Developing Multi-Agent Systems with Automatic Agent Generation and Dynamic Task Allocation Mechanisms

Xiaoqin Zhang
Computer and Information Science Department
University of Massachusetts at Dartmouth
x2zhang@umassd.edu

Haiping Xu
Computer and Information Science Department
University of Massachusetts at Dartmouth
hxu@umassd.edu

Bhavesh Shrestha
Computer and Information Science Department
University of Massachusetts at Dartmouth
g_bshrestha@umassd.edu

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design, Standardization, Verification.

Keywords

Role-Based Agent Development, Multi-Agent Systems, Motivations, Role-Agent Mapping, Task Allocations

1. INTRODUCTION

Multi-Agent System (MAS) is a suitable programming paradigm for distributed information systems and applications. We have been working on a set of technologies and mechanisms to ease and formalize the development of MAS, and to increase its reliability and reuse-ability too. We aim to cover the analysis and modeling, design and implementation phases. The first goal is to **separate concerns**. We have proposed a three-layered development process to separate the multiple issues in a multi-agent system, while some of them are application-dependent, others are not; some of them are platform-dependent and others are not. We have also aimed to separate the domain knowledge and the intelligent problem-solving capabilities. We adapt a role-based modeling approach, conceptual roles are defined with the domain related knowledge, such as goals, permissions, organizational relationship, and interaction protocols, etc; where agent is a concrete entity equipped with motivations, resources and problem-solving capabilities.

The second goal is to **automate the agent generation process**, while utilizing the existing tools and mechanisms as much as possible. We propose to create agents using a drag-and-drop mechanism where the user can select components to plug in the system depending on the application requirement. We adapt a quantitative model of motivation named MQ framework [4]. Based on this MQ framework, the agent can perform a quantitative reasoning on how important a role instance is given its preference, its utility function and its current achievement. In the definition of a role, we introduces a RTÆMS language (Role-Based Task Analyzing, environment Modeling, and Simulation) to represent the domain knowledge about how to achieve a goal. RTÆMS language is an extension of TÆMS language [1], a hierarchical task network representation language with task inter-relationships and quantitative descriptions of different alternatives to achieve a goal. When an agent takes a role instance, it has access to this RTÆMS representation of the goal. As a result, the existing planning/scheduling [5] and coordination [2] mechanisms based on TÆMS language can easily be exploited by the agent.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

2. AUTOMATIC AGENT GENERATION PROCESS

We have developed a tool to support the automatic agent generation process. This tool is created by extending the current JAF framework [3] developed by MAS lab at UMass Amherst. This tool includes a graphic user interface, which can be used to create new agents, modify existing agents, run agents and delete agents. A screen shot of the graphic user interface is shown in Figure 1.

Agent class is defined by a set of attributes, motivations, utility function, and a set of reasoning mechanisms and execution mechanisms [6]. The user can create a new agent (class) through this interface. The user can define a variety of attributes including name, qualification, and other parameters to control the agent execution process such as log file name and log level. Qualification is an attribute that describes a particular capability or certificate this agent class owns, which is used in the role-agent mapping process to decide whether an agent is qualified for a particular role.

3. DYNAMIC TASK ALLOCATION THROUGH ROLE-AGENT MAPPING

In RADE framework, agents can dynamically choose the role instances, and role instances can be created dynamically too. In the development phases, roles and agents are designed separately. In the implementing phases, agents are created by users. In addition, there is a *role space* component built in the system to manage role instances:

When the system execution starts, one or more role in-

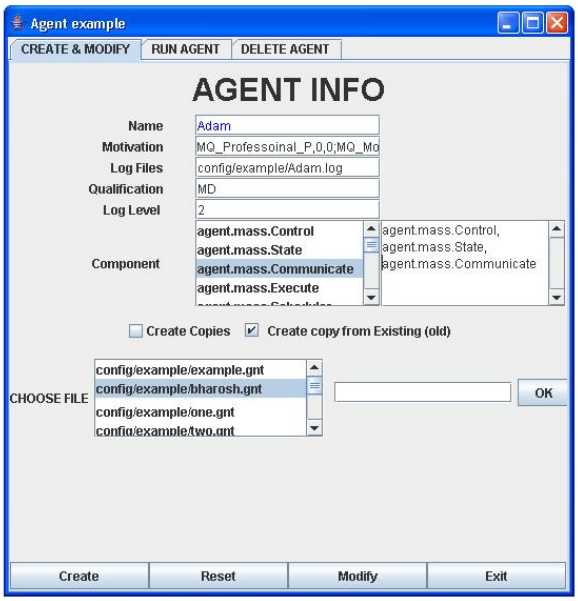

Figure 1: Automatic Agent Generation Interface

stances are created by the human user. Those agents who are interested in taking a particular role instance send messages to the role space. The role space then checks the qualification of the agents. If an agent is currently taking other role instances, it checks if this role instance is compatible with the other role instances, according to the incompatible relationship defined between role classes.

After this process, a list of qualified agents is sent to the the creator of this role instance (in this case, the creator is the human user, it can be an agent too). The creator then selects one agent from this list to take the role instance. This selection is totally based on the creator's preference, the user can define different criteria for the selection, such as based on the profile of the candidate agent, or the experience of previous interaction with the candidate agent.

When an agent takes a role instance, it checks the goals that belong to this role instance and decides if more role instances need to be created to carry the subgoals or to achieve some necessary preconditions. If this is the case, more role instances will be created and posted in the role spaces. The process described above is repeated until no more role instances are created.

An agent decides whether it is interested in a role instance by checking if some of the goals that belong to the role instance match the agent's motivation. A goal G matches agent A's motivation if and only if:
$\exists\, MQ_x \in MQPS(G), \exists\, MQ_y \in Motivations(A)$, $MQ_x.type == MQ_y.type$
According to the above definition, there may be multiple role instances an agent is interested at the same time. How much the agent is interested in a particular role instance depends on the type and number of units of MQ associated with the goal that belongs to this role instance, the agent's preference on different MQs given its current MQ accumulations. and the agent's resource and capability.

Since each goal defined in a role instance essentially represents a task to be accomplished, so the role-agent mapping process is a task allocation process. In this process, the

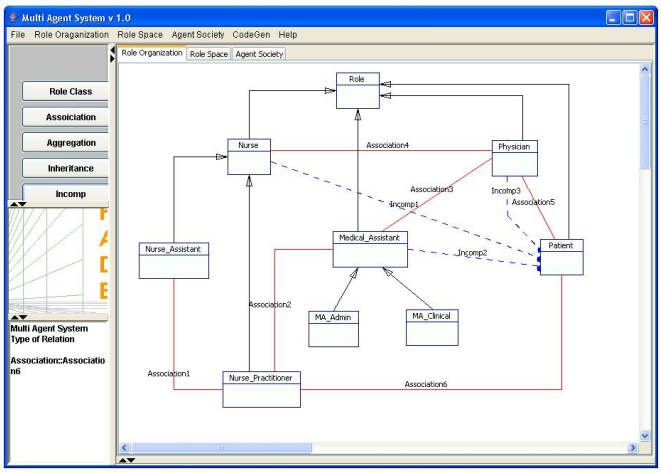

Figure 2: RADE Interface for Creating Roles

agent decides which task it would like to take depending on the user-defined preference functions, its previous experience of task accomplishment and its resource limitation. On the other hand, which agent is chosen to perform this task also depends on the qualification requirement, the organizational rules (represented as the incompatibility relationship) and other dynamic issues such as the agent's previous experience.

4. HEALTH CARE APPLICATION DOMAIN

4.1 Role Definition

Figure 2 shows the RADE interface for user to create role classes and define the interrelationships among role classes. In this example, the interrelationships include *inheritance, association* and *incompatibility*. Each role is defined with a goal, a plan tree, a motivational quantity production set (MQPS), a certificate and other attributes. A goal is a task this role needs to accomplish, and the plan tree specifies the domain knowledge of how to accomplish this goal in terms of decomposing it as sub-goals.

ROLE: Physician
GOAL: Provide Cure
MQPS: (MQ_professional_P, p1), (MQ_moral_P, p2), (MQ_experience_P, p3)
CERTIFICATE: MD (Doctor of Medicine)

For example, Physician role is defined with a goal to *provide care*. The **plan tree** shown in Figure 3 provides domain knowledge of how to accomplish this goal. The **MQPS** specifies the type and the number of units of motivational quantities can be collected by the agent after it accomplishes the goal defined in the role. The **Certificate** defined in the role describes the qualification requirement for this role. This role can only be taken by an agent who has this specified certificate. For example, *Physician* role is defined with a certificate of MD (Medical Doctor).

4.2 Agent Definition

As Figure 1 shows, a user creates an assistant agent named Adam. The user specifies his preference on choosing tasks by defining the motivation of this agent. The user specifies three long-term goals: professional achievement, moral achievement and experience achievement, as a physician,

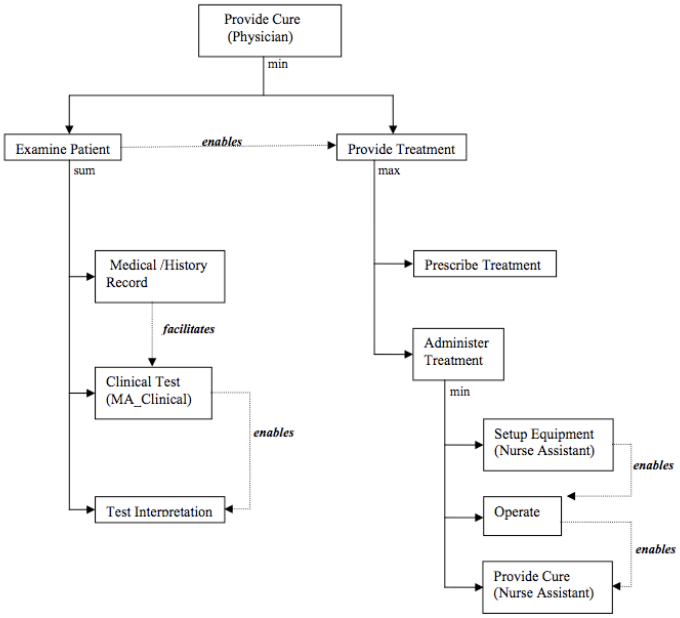

Figure 3: Plan Tree definition for Physician Role

Table 1: Agent's Motivation

MQ Type	Function Index	Initial Amount
MQ_Professional_P	0	0
MQ_Moral_P	1	1
MQ_Experience_P	2	2

which are represented by three types of MQs shown in Table 1 . The function index specifies a utility function that maps a certain number of units of MQ of this type into the agent's local utility. Since the function can be a non-linear function and is also context sensitive, the initial amount of this type MQ is also important. The user also provides this agent his qualification MD so this agent can be qualified for a *Physician* role.

4.3 Runtime Scenario

This system is modeling a hospital organization. When the system is initialized, the system administer creates several *Patient* role instances to express the expected service requirements from patients. When a (real) patient Bryan enters in this hospital for service, an personal assistant agent named Bryan is created for this patient, and this agent takes one *Patient* role instance. When agent Bryan takes the *Patient* role instance, it has one goal to achieve: *Get Cure*. The plan tree of this goal describes that another goal *Provide Care* is a precondition of this goal (*Provide Care* **enables** *Get Cure*), and the goal *Provide Care* belongs to a *Physician* role. Based on this information, agent Bryan sends a request to the role space for creating a *Physician* role instance.

When this *Physician* role instance is created in the role space, all agents who are interested in taking any additional role instances receive a message for this update. After receiving this message, the agent looks at the goal associated with this role instance, especially the MQPS and to see if it matches its own motivation. If the MQPS contains the same type of MQ the agent has in its motivation, the agent is interested in taking this role instance. It is also possible that the role space would receive requests from multiple agents for the same role instance. The role space verifies the qualification of each agent by matching the agent's qualification to the certificate requirement defined in the role class that this role instance belongs to. After that, a short list of agents are sent to agent Bryan, who is the creator of the role instance. Bryan will select one agent from this list to take the role instance according to the criteria defined by its user. Assume that agent Adam is selected by agent Bryan to take the *Physician* role instance, it loads the plan tree of the goal *Provide Care* associate with the *Physician* role. It finds that to accomplish this goal, three subgoals *Clinical Test*, *Setup Equipment* and *Provide Cure* must be accomplished by other roles. In response to this need, agent Adam requests three new role instances to be created in role space, one *MA_Clinical* role instance and two *Nurse Assistant* role instances. Other agents who are interested in these role instances would send requests to the role space, a short list of candidates will be sent the agent Adam for selection. This process will continue until no more new role instance is needed and all role instances have been taken.

After a goal defined in a role instance is accomplished, the agent will collect the MQs as defined in the MQPS of this role instance. The agent will release this role instance, and this role space will delete this role instance. In the system runtime, new role instance is created according to the need to accomplish a certain goal. Agent is mapped to the role instance according to the matching of the motivation, the qualification and the compatibility. Since each role instance is associated with a goal, the mapping process is also a task allocation process. In this process, the agent is reasoning on its local utility achievement, described as its motivation and MQ mapping functions. The domain-related constrains such as qualification and compatibility are defined in the role and monitored by the role space. This implementation realizes the **separation of concern** principle.

5. REFERENCES

[1] K. Decker. TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms. In *Foundations of Distributed Artificial Intelligence, Chapter 16*, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.

[2] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of t he GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

[3] R. Vincent, B. Horling, and V. Lesser. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator. *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems.*, 1887, January 2001.

[4] T. Wagner and V. Lesser. Evolving real-time local agent control for large-scale mas. In J. Meyer and M. Tambe, editors, *Intelligent Agents VIII (Proceedings of ATAL-01)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2002.

[5] T. A. Wagner, A. J. Garvey, and V. R. Lesser. Criteria Directed Task Scheduling. *Journal for Approximate Reasoning (Special Scheduling Issue); a version is also available as UMass Computer Science Technical Report 1997-59*, 19:91–118, January 1998.

[6] X. Zhang and H. Xu. Towards automated development of multi-agent systems using rade. In *Proceedings of The 2006 International Conference on Artificial Intelligence*, pages 44–50, Las Vegas, Nevada,, June 2006. ICAI-06.