situated in different domains. However, a major extension of the role-based infrastructure will be a GUI enhanced toolkit, providing a user-friendly environment for rapid construction of role-based agents. This toolkit, together with the ongoing implementation of the Agent Hull, is our R&D focus for the near future.

Acknowledgement

Parts of this work have been conducted in the course of the R&D project MaBE (Multiagent Business Environment, http://www.mabe-project.com), funded by the European Commission under the GROWTH program.

References

[1] E. Adams, “Where are Agents Going in eMarkets?”, Technical Report, Lante Corporation, July 2000.

[2] E. P. Andersen, “Conceptual Modelling of Objects: A Role Modelling Approach”, in Dept of Computer Science. 1997, University of Oslo: Oslo, Norway. p. 333.

[3] A. I. Antón, *Goal Identification and Refinement in the Specification of Software-Based Information Systems*. 1997, Georgia Institute of Technology. p. 261.

[4] F. Bellifemine, A. Poggi, and G. Rimassa. “Developing multiagent systems with JADE”, Proc. Intelligent Agents VII: 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL), number 1986 in Lecture Notes in Computer Science, pp 89-103, Springer, Heidelberg, 2001.

[5] M. Carpenter, A. Gledson, and N. Mehandjiev, “Implementing Dynamic Ontologies in Agent-Based Business Support Systems”, AAMAS Workshop on Agent-Oriented Information Systems, 2004.

[6] M. Cossentino and C. Potts, “PASSI: a Process for Specifying and Implementing Multi-Agent Systems Using UML”.

[7] S. A. DeLoach, “Analysis and Design using MaSE and agentTool”. 12th Midwest AI and Cognitive Science Conference (MAICS 2001), Oxford, Ohio, March 31 - April 1, 2001.

[8] S. A. DeLoach, M.F. Wood, and C.H. Sparkman, “Multi-Agent Systems Engineering”. International Journal of Software Engineering and Knowledge Engineering, 2001. 11(3): p. 231-258.

[9] J. Ferber and O. Gutknecht, “A meta-model for the analysis and design of organisations of Multi-Agent systems”. in Proceedings of the International Conference in Multi-Agent Systems (ICMAS 98). 1998. Paris, France: IEEE Computer Society Press.

[10] Foundation for Intelligent Physical Agents, ”FIPA Agent Management Specification”, December 2002. http://www.fipa.org/specs/fipa00023.

[11] I. Jacobson, “Object-Oriented Software Engineering: A Use Case Driven Approach”. 1992: Addison-Wesley Professional. 552.

[12] N. R. Jennings, “On Agent-based Software Engineering”. Artificial Intelligence, 2000. 117: p. 277-296.

[13] E. A. Kendall, “Role models - patterns of agent system analysis and design”. BT Technology Journal, 1999. 17(4): p. 46-57.

[14] E. A. Kendall, “Agent Analysis and Design with Role Models”, in Volume 1: Overview. 1999, BT Exact Technologies: Martlesham Heath, UK. p. 89.

[15] E. A. Kendall, “Agent Software Engineering with Role Modelling”. Agent-Oriented Software Engineering I, First International Workshop (AOSE 2000), Limerick, Ireland, ed. P. Ciancarini and M.J. Wooldridge. Vol. 1957. 2001, Berlin: Springer Verlag. 163-169.

[16] M. Luck, P. McBurney and C. Priest, “Agent Technology: Enabling Next Generation Computing - A Roadmap for Agent Based Computing”, AgentLink II, 2003.

[17] G. Neumann and M. Strembeck, “A Scenario-driven Role Engineering Process for Functional RBAC Roles”, in Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT’02). 2002. p. 33-42.

[18] S. Osborn, “The role of agents in business to business (B2B) electronic commerce”, AgentLink Newsletter 6, January 2001, pp. 6-8.

[19] I. Philippow, D. Streitferdt, and M. Riebisch, “Design Pattern Recovery in Architectures for Supporting Product Line Development and Application, in Modelling Variability for Object-Oriented Product Lines”, M. Riebisch, J.O. Coplien, and D. Streitferdt, Editors. 2003, BookOnDemand Publ. Co: Norderstedt. p. 42-57.

[20] T. Reenskaug, P. Wold, and O.A. Lehne, “Working with Objects”, The OOram Software Engineering Method. 1996, Greenwich: Manning Publications. 420.

[21] D. Richle, "A Role-Based Design Pattern Catalog of Atomic and Composite Patterns Structured by Pattern Purpose". 1997, Ubilab, Union Bank of Switzerland: Zurich. p. 48.

[22] R. S. Sandhu et al., "Role-Based Access Control Models", IEEE Computer, Volume 29, No. 2, pp. 38-47, 1996.

[23] W. R. Scott, "Organisations: Rational, Natural and Open Systems". 2003, New York, NY: Prentice Hall International. 430.

[24] F. Shull et al., "An inductive method for discovering design patterns from object-oriented software systems." 1996, University of Maryland.

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