Building a Health Care Multi-Agent Simulation System with Role-Based Modeling

Xiaoqin Zhang, Haiping Xu & Bhavesh Shrestha

Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747, U.S.A.
{x2zhang | hxu }@umassd.edu, bhaveshshrestha@gmail.com

ABSTRACT

Multi-Agent System (MAS) is a suitable programming paradigm for simulating and modeling health care systems and applications, where resources, data, control and services are widely distributed. We have developed a multi-agent software prototype to simulate the activities and roles inside a health care system. The prototype is developed using a framework called Role-based Agent Development Environment (RADE). In this chapter, we present an integrated approach for modeling, designing and implementing a multi-agent health care simulation system using RADE. We describe the definition of role classes and agent classes, as well as the automatic agent generation process. We illustrate the coordination problem and present a rule-based coordination approach. In the end, we present a runtime scenario of this health care simulation system, which demonstrates that dynamic task allocation can be achieved through the creation of role instances and the mapping from role instances to agents. This scenario also explains how agents coordinate their activities given their local constraints and interdependence among distributed tasks.

KEYWORDS

Multi-agent systems, agent-based simulation, role-based modeling, coordination, task allocation.

INTRODUCTION

Multi-Agent System (MAS) is a suitable programming paradigm for simulating and modeling health care systems and applications, where resources, data, control and services are widely distributed. We have developed multi-agent software to simulate the activities and roles inside a health care system. Such software can be used to assist the collaborative scheduling of complex tasks that involve multiple personals and resources. In addition, it can be used to study the efficiency of the health care system and the influence of different policies.

However, the application of multi-agent system has been limited by the difficulty of developing agent-based systems, and considerable amount of time and highly experienced programmers are required to develop a multi-agent system. After such system is built, it is also difficult to test and maintain the system because of its complexity. The reusability of such system is low; it is unlikely to use an existing system for another application domain with little or minor change. In this chapter, we will describe a role-based approach to building multi-agent systems for health care simulation and modeling. With this approach, we are able to separate the concern on domain knowledge and the concern on intelligent problem-solving capabilities. In this approach, conceptual roles, such as physicians, nurses and patients are defined with the domain related knowledge including goals, permissions, organizational relationship, and interaction protocols, etc; where an agent is a concrete entity equipped with motivations, resources and problem-solving capabilities, which can be used to represent a real person in a health care system. Each agent can be configured based on different specifications according to the real person's situation and needs. Then the agent instance is dynamically generated for the real person who enters the system.

In this chapter, we will also describe an automated agent generation process, which utilizes 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 into the agent depending on application requirements. We adopt a utility-driven agent architecture with quantitative reasoning capabilities. Besides the logical reasoning on the matching of motivations and the conflicts among different roles, we adapt a quantitative model of motivation named MQ (motivation quantities) framework. Based on the MQ framework, an 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 introduce a formal language called RTÆMS (Role-based Task Analyzing, Environment Modeling, and Simulation) to represent the domain knowledge about how to achieve a goal. RTÆMS language is a hierarchical task network representation language with task interrelationships and quantitative descriptions of different alternatives to achieve a goal. The domain expert can specify how a complicated health service task should be performed with the collaboration of multiple roles inside the system. Each agent is also equipped with the capability for planning, scheduling and cooperation; hence, an agent can schedule its local activities with the consideration of the constraints from other agents. Meanwhile, a user of the system can choose different collaboration rules according to the organizational rules and the specific needs in the system.

In the rest of this chapter, we first discuss related work in several research areas. Afterwards, we describe how to construct a health care simulation system using the approach described above, and show how to define roles and their interrelationships, and how to define agent classes. Then, we present an automatic agent generation tool as well as a rule-based coordination approach. Finally, we use a runtime scenario to demonstrate how new role instances are created, how agents are taking new roles, planning and scheduling their tasks, and collaborating with each other to achieve a complex goal.

BACKGROUND

Researchers have studied a number of approaches for defining and developing autonomous agents and multi-agent system from different directions. Here we discuss related research work in four areas: agent development framework, role-based modeling of agent-based systems, specification of coordination rules, and model-driven development of multi-agent systems.

1. Agent Development Framework

DECAF (Graham, Decker & Mersic, 2003) and JADE (Bellifemine et. al, 2003) are examples of the frameworks that can be used to generate domain specific agents. DECAF (Distributed, Environment-Centered Agent Framework) developed in University of Delaware, is a toolkit to build multi-agent systems. The toolkit provides a stable platform to design, rapidly develop, and execute intelligent agents to achieve solutions in complex software systems. DECAF provides the necessary architectural services of an intelligent agent: communication, planning, scheduling, execution monitoring, coordination, and eventually learning and self-diagnosis. Plan editor is a GUI that provides the interface for control or programming of DECAF agents. In the Plan editor, executable actions are treated as basic building blocks, which can be chained together to achieve a larger and more complex goal in the style of a hierarchical task network. This provides a software component-style programming interface with desirable properties such as component reuse and some design-time error-checking. The chaining of activities can involve traditional looping and if-then-else constructs. This part of DECAF is an extension of the RETSINA (Williamson, Decker & Sycara, 1996) and TÆMS (Decker, 1996). task structure frameworks. Each action of an agent can also have a performance profile, which is used and updated internally by DECAF to provide real-time local scheduling services.

JADE (Java Agent Development Framework) (Bellifemine et. al, 2003) is a software framework fully implemented in Java language distributed by Telecom Italia. It simplifies the implementation of multi-agent systems through a middleware that complies with the FIPA specifications. The agent platform can be distributed across machines and the configuration can be controlled via a remote GUI. The configuration can be changed at runtime by moving agents from one machine to another, when required. The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching. JADE implements a full FIPA communication model, and its components have been clearly distinct and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontology, and finally, transport protocols. Most of the interaction protocols defined by FIPA are available and can be instantiated after defining the application-dependent behaviour of each state of the protocol. Agent management ontology has been implemented, as well as the support for user-defined content languages and ontology that can be implemented, registered with agents, and automatically used by the framework. JADE has also been integrated with JESS, a Java shell of CLIPS, in order to exploit its reasoning capabilities.

The goals of both these frameworks are to develop a modular platform to allow for rapid development of third-party domain agents, and provide a means to quickly develop complete multi-agent solutions using combinations of domain-specific agents and standard middle-agents. These frameworks specify agents in terms of roles they play, and assume that agents do not change their roles at run time. In contrast, we implemented an automated agent generation mechanism using the RADE framework. Using this framework, we can separate the domain knowledge and the intelligent problem solving capabilities. So an agent can be created with intelligent capabilities and motivations, and can take up different roles dynamically.

2 Role-Based Modeling

The related work in the second area is to propose role-based methodology for developing multi-agent systems. Approaches like Gaia (Wooldridge, Jenning, & Kinny, 2000; Zambonelli, Jennings & Wooldridge, 2003) and MaSE (DeLoach, Wood, & Sparkman, 2001) can be used to model multi-agent system societies in terms of organizations or groups composed of a collection of roles related to one another and participating in patterns of interactions with other roles. The agents are then specified in terms of a set of roles they play. These approaches explicitly assume that the inter-agent relationships and the abilities of agents do not change at run-time and that all the agents are explicitly designed to cooperatively achieve common goals.

The Gaia methodology can be used to model both the macro aspect and the micro aspect of a multi-agent system. It covers the analysis phase and the design phase. In the analysis phase, the role model and interaction model are constructed. Based on the analysis models, in the design phase, three models, the agent model, service model and acquaintance model are constructed during the initial design of the system, and then are refined during the detailed design phase using conventional object-oriented methodology. The later version of Gaia (Zambonelli, Jennings & Wooldridge, 2003) extends the former one in order to better suit to open multi-agent systems by introducing two new abstractions: (1) organizational rules (explicit identification of relationships and constraints between roles and protocols), and (2) organizational structures (explicit specification of organizations in terms of their topology and control regime).

The MaSE methodology is a specialization of more traditional software engineering methodologies (DeLoach, Wood, & Sparkman, 2001). During the analysis phase of the MaSE methodology, a set of roles are produced, which describes entities that perform some function within the system. In MaSE, each role is responsible for achieving or helping to achieve specific system goals and sub-goals. During the design phase, agent classes are created according to the roles defined in the analysis phase.

In our approach, the components of role instances and agent instances are loosely coupled, where agents can take or release role instances at runtime without knowing the internal structure of role instances. Thus, role classes and agent classes can be designed and implemented independently.

3 Coordination Rules

The related work in the third area is definition of coordination rules. Projects such as AgenTalk (Kuwabara, Ishida, & Osato, 1995) use scripts and finite state machine to define coordination rules. AgenTalk is a language for describing coordination protocols for multi-agent systems co-developed by NTT Communication Science Laboratories and Ishida Laboratory, Department of Information Science, Kyoto University. It provides an explicit state representation of a protocol, and a finite state machine that allows variables to be used as a basis to describe coordination protocols, called a script. Using this model, states of a protocol are explicitly defined, and actions of an agent can be defined for each state. Protocols can be defined incrementally by extending existing scripts. It provides a programming interface that specifies the portion of a state transition rule that needs to be customized for each agent. The AgenTalk has been implemented in Common Lisp.

In ROPE project (Becht et. al., 1999), cooperation process is built as a separated component from the concrete agents; the ROPE engine provides execution of the cooperation process, which is described as a high-level Petri net class. However, the implementation of ROPE engine is based on shared memory, which is not always feasible for agents that are widely distributed on different machines. Additionally, the cooperation process in ROPE is based on token and transition firing, which is not feasible enough to support more proactive cooperation and collaboration, i.e. agents are able to consider the cooperation and coordination needs when they are planning their own activities.

A set of domain-independent general collaboration mechanisms, Generalized Partial Global Planning (GPGP) (Lesser, et. al. 2004), based on TÆMS language (Decker, 1996) has been developed. We have reused some of GPGP similar mechanisms in RADE (Zhang & Xu, 2006) framework based on RTÆMS language. In framework such as AgenTalk, the emphasis is on the flow of messages and how the dialog between agents is structured. Such

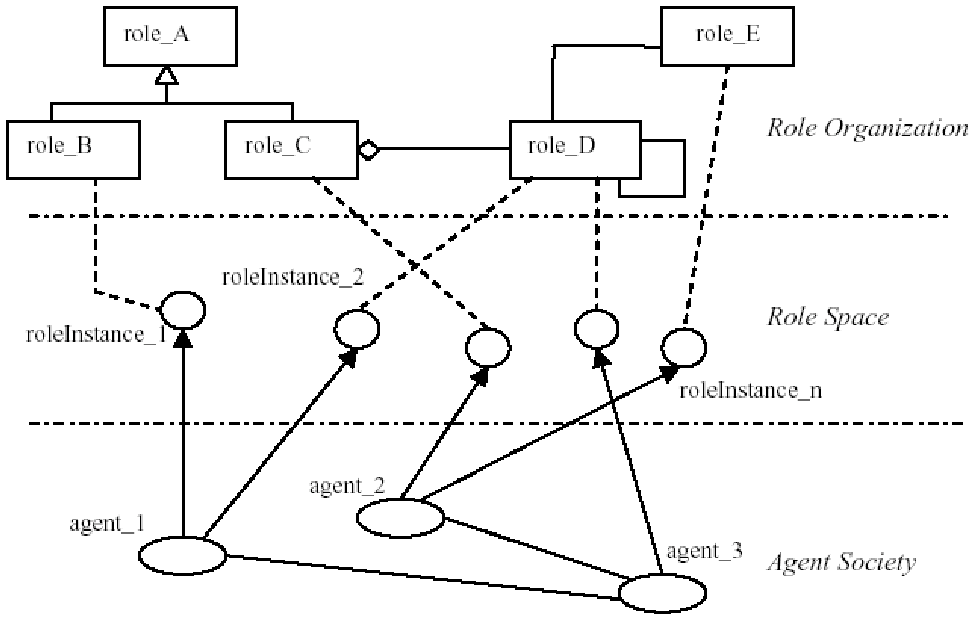


Figure 1. RADE Concept (© [2007], [Journal of Computational Intelligence Theory and Practice]. Used with permission.)

framework combines finite state machines with enhancements. In contrast, GPGP focuses on a domain independent and quantitative evaluation of the interactions among tasks and the dynamic formation of temporal constraints to resolve and to exploit these interactions. Our implementation gives a user the freedom to choose the appropriate coordination rule according to the application domain.

4 Model-Driven Development

Previous work on model-driven development of multi-agent systems can be summarized as follows. Gracanina, Boher and Hincey proposed a model-driven architecture framework as an extension to Cognitive Agent Architecture (COUGAAR) (Gracanin, Bohner & Hinchey, 2004). The Cognitive Agent Architecture is a distributed agent architecture that has been developed primarily for very large-scale, distributed applications that are characterized by hierarchical task decompositions, and as such, it is well suited for autonomic systems. The framework consists of two main parts, General COUGAAR Application Model (GCAM) and General Domain Application Model (GDAM). The GCAM provides representation in its model of the COUGAAR basic constructs, and the GDAM defines the requirements and the detailed design.

Maria, Silva and Lucena (2005) proposed an MDA-based approach to developing multi-agent systems. They first use MAS modeling language (MAS-ML) to model MAS by creating the platform independent models (PIM). Then the MAS-ML models are transformed into UML models using the ASF framework, which defines a set of object-oriented models for MAS entities specified in MAS-ML. The UML models are then transformed into code.

We have proposed three levels of models for developing role-based open multi-agent systems (Xu, Zhang & Patel, 2007), namely AIPIM (Application Independent Platform Independent Model), ASPIM (Application Specific Platform Independent Model), and ASPSM (Application Specific Platform Specific Model), as a refinement process. In each level of the models, role components and agent components are always separated and designed independently. Role instances and agent instances interact with each other only at runtime through an A-R (Agent-Role) mapping mechanism.

ROLE-BASE MODELING APPROACH

The basic idea of the role-based agent development environment (RADE) is illustrated in ***Figure 1***. The top level is the *role organization*, which defines the conceptual roles and their relationships such as inheritance, aggregation, association and incompatibility. In health care systems, conceptual roles represent all possible job titles in the system, such as *physician* and *nurse*. The relationships describe how these roles relate to each other. The second level is the *role space*, which consists of multiple role instances; each role instance is instantiated from a conceptual role dynamically. For example, whenever there is a need to cure a patient, a new physician role instance is created with the goal to cure a patient. A role instance represents the task that needs to be accomplished in the system. The bottom level is the *agent society*, which consists of multiple agent entities. Agent can take or release role instances dynamically, where the mapping from role instances to agents is called A-R mapping, which represents that a real person takes a task in the system.

In an actual software system, agent instances are automatically generated based on the definition of agent classes. Each agent instance is a software entity that performs specific functions and also coordinates and communicates with other agent instances. On the contrary, role classes are defined to incorporate domain knowledge and organizational relationships. Each role class is associated with specific goals and detailed descriptions of how to achieve such goals. The relationships among different role classes also depict the organizational relationships among the real-world entities represented by these roles. Such information is expected to be provided by domain experts rather than software engineers. At system runtime, role instances are created dynamically either by a human user or by an agent when certain goals are needed to be realized. Those role instances mainly carry domain knowledge; however, they do not actually perform any actions like agents. When an agent takes a role instance, the agent uses the knowledge incorporated in the role instance in order to achieve the goals defined in it.

One major advantage of the RADE approach is that it supports the separation of domain knowledge and the agent framework for the simulation system. Any domain knowledge relates to the health care domain can be specified by domain experts through definition of roles and their interrelationships. On the other hand, software engineers are responsible to develop automatic agents that actually perform tasks in the simulation system.

DEFINING ROLES AND ROLE SPACE

The definition of a role class includes the following information:

1. A set of attributes, such as role name and identification.
2. A set of goals; each goal is associated with a plan tree, which is a hierarchal description of the alternatives to accomplish a goal.

3. A set of actions that can be performed by this role, i.e. a *Physician* role can perform an action of *Prescribe Medicine*.

4. Qualification: the requirement needed to take such a role.

5. The permission of this role, which specifies what information and resource are allowed to be access by this role. For instance, a *Physician* role has the permission to access the patient's medical record.

6. A set of protocols, which describe how this role should interact with other roles.

All above information is domain-dependent; hence an expert in health care domain who is familiar with all those rules and regulations can define those role classes. The formal definition of role class in Object-Z can be found in (Xu & Zhang 2005).

In the health care simulation system, we have defined the following role classes:

1. Patient: A person who seeks for health care.

2. Physician: A person who determines whether diagnostics are to be undertaken, provides prescriptions, performs medical and surgical interventions, has the ability to direct patient care and advance a patient to the next step of care.

3. Medical Assistant: A health care professional who performs a variety of clinical, clerical and administrative duties within a health care setting. There are two roles defined as subclasses of this role class:
    a. Administrative Medical Assistant (MA Admin): Medical assistant who performs the administrative job.
    b. Clinical Medical Assistant (MA Clinical): Medical assistant who performs the clinical job.

4. Nurse: There are two roles defined as subclasses of this role class:
    a. Nurse Assistant is a nurse who assesses the patient's medical problem, provides care and helps to set up laboratory specimen and medical instruments.
    b. Nurse Practitioner: a registered nurse who has completed an advanced training program in primary health care delivery, and may provide primary care for non-emergency patients, usually in an outpatient setting.

Figure 2 shows the RADE interface for a user to create role classes and define the interrelationships among role classes. In this example, the interrelationships include inheritance, association and incompatibility. An inheritance relationship describes the generalization/specification relationship between two role classes. For example, both *MA Admin* and *MA Clinical* inherit the *Medical Assistant* role class since they are specified medical assistants. Association is a very common relationship between role classes; it indicates that an instance of one role class may perform an action on an instance of another role class. Association relationships exist between *Physician* and *Nurse*, *Physician* and *Patient*, etc. Incompatibility relationship describes the constraints that the role instances of two role classes cannot be taken by the same agent in the same interaction scenario. For example, an agent cannot take a Physician role instance for treating a Patient role instance if the agent is already taking this Patient role instance; however, the agent can take another Physician role instance for treating another Patient role instance that is not taken by this agent. The definition of such relationships depends on the domain knowledge, so we feel that the domain experts are the best candidate to use this interface to define the role classes and their interrelationships.

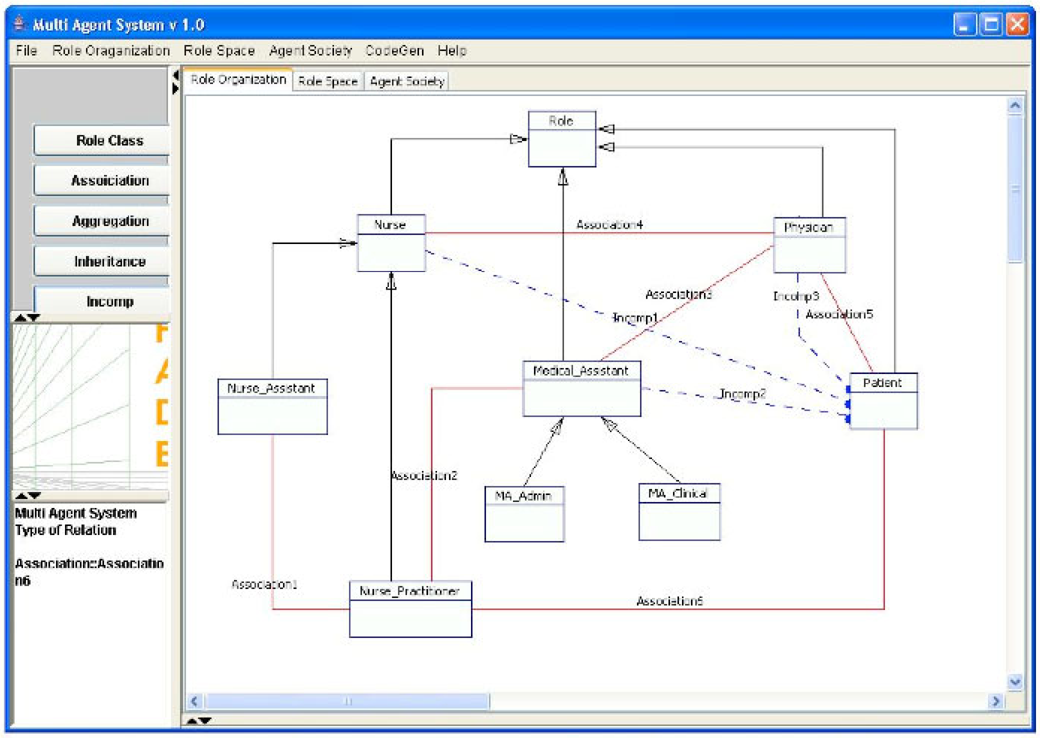


Figure 2 RADE Interface for creating roles (© [2007], [Journal of the Brazilian Computer Society]. Used with permission)

In this example, Physician role is defined with a goal to provide cure. The plan tree provides domain knowledge of how to accomplish this goal. To represent the domain knowledge, we introduce RTÆMS (Role-Based Task Analyzing, environment Modeling, and Simulation) language as an extension of the TÆMS language (Decker, 1996). TÆMS is a hierarchical task representation language, which supports representation of relationships among goals and sub-goals, the quantitative description of the atomic approaches and uncertainties, and resources. We extend the TÆMS language by introducing a role attribute for task nodes that represent goals and sub-goals. The attribute role specifies what roles are needed to carry out this goal or sub-goal. ***Figure 3*** shows the plan tree for the goal 'Provide Cure', which includes two sub-goals: 'Examine Patient' and 'Provide Treatment'. The goal 'Provide Cure' is associated with a **min** quality accumulative function (**qaf**), which specifies the following relationship:

Quality(ProvideCure) = min(Quality(ExaminePatient), Quality(ProvideTreatment))

Each role is defined with a goal, a plan tree, a motivational quantity production set (MQPS), a certificate and other attributes. A goal represents a task that the role needs to accomplish, and the plan tree specifies the domain knowledge of how to accomplish the goal in terms of decomposing it as sub-goals. Consider the following role class.

ROLE: Physician
GOAL: Provide Cure
MQPS: (MQ_professional, p1), (MQ_moral, p2), (MQ_experience, p3)
CERTIFICATE: MD (Doctor of Medicine)

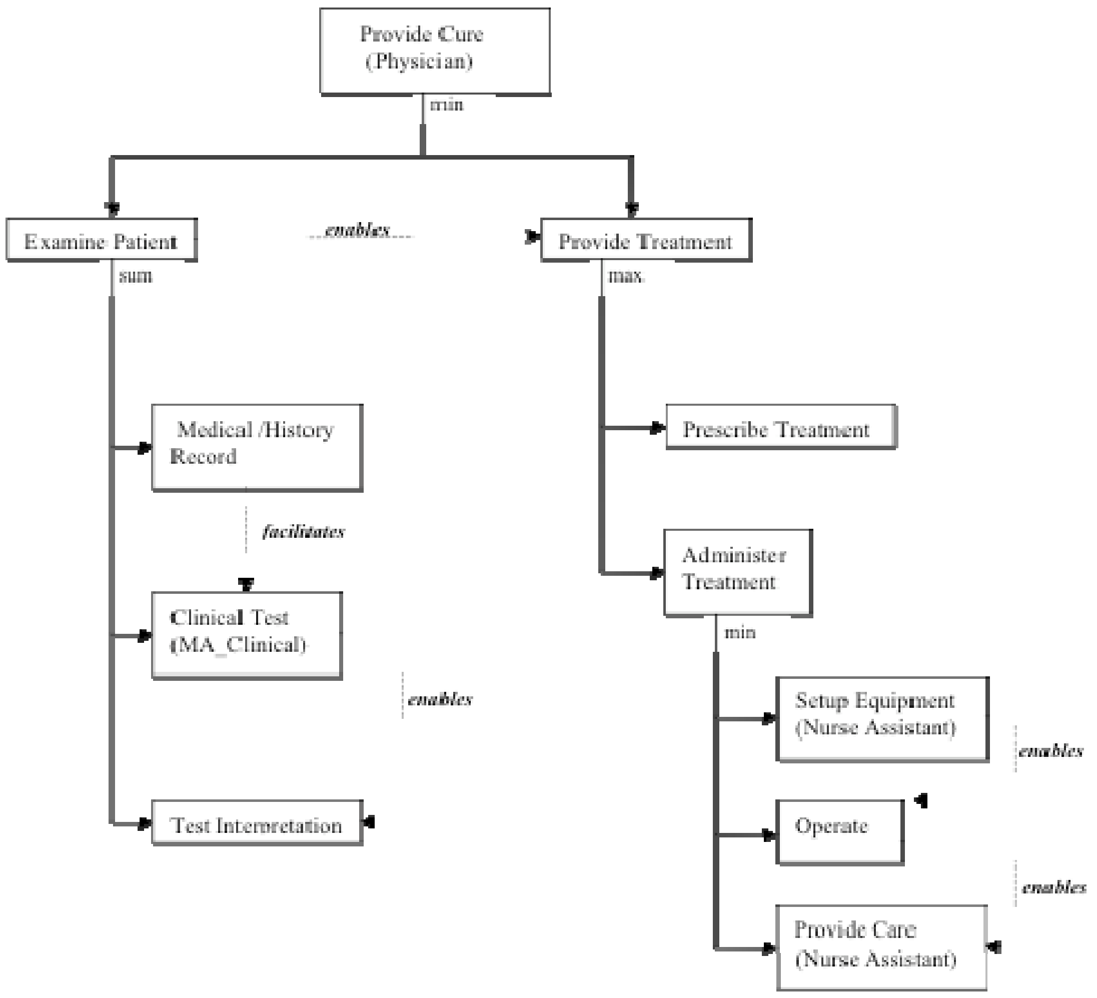


Figure 3 Plan tree for goal Provide Cure in RTÆMS representation (© [2007], [Journal of the Brazilian Computer Society]. Used with permission)

This min quality function associated with a goal means that the success of this goal depends on the success of all of its sub-goals. Meanwhile, the use of max quality function specifies that there are several alternatives to achieve the goal. For instance, to 'Provide Treatment' for the patient, the Physician can choose either 'Prescribe Treatment' or 'Administer Treatment'. Other available quality accumulation functions in RTÆMS language are: *sum* and *seq_sum*.

Each sub-goal can be decomposed into smaller goals, i.e. 'Examine Patient' consists of three sub-goals: '(Read) Medical History Record', 'Clinical Test 'and 'Test Interpretation'. For those non-local goals, where the tasks need to be performed by other roles, the specification of other roles is included in the plan tree description. For example, 'Clinical Test' should be performed by a Clinical Medical Assistant (MA Clinical), and task 'Setup Equipment' and 'Provide Care' belongs to the Nurse Assistant role. The dash lines represent the interrelationship between goals/sub-goals. For example, 'Clinical Test', which enables 'Test Interpretation,' means that the first goal 'Clinical Test' needs to be achieved successfully before it is possible to implement the second goal 'Test Interpretation'. In addition, '(Read) Medical History Record' facilitates the 'Clinical

Test' process because it can provide some useful information about the patient. Other types of interrelationships defined in TÆMS include *disables* and *hinders*. The primitive goal (lowest-level goal) in the TÆMS representation can be specified with more details in another plan tree that is associated with another role. For example, the plan tree for the sub-goal 'Provide Care ' is described in ***Figure 3***, this information belongs to the role Nurse Assistant. The plan tree represented in RTÆMS shows all possibilities to achieve a goal and the interrelationship among goals/sub-goals. It provides fundamental knowledge for agents to plan and schedule its local activities, and it also supports the collaboration and cooperation among agents. More details about the plan tree will be discussed later in the Section of Coordination.

Each goal is associated with a motivational quantity production set (***MQPS***): MQPS = {(MQi, qi), (MQj, qj), (MQk, qk)...}, which represents the success accomplishment of the goal that generates qi amount of MQi, qj amount of MQj, qk amount of MQk, etc. The MQPS describes how this goal contributes quantitatively to some higher-level goals (abstract goals), which are built in an agent's motivation. For instance, when an agent fulfills a goal 'Provide Cure', it collects p1 units of MQ_professional, p2 units of MQ_moral and p3 units of MQ_experience. The agent uses the MQPS specification in the goal definition and its motivation to determine whether it is interested in a role instance, and how interested it is.

The *Qualification* defined in a role class describes the requirements for a particular role. Only an agent who has the specified certificate can take a role instance of that role class. For example, Physician role is defined with a certificate of MD (Medical Doctor); only an agent with a MD certificate can take a Physician role instance.

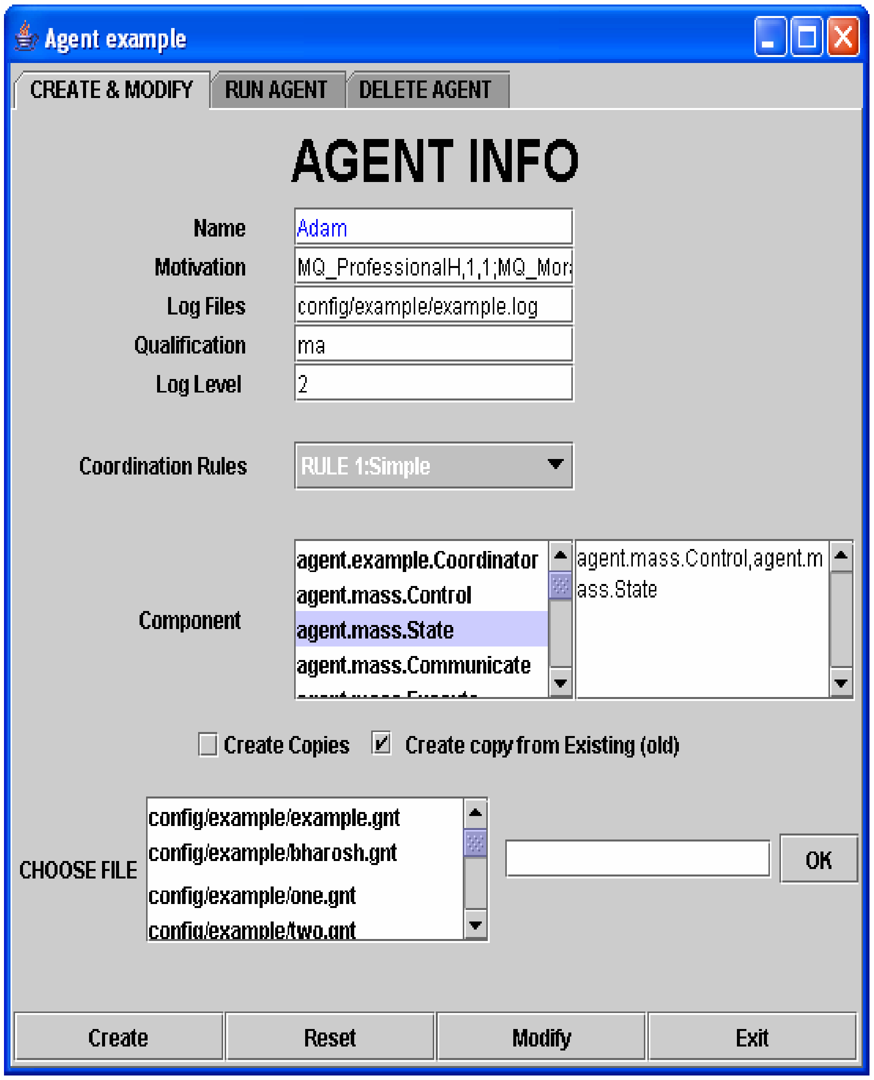
DEFINING AND DEVELOPING AGENT CLASSES

Agents are the real programmed entities running in the system. In the health care simulation system, each agent represents a personal assistant for a human user in the real world. The agent is responsible for scheduling a user's daily tasks according to the user's preference and constraints. The agent is also responsible for coordinating with other agents when coordination is needed between its own user and other users. A formal definition of agent class in Object-Z can be found in (Zhang, Xu & Shrestha 2007). An agent class definition includes: a set of attributes, motivations, utility function, sensor data, a set of reasoning mechanisms, and execution mechanisms.

Agent attributes include agent names, user, identification, and other descriptive characteristics. The values of these attributes are set when an agent instance is instantiated from the agent class. Different agent instances have different attribute values.

Motivation is defined as "any desire or preference that can lead to the generation and adoption of goals, and which affects the outcome of the reasoning or behavioral task intended to satisfy those goals" (Luck & d'Inverno, 1995). Motivation is the key for an agent to decide which goals it should pursue and how to pursue a goal. We adopt a quantitative view of motivation in our practice. Motivation is defined as a set of motivation quantities (MQs) (Wagner & Lesser 2002).) that the agent tracks and accumulates. Each MQ is associated with a preference function and represents progresses towards an abstract goal. An abstract goal is a long-term commitment to make progress toward certain direction but not a concrete task with a specified plan. For example, a user creates an assistant agent named Adam. The user specifies his preference on choosing tasks by defining the motivation of this agent as:

Motivation: {MQ_Professional, 0, 0;
MQ_Moral, 1, 1; MQ_Experience, 2, 2}



AGENT INFO

Name	Adam
Motivation	MQ_ProfessionalH,1,1;MQ_Mor
Log Files	config/example/example.log
Qualification	ma
Log Level	2
Coordination Rules	RULE 1:Simple
Component	agent.example.Coordinator agent.mass.Control agent.mass.State agent.mass.Communicate
	agent.mass.Control,agent.mass.State

☐ Create Copies ☑ Create copy from Existing (old)

CHOOSE FILE
config/example/example.gnt
config/example/bharosh.gnt
config/example/one.gnt
config/example/two.gnt

Figure 4 Automatic agent generation interface (© [2007], [Journal of the Brazilian Computer Society]. Used with permission)

The motivation specifies three long-term goals the user has: professional achievement, moral achievement and experience achievement, which are represented by three types of MQs due to the user's Physician role. The two numbers following the MQ name is the function index and the initial amount of this type of MQ. 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 could 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.

Each agent collects sensor data from the environment. For software agents built in this system, sensor data refers to the messages and information the agent receives from the environment including other agents. Based on the sensor data it collects and its motivation, the agent uses its reasoning mechanisms to make decisions. The decisions are made at different levels: selection of roles, selection of goals, and selection of the approach to fulfilling the goals. The first issue is resolved by A-R mapping mechanisms, and the later two issues are inter-related, which are solved by planning /scheduling mechanisms. Given the formal definition of motivations, goals and the detailed description of alternatives to achieve a goal, it is possible to build some general, domain-independent reasoning mechanisms/toolkits. The user can select appropriate components from such toolkits and add them to the agent; the user can also customize these general mechanisms and toolkits by setting up certain parameters. These general mechanisms and toolkits are reusable for agents in different application domains.

Each agent is equipped with some execution mechanisms that can be used to generate the output, which changes the environment. For software agents, the execution mechanisms are the primitive actions to change the environment state. Some of these execution mechanisms are domain-dependent. For example, in our health care simulation system, an agent representing a hospital worker is built with an execution mechanism to set up medical equipment, which is an action the person can perform in real world. Other execution mechanisms could be application-independent, such as sending a message to another agent.

AUTOMATIC AGENT GENERATION PROCESS

After the user has defined role classes and agent classes, agent can be automatically created using a tool we developed. The basic idea of automatic generation of agents is to use component-based agent architecture, where the user can select the components to be included in this agent, and specify a set of attributes of the agent.

The designer or the user of the agent needs to decide what reasoning tool should be built in and select the appropriate execution tools for the agent according to the design purpose of the agent. It is assumed that there are a set of reasoning and execution mechanisms available in the toolkit, which can be selected and plugged into the agent seamlessly.

Based on the general agent architecture, we developed a tool to support the automatic agent generation process. This tool is created by extending the JAF framework (Vincent, Horling & Lesser, 2001) developed by MAS lab at University of Massachusetts, Amherst. This tool includes a graphic user interface (GUI), 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 4***.

The user also defines the agent's reasoning and execution mechanisms by selecting a number of ready-to-plug-in components such as: planning, scheduling, communication, etc. The user can select what coordination rule should be used by this agent. We will discuss more about the coordination rule in the next section. After an agent class is created, one or multiple agent instances (the executable programs) can be created from this class definition. Each agent instance is an independent program, and the agent is named after its class with a unique number ID. For example, when a user creates an agent class "X" and three agent instances of this class, the three agents are named as "X_1", "X_2" and "X_3," respectively. The user can run agents from this interface by clicking on the "RUN AGENT" menu box on the top, and selecting a number of agents to run from a list of agents that have already been created. Multiple agents can be created and run on difference machines. The user can choose to delete existing agents by clicking on the "DELETE AGENT" menu box. Finally, the user has an option to choose the coordination rules from three types of rules, namely simple rules, hard and soft relationships based rules, and priority based rules.

AGENT COORDINATION AND COOPERATION

In a health care simulation system with complex activities, distributed information and resources, agents need to coordinate and cooperate on their actions. Efficient coordination and cooperation mechanisms are important for the performance of the system. An agent should coordinate its own actions with those of other agents when there are constraints and interdependencies among their actions.

The RTÆMS language supports collaborations and cooperation by specifying interrelationship among goals and sub-goals, so agents know when and with whom they need to collaborate and cooperate. A set of domain-independent general collaboration mechanisms (GPGP) based on TÆMS language (Lesser et. al., 2004), has been developed, where some of GPGP similar mechanisms are reused in RADE framework based on RTÆMS language. Agents can coordinate and cooperate with each other using the set of mechanisms according to the protocols defined in the role, which specify how the interaction between roles should proceed.

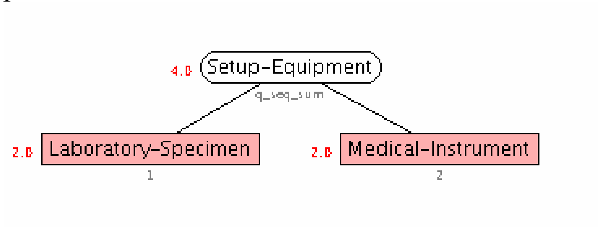


Figure 5 Plan tree for 'Setup-Equipment' in RTÆMS representation

Figure 3 and ***Figure 5*** illustrate pictorially the information that are captured in a RTEAMS representation, which include:

1. Top-level goals that an agent intends to achieve including the deadline for their completion. In ***Figure 3***, 'Provide Cure' is the top-level goal that needs to be completed and in ***Figure 5***, 'Setup Equipment' is the top-level goal that needs to be completed.

2. One or more of the possible ways of achieving goals is expressed as an abstraction hierarchy whose leaves are basic action instantiations, called methods. In ***Figure 5***, the top-level goal 'Setup Equipment' has sub-goals "Laboratory Specimen' and 'Medical Instrument', which are the methods. These sub-goals need to be completed before the top-level goal can be achieved.

3. Quantitative definition of the degree of achievement in terms of measurable characteristics, such as solution quality and time, is called the quality accumulation function (qaf). In ***Figure 5***, there exits a quality accumulation function *seq_sum* between the sub-goals "Laboratory Specimen' and 'Medical-Instrument'. The total quality of the goal "Setup Equipment" is the sum of the quality of its sub-goals "Laboratory Specimen' and 'Medical Instrument', and these two sub-goals need to be accomplished in a sequence order.

4. Task relationships indicate how basic actions or abstract task achievement affect task characteristics such as its quality and time, elsewhere in the task structure. In ***Figure 3***, there exits a "facilitates" relationship between the task 'Medical History Record' and 'Clinical Test'. A *facilitates* relationship indicates that if the task 'Medical History Record' is completed before the start of task 'Clinical Test', it will increase the quality, and reduce the cost and duration of task 'Clinical Test' by some value.

Task relationships represent a measure view of temporal constraints among activities as a result of information sharing relationships. An *enables* relationship is a hard relationship that essentially acts as a binary switch. In this case, the target method or task cannot accrue quality until the enabling interrelationship is active. A *disables* relationship indicates the exact converse of an *enables* relationship, which precludes the possibility of performing an activity when another activity is performed,. Both a *facilitates* and *hinders* relationship are soft relationships. When a 'facilitates' relationship is active, the targets' quality is increased by some quality power, and the duration and cost are reduced by the duration power and cost power, respectively. Similarly, when a 'hinders' relationship is active, the target's quality is reduced, while the duration and cost are increased. These relationships are called non-local effects if they are relationships between tasks situated in different agents for coordination. Relationships among tasks in the same agent are not of direct concern of the coordination component. The measured view of these relationships indicates how the quality of the information generated by an activity will affect the performance characteristics of the activity using this information, such as the length of its execution and the quality of its resulting solution.

There is a strong connection between the coordination module and a local scheduler module that is part of each agent's architecture. In our work, the agent's local optimization expert is the Design-to-Criteria Scheduler (DTC) (Wagner, Garvey & Lesser, 1998). During the coordination process, the coordination module queries the DTC scheduler repeatedly to explore the implications of constraints. The coordination and DTC module present in each agent can guide the agent's activities using knowledge of its own local situation and partial knowledge of the activities being carried by other agents. The coordination component in each agent also coordinates with that of other agents to generate constraints on local control that leads to more coherent agent activities.

Each agent starts its coordination component by constructing its own local view of the activities that the agent intends to pursue, as well as the relationships among these activities (Lesser et. al., 2004). The RTÆMS representation is used by the problem solving, coordination and scheduling components as a common communication language. The coordination component helps to construct a global view for an agent, and to recognize and respond to particular inter-agent task structure relationships by making commitments to other agents. The commitments result in coordinated behavior by affecting the tasks an agent executes and the results transmitted. The DTC scheduler, based on commitments, agent's goal, the local and non-local values of tasks, and other agent activity constraints, creates a schedule of activities for the agent, which must meet the real-time deadlines. The coordination component coordinates the activities of an agent through modulating its local control as a result of placing commitments and constraints on the local scheduler.

The coordination component uses the RTÆMS task structure representation to add an extension of local and non-local commitments to task achievement. The coordination includes the goals that the agent is currently pursuing, the goals it will likely pursue in the near future, the characteristics of the abstract tasks and basic actions available to achieve these goals, their relationships to other tasks, and the degree of achievement necessary for each goal.

A user can choose a coordination rule from three types of coordination rules, namely Rule1 (simple), Rule 2 (hard and soft relation), and Rule 3 (priority based). The coordination mechanism between agents depends on selection of a specific rule.

Suppose we have two agents A and B shown in ***Figure 6***. Agent B is performing task B1. Task B1 has subtask A1 and B2. Subtask A1 is performed by agent A and subtask B2 is performed by agent B itself. There is an 'enables' relationship from A1 to B2.

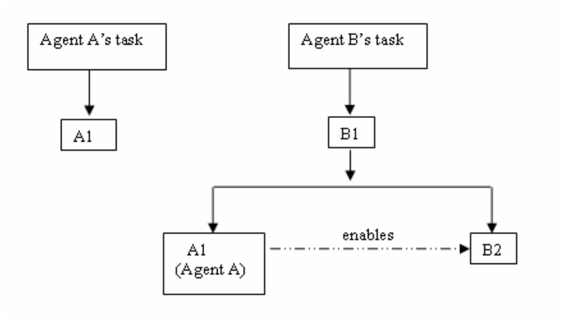


Figure 6 Agent A and Agent B's initial task view with enables relationship

When a user selects Rule 1 (simple), the agents use a very simple coordination mechanism - they only consider the quality accumulation function but not the hard and soft relationships between the tasks. As shown in *Figure 6*, suppose there is a *seq_sum* quality accumulation function associated with task B1, agent B recognizes that the quality achievement of B1 depends on the accomplish of task A1 and it has to be performed before task B1, it then sends a message to agent A asking it to perform task A1 by a given deadline. Agent A replies with the start time and finish time for task A1 according to its local schedule. Upon receiving this message agent B reschedules the start time of its task B1 to the finish time of task A1. This is the Scenario 1 described in *Table 1*.

Scenario	Using Rule		Agent A - task A1				Agent B - task B2			
			Start time	Duration	Deadline	Priorty	Start time	Duration	Deadline	Priorty
1	Rule 1	Intial schedule	5	5	15	5	15	5	25	5
		After Coordination	5	5	15	5	10	5	25	5
2	Rule 2	Intial schedule	5	5	15	5	5	5	25	5
		After Coordination	0	5	15	5	5	5	25	5
3	Rule 2	Intial schedule	5	6	20	5	5	5	25	5
		After Coordination	0	6	20	5	6	5	25	5
			Agent A - task A1				Agent B - task B2			
			Start time	Duration	Deadline	Priorty	Start time	Duration	Deadline	Priorty
4	Rule 3	Intial schedule	5	5	25	3	5	6	15	5
		After Coordination	0	5	25	3	5	6	15	5
			Agent A - task A2							
		Intial schedule	0	5	25	7				
		After Coordination	5	5	25	7				

Table 1 Coordination scenarios using different rules

Rule 2 deals with both hard and soft relationships together with the quality accumulation functions and non-local tasks. Hard relationships include the *enables* and *disables* relationship, and soft relationships include *facilitates* and *hinders* relationships.

As shown in ***Figure 6***, task A1 has *enables* relationship with task B2. Agent B sends a message to agent A saying that task B2 has an *enables* relationship with task A1 and should complete task A1 by a given deadline. Agent A checks the start time of task A1. If the start time of task A1 is less than or equal to the start time of task B2, then agent A makes a commitment to agent B that it can finish the task B2 by the given deadline. If the finish time of task A1 is greater than the start time of task B2, agent A then moves task A1's start time to task A1's earliest start time. Agent A sends the new start time and finish time of task A1 to agent B. If the finish time of task A1 proposed by agent A is less than or equal to the start time of task B2, agent B follows its normal schedule. Otherwise, agent B temporarily sets task B2's start time to the finish time of task A1 as proposed by agent A and calculates its new finish time for task B2. If the new finish time falls within the deadline of task B2, then agent B reschedules its task B2 with new values. Otherwise, task B2 is not performed.

Scenario 2 in *Table 2* explains how agents coordinate with each other using Rule 2. In the initial schedule for task B2, the start time is 5. Agent A sets its start time to its earliest start time (0). Now the new finish time for task A1 is 5. Since the new finish time for task A1 is equal to the start time of task B2, the schedule for task B2 remains unchanged.

Scenario 3 shows a different case. In the initial schedule for task B2, the start time is 5. Agent A sets its start time to its earliest start time (0). Now the new finish time for task A1 is 6. Since the new finish time for task A1 is greater than the start time of task B2, rescheduling of task B2 is needed. Task B2 has a new start time as 6 after rescheduling.

Rule 3 is based upon priority of a task, which takes into consideration the hard and the soft relationships. Rule 3 is useful when an agent is performing more than one task. In the RTÆMS representation, each task has a new attribute called "priority", with its value ranging from 1 (i.e., the highest priority) to 10 (i.e., the lowest priority).

Let us assume that agent A has two tasks A2 and A1. Task A1 has *enables* relationship with task B2. Agent B sends a message to agent A saying that task A1 has an *enables* relationship with task B2 and requires task A1 to be completed by a given deadline. Agent A checks the start time of task A1. If the start time of task A1 is less than or equal to the start time of task B2, agent A makes a commitment to agent B that it can finish the task B2 by the given deadline. If the finish time of task A1 is greater than the start time of task B2, agent A then checks the start time and finish time of task A2. If task A2 is performed before task A1, agent A compares the priority of task A1 and A2. If the priority of task A1 is higher than that of task A2, agent A reschedules task A1 to be performed before task A2, and the new start time and finish time of task A1 is sent to agent B. Otherwise, agent A sets the start time to task A1 to its earliest start time. Agent A sends its new start time and finish time to agent B. If the finish time of task A1 proposed by Agent A is less than the start time of task B2, agent B follows its normal schedule. If the proposed finish time of task A1 is greater, agent B temporarily sets B2's start time to A1's finish time and calculates the new finish time for task B2. If the new finish time is no later than the deadline of task B2, agent B reschedules its task B2 with new values; otherwise, task B2 is not performed.

Table 1 shows how Rule 3 works. The initial start time of task B2 is 6 and agent A cannot complete the task A1 before 6. Hence, agent A compares the priority of task A2 and A1. Since priority of task A1 is higher, A1 is performed before A2, and the new schedule is sent to agent B.

Similarly, these coordination rules can be used to support other non-local relationships, including disables, facilitates and hinders.

RUNTIME SCENARIO

Now we present a runtime scenario for a hospital organization to describe how the health care simulation system works. The scenario demonstrates how the dynamic task allocation is accomplished through the A-R mapping mechanism, and how agents coordinate with each other in their activities. In this scenario, a special agent role space is first created. Role space agent is initially not taking any active role in the system; rather, it is mainly responsible for maintaining and managing the role instances in the system. The role space checks the plan tree of a role instance, when this role instance is taken by an agent, which recognizes the needs to create new role instances. The role space selects the appropriate agent for the role instance after verifying the qualification and consistency of the candidates.

When the system is initialized, the system administer creates several Patient role instances to express the expected service requirements from patients. The number of Patient role instances depends on the capability of the hospital. These patient role instances are posted in the role space and are not active until they are taken by some agents. When a (real) patient Bryan enters the hospital for services, a personal assistant agent named Bryan is created for this patient, and the agent takes one Patient role instance. In this case, Bryan uses the coordination Rule 3, which is specified when the user defines the Patient agent class.

When agent Bryan takes the Patient role instance, it has one goal to achieve: 'Get Cure'. The plan tree of this goal describes that two sub goals 'Assist Patient' and 'Provide Cure' must be achieved so that the goal 'Get Cure' can succeed. The goal 'Assist Patient' belongs to a MA Admin (Administrative Medical Assistant) role and the goal 'Provide Cure' belongs to a Physician role. Based on this information, a Physician role instance and an MA Admin role instance are created by the role space.

Four other agents, Adam, Cathy, Kevin and David that represent four medical professionals are also created and active in the system. Both agent Adam and the remaining agents are initialized with coordination Rule 3. They have been idle and sent requests to the role space for available role instances. When the MA Admin and Physician role instances are created in the role space, all three agents who are interested in taking any additional role instances receive a message for this update. After receiving the message, the agent checks the goal associated with the role instance, especially the MQPS, to see if it matches its own motivation. If the MQPS contains the same type of the agent's MQ in its motivation, the agent is said to be interested in taking that role instance.

For example, the Physician role instance has MQPS as: (MQ_professional, p1), (MQ_moral, p2), (MQ_experience, p3), all these three types MQ's belong to agent Adam's motivation. So Adam is interested in this role instance. How interested Adam is for this role instances depends on the actual values of p1, p2 and p3, the exact structures of the mapping functions with index 0, 1, and 2, and the current accumulation of these MQ's for agent Adam.

If agent Adam is interested in multiple role instance openings, it will compare the degree of interests in these role instances and select the most interested ones, and send requests to the role space. It is also possible that the role space receives requests from multiple agents for the same role instance. In this case, the role space verifies the qualification of each agent by matching the agent's qualification with the certificate requirement defined in the corresponding role class. For example, agent Adam is qualified for this role instance because it has a MD qualification that matches the certificate requirement of the Physician role class. The role space also checks if this role instance is compatible with other role instances the agent is taking right now. For instance, suppose agent Bryan has a MD qualification and it is also interested in this Physician role instance; however, according to the incompatibility relationship between the Physician role and the Patient role, agent Bryan cannot take this role instance because it takes the Patient role instance related to this Physician role instance.

After verifying the qualification and checking the consistency, the role space selects an appropriate agent (agent Cathy) for the MA Admin role instance, whose goal is to 'Assist Patient'. The role space then tells agent Cathy that the task 'Assist Patient' has an *enables* relationship with the task "Provide Cure'. The plan tree for the goal 'Assist Patient' consists of four sub goals: 'Greet Patient', 'Schedule Appointment', 'Admit Patient', and 'Answer Telephone'. All of these sub-goals can be performed by the same agent who takes the MA Admin role instance, so no new role instance has to be created.

After assigning the MA Admin role instance to agent Cathy, the role space assigns the Physician role instance to another appropriate agent - Adam, based on its qualification. The role space then tells agent Adam that task 'Assist Patient' enables its task 'Provide Cure'. The goal of taking the Physician role by agent Adam is to 'Provide Cure'. The role space reads the plan tree associated with the goal, and finds that in order to accomplish this goal, sub-goals 'Setup Equipment' and 'Provide Care' must be accomplished by other roles. In response to this need, new role instances Nurse Assistant and MA Clinical (Clinical Medical Assistant) are created. The role space then selects appropriate agents Kevin and David to take these role instances respectively. 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 utility as defined in the MQPS of this role instance, and release the role instance, which will be further deleted by the role space.

After all role instances have been assigned to appropriate agents, the role space sends a table of roles to the agent who is performing that role, followed by a message to start the coordination. The agents can now begin the coordination process. For example, as shown in ***Figure*** *7*, Patient Bryan a goal to 'Get Cure", which has two non-local subtasks 'Assist Patient' and 'Provide Cure' performed by MA Admin Cathy and Physician Adam, respectively. Patient Bryan sends a message to both agents to ask them to complete the task within the deadline. Agents Cathy and Adam reply to Patient Bryan with their scheduled execution time. The Physician Adam coordinates with MA Admin Cathy using coordination Rule 3 to schedule the task 'Assist Patient' before 'Provide Cure'. There is a *facilitate*s relationship between task '(Read) Medical History Record' and task 'Clinical Test'. Since both tasks belong to the same agent, so the 'facilitates' relationship is taken care of by agent Cathy's local scheduler.

Since task 'Assist Patient' has an *enables* relationship with task 'Provide Cure', Physician Adam requests MA Admin Cathy to complete the task by 12. However, Cathy has another task 'Clean' that is scheduled for time 0 to 12, and the task 'Assist Patient' is scheduled for time 12 to 24. Cathy compares the priority of task 'Assist Patient' and task 'Clean': priority of task 'Assist Patient' is higher so this task is rescheduled before the task 'Clean'. Nurse Assistant Kevin can perform the task 'Provide Care' after task 'Operate' performed by Physician Adam. Similarly, Clinical MA David can perform the task 'Setup Equipment' before task 'Operate' and meet the deadline requested by Physician Adam. So no more rescheduling is necessary. The initial schedule for all tasks and new schedule for task 'Clean' are shown in *Table 2*.

After the coordination is complete the agents can now begin execution, Patient Bryan can now begin executing its task 'Get Cure', which has subtasks 'Assist Patient' and 'Provide Cure'. The task 'Assist Patient' should be performed by MA Admin Cathy. Patient Bryan agent sends a message to MA Admin Cathy to begin the task 'Assist Patient'. MA Admin Cathy then begins executing the task 'Assist Patient', which has the subtasks 'Answer Telephone', 'Schedule Appointment', 'Greet Patient' and 'Admit Patient'. The quality of the task 'Assist Patient' is defined by the quality accumulative function "seq_sum", which is the total quality of all of its sub-tasks performed in sequence. Since MA Admin Cathy itself can perform all of the subtasks, it starts the execution immediately. After Cathy completes the task 'Assist Patient', it collects the motivation quantities as defined in the MQPS of this role instance.

Agent		Intial schedule				New schedule			
Admin-MA	Task	Start time	Duration	Finish time	Priorty	Start time	Duration	Finish time	Priorty
	Clean	0	12	12	7	12	12	24	7
	Assist-Patient	12	12	24	3	0	12	12	3
	Answer-Telephone	12		15		0	3		
	Schedule-Appointment	15		18		3	6		
	Greet-Patient	18		21		6	9		
	Admit-Patient	21		24		9	12		
Physician	Task	Start time	Duration	Finish time	Priorty				
	Provide-Cure	12	86	79	3				
	Medical-History-Record	12		15					
	Clinical-Test	15		17					
	Test-Interpretation	17		20					
	Setup-Equipment	20		40					
	Operate	40		43					
	Provide-Care	43		79					
Nurse-Assistar	Task	Start time	Duration	Finish time	Priorty				
	Provide care	46	33	79	3				
	Walk-Patient	46		49					
	Clean-Room	49		52					
	Server-Meal	52		55					
	Dress-Patient	55		58					
	Check-Pulse	58		61					
	Check-Blood-Pressure	61		64					
	Check-Temperature	64		67					
	Check-Respiratory-Rate	67		70					
	Physical-Condition	70		73					
	Mental-Condition	73		76					
	Emotional-Condition	76		79					
Clinic MA	Task	Start time	Duration	Finish time	Priorty				
	setup-equipment	26	6	32	5				
	Laboratory-Specimen	26		29					
	Medical-Instrument	29		32					

Table 2 Task schedules

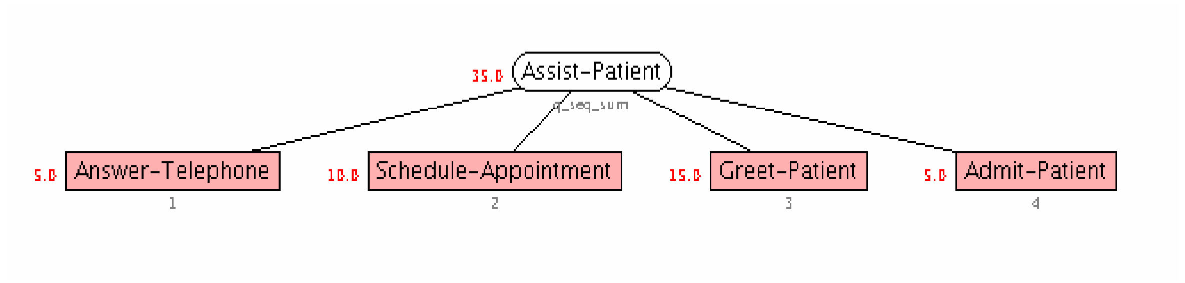


Figure 7 Plan tree for 'Assist-Patient' in RTÆMS representation

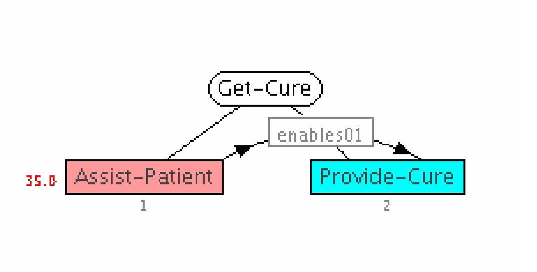


Figure 8 Plan tree for updated 'Get-Cure' in RTÆMS representation

Upon receiving this message, patient Bryan updates its own task structure. MA Admin Cathy has rescheduled this task after the task 'Assist Patient'. So when Cathy completes the task 'Assist Patient', it begins executing the task 'Clean'. Now Patient Bryan can start executing the task 'Provide Cure'. Since the task 'Provide Cure' is performed by Physician Adam, so Patient Bryan sends a message to Physician Adam saying that it can start the execution. Physician Adam begins the execution of the task 'Provide Cure'. The task 'Provide Cure' has subtasks 'Examine Patient' and 'Provide Treatment' as shown in ***Figure 3***.

Physician Adam begins executing 'Examine Patient', which has subtasks '(Read) Medical History Record', 'Clinical Test' and 'Test Interpretation', which can all be performed by Physician Adam. After completion of these subtasks, it then begins executing task 'Provide Treatment', which has subtasks 'Prescribe Treatment' and 'Administer Treatment' with the quality accumulative function "max", which means only one of these two subtask needs to be accomplished.

If Physician Adam decides to perform the task 'Administer Treatment', then the three subtasks 'Setup Equipment', 'Operate' and 'Provide Care' need to be accomplished. The task 'Setup Equipment' is performed by MA Clinical agent David. So Physician Adam sends a request to Clinical MA David to perform the task 'Setup Equipment'. David starts executing the task 'Setup Equipment', which has subtasks 'Laboratory Specimen' and 'Medical Instrument'. After the completion, David sends a message to Physician Adam, saying that the task has been completed, together with the quality accumulated, cost accrued and the time taken. Upon receiving this message, physician Adam updates its task structure and begins executing 'Operation', which is performed by itself.

Similarly, the task 'Provide Care' is performed by Nurse Assistant Kevin. Physician Adam sends a request to Kevin to execute the task. Kevin begins executing the task 'Provide Care', which has the subtasks 'Serve Patient', 'Provide Skin Care' and 'Observe Patient'. Nurse Assistant Kevin itself can perform all of these subtasks.

.

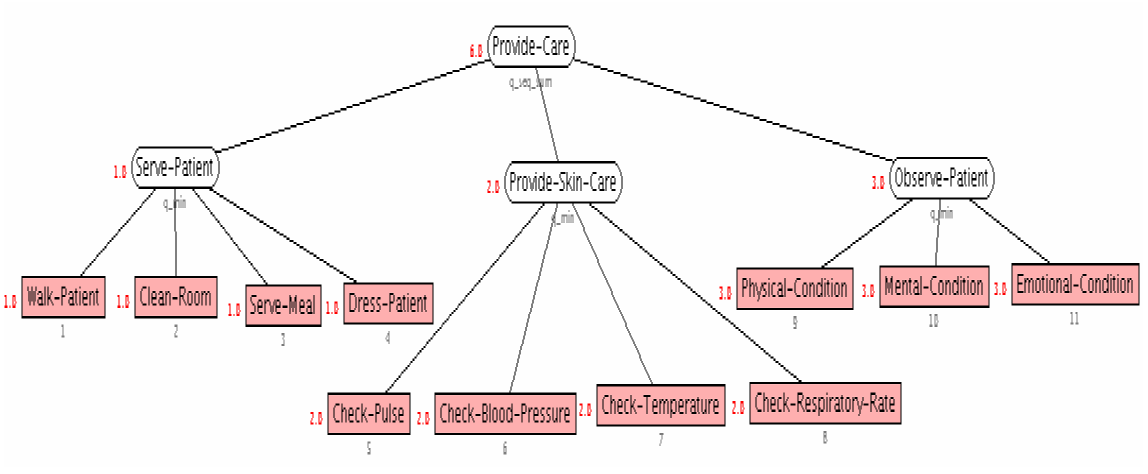


Figure 9 Plan tree for updated 'Provide Care' in RTÆMS representation

After the completion of the task, Kevin sends a message to Physician Adam, saying that the task has been completed, together with the quality accumulated, cost accrued and time taken. Upon receiving this message, physician Adam updates its task structure. Since task 'Provide Cure' has now been completed, Adam sends a message to Patient Bryan that the task has been completed, together with the quality accumulated, cost accrued and time taken. Upon receiving this message, Patient Bryan updates its task structure.

FUTURE TRENDS

The future work includes further development of the system based on the current prototype. Especially, we are interested providing support for users to define interaction protocols in role classes, and integrating those domain-dependent protocols with domain-independent communication mechanisms in agents. We are also interested in experimenting with large systems, more complex scenario and analyzing the system performance.

CONCLUSIONS

In this chapter, we describe a multi-agent health care simulation system built using RADE framework. The integrated framework supports role-based design of multi-agent systems as well as implementation of utility-driven agents that can use a variety of existing agent reasoning and coordination mechanisms. We describe how the roles are defined, how agents are created, and how the role instances are mapped to agents. We also describe the rule-based coordination mechanisms and present a runtime scenario that shows how the simulation system works and how agents coordinate with each other to schedule their local activities. This work verifies the feasibility of modeling health care system with multi-agent approach and demonstrates the strength of automatic coordination, planning and scheduling.

REFERENCES

[1] Becht, M., Gurzki, T., Klarmann, J. & Muscholl, M. (1999). ROPE: Role-Oriented Programming Environment for Multi-Agent Systems. *Conference on Cooperative Information Systems*. pp. 325–333.

[2] Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. (2003). JADE A White Paper. *EXP in search of innovation - Special Issue on JADE, TILAB Journal*, 3, 6-19.

[3] Decker, K. (1996). TAEMS: A Framework for Environment Centered Analysis and Design of Coordination Mechanisms. In G. O'Hare and N. Jennings (Eds.), *Foundations of Distributed Artificial Intelligence*, (pp. 429-448). Wiley Inter-Science.

[4] DeLoach, S., Wood, M., & Sparkman, C.H. (2001). Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11, 231–258.

[5] Graham, J. R., Decker, K. S. & Mersic, M. (2003). DECAF – A Flexible Multi Agent System Architecture. *Autonomous Agents and Multi-Agent Systems*, 7(1-2), 7-27.

[6] Gracanin, D., Bohner, S. A., Hinchey, M. (2004). Towards a Model-Driven Architecture for Autonomic Systems. *Proceeding of 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04).* pp. 500-505.

[7] Kuwabara, K. , Ishida, T., & Osato, N. (1995). AgenTalk: Describing Multiagent Coordination Protocols with Inheritance. *Proc. 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '95).* pp. 460-465.

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

[9] Luck M. & d'Inverno, M. (1995). A Formal Framework for Agency and Autonomy. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95).* pp. 254-260. AAAI Press/ MIT Press.

[10] Maria, D. B.A., Silva, V.T. & Lucena, C.J.P. (2005). Developing Multi-Agent Systems Based on MDA. *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05).* Porto, Portugal.

[11] Vincent, R., Horling, B., & Lesser, V. (2001). 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*.

[12] Wagner, T., Garvey, A., & Lesser, V. R. (1997). Complex Goal Criteria and its Application in Design-to-Criteria Scheduling. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

[13] Wagner, T. & Lesser, V. (2002). Evolving Real-Time Local Agent Control for Large-Scale MAS. In J.J. Meyer and M. Tambe (Eds.), *Intelligent Agents VIII (Proceedings of ATAL-01)*, *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin.

[14] Williamson, M., Decker, K.S., & Sycara, K. (1996). Unified information and control flow in hierarchical task networks. *Proceeding of the AAAI-96 workshop on Theories of Planning, Action and Control*.

[15] Wooldridge, M., Jennings, N., & Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems,* 3, 285–312.

[16] Xu, H. & Zhang, X. (2005). A Methodology for Role-Based Modeling of Open Multi-Agent Software Systems. *ICEIS (3).* pp. 246–253.

[17] Xu, H., Zhang, X. & Patel, R. J. (2007). Developing Role-Based Open Multi-Agent Software Systems. *International Journal of Computational Intelligence Theory and Practice (IJCITP),* 2(1): 39-56.

[18] Zambonelli, F., Jennings, N. & Wooldridge, M. (2003). Developing Multiagent Systems: The Gaia Methodology. *ACM Trans. on Software Engineering and Methodology*, 12, 317–370.

[19] Zhang, X. & Xu, H. (2006) Towards Automated Development of Multi-Agent Systems Using RADE. *Proceedings of the 2006 International Conference on Artificial Intelligence (ICAI'06)* (pp. 44-50). Las Vegas, Nevada.

[20] Zhang, X., Xu, H. & Shrestha, B. (2007). An Integrated Role-Based Approach for Modeling, Designing and Implementing Multi-Agent Systems. *Journal of the Brazilian Computer Society (JCBS): Special Issue on Software Engineering for Multi-Agent Systems.* 13(2): 45-60.