Visualizing a Multiagent-Based Medical Diagnosis System Using a Methodology Based on Use Case Maps

Tawfig Abdelaziz[1], Mohamed Elammari[2], and Rainer Unland[1]

[1] University of Duisburg-Essen
Institute for Computer Science and Business Information Systems (ICB)
Data Management Systems and Knowledge Representation
Schützenbahn 70, D-45117 Essen
{Tawfig,unlandr}@informatik.uni-essen.de
[2] Garyounis University
Faculty of Science, Department of Computer Science
Benghazi, Libya
elammari@ccs.carleton.ca

Abstract. Multi-agent systems are realized as a practical solution to the problem of constructing flexible and dynamic environments. Understanding such systems needs a high-level visual view of how the system operates as a whole to achieve some application related purpose. In this paper, we demonstrate how a visual high-level view called Use Case Maps (UCMs) may help in visualizing, understanding, and defining the behaviour of a Multi-agent medical diagnostic system.

1 Introduction

Nowadays, the concept of agent represents an important revolution and new approach to the development of complex software systems. There are many definitions of agents and multi-agent systems. However, we have chosen what we found most suitable to our problem domain.

An agent is an encapsulated software system that is intelligent and capable of actively performing autonomous actions in order to meet its objectives [7]. Multi-agent systems are collections of autonomous agents that interact or work together to perform tasks that satisfy their goals [6]. A methodology is a software engineering method that spans many disciplines, including project management, analysis, specification, design, coding, testing, and quality assurance. All of the methods guiding this field are usually mixtures of all of these disciplines. To understand and model agent systems some methodologies were proposed like *Agent-oriented methodology HIM* [1]*, Gaia methodology* [12], *The Styx Agent methodology* [13], *PASSI methodology* [14], *Tropos methodology* [15], *AgentUML* [16], … *etc*. Developing agent systems requires suitable methodologies and good software development techniques.

A lot of different Agent Oriented Software Engineering (AOSE) methodologies have been compared in [17, 18], including the AAII methodology, AgentUML, Gaia, MaSE, Tropos, Prometheus and ROADMAP. Each methodology has it is strong and weak points, and each included features which are tailored for a specific application domain. It is clear that there is no methodology, which can do it all. We suggest the use of the HIM methodology to develop a powerful multi-agent medical diag-

G. Lindemann et al. (Eds.): MATES 2004, LNAI 3187, pp. 198–212, 2004.
© Springer-Verlag Berlin Heidelberg 2004

nostic system, because it captures the most shared elements of the existing methodologies, such as cooperation and interaction, organizational design, communication, collaboration, and coordination.

The HIM methodology is especially tailored for agent systems; it provides a systematic approach for generating implementable system definitions from high-level designs. The methodology captures effectively the complexity of agent systems through depicting the internal structure of agents, relationships, conversations, and commitments. Also HIM contributes to the development of agent-oriented programming as a discipline. One of the system views suggested by the methodology is the *Agents System view.* This view is represented by a high level visual view, which is realized by use case maps (UCMs) [1,2,3,4]. The view enables the understanding of agent systems and shows how all system components work together to achieve some application related purpose. HIM confirms that any design process that is tailored to agents should provide support for the following:

System View: Understanding agent systems requires a high-level visual view of how the system works as a whole to accomplish some application related purpose. It is difficult to express this understanding if the only models available are low-level design diagrams, such as object interaction diagrams, and class inheritance hierarchies [1]. We need a macroscopic, system-oriented model to provide a means of both visualizing the behaviour of systems of agents and defining how the behaviour will be achieved, at a level above such details.

Structure: The internal structure of agents consists of some aspects such as goals, plans, and beliefs. This structure should help to discover other agents in the system and their internal structure

Relationships: An agent has dependencies and jurisdictional relationships with other agents. An agent might be dependent on another agent to achieve a goal, perform a task or supply a resource. A process should capture the different inter-agent dependencies and jurisdictional relationships.

Conversations: Agents must cooperate and negotiate with each other. When agents communicate, they engage in conversations. A process should capture the conversational messages exchanged and facilitate the identification of conversational protocols used in communication.

Commitments: Agents have obligations and authorizations about services they provide to each other. A process should capture the commitments between agents and any conditions or terms associated with them.

Systematic Transitions: A good design process should provide guidelines for model derivations and define traceability between the models.

In this paper, we describe a process for designing agent systems in which a visual technique is used to provide a general view of the system as a whole and to provide a starting point for developing the details of agent models and software implementations to satisfy the requirements. A new aspect in this methodology is that systems are developed through a series of levels of abstraction in which humans, with machine assistance, can manipulate abstractions at one level into abstractions at the next lower level. HIM methodology consists of two phases the *discovery* and *definition* phases. *The discovery phase* guides the identification of agent(s) (types) and their high-level

behaviour. The ultimate goal of this phase, apart from discovering the agents and their relationships, is to produce models that capture the high-level structure and behaviour of the system. *The definition phase* produces implementable definitions. The goal is to get a clear understanding of the behaviours, the entities that participate in exhibiting these behaviours and their interrelationships, as well as inter-agent conversations and commitments.

2 HIM Methodology

We have five models that are generated by the HIM methodology. Figure 1 shows these models and the traceability between them.

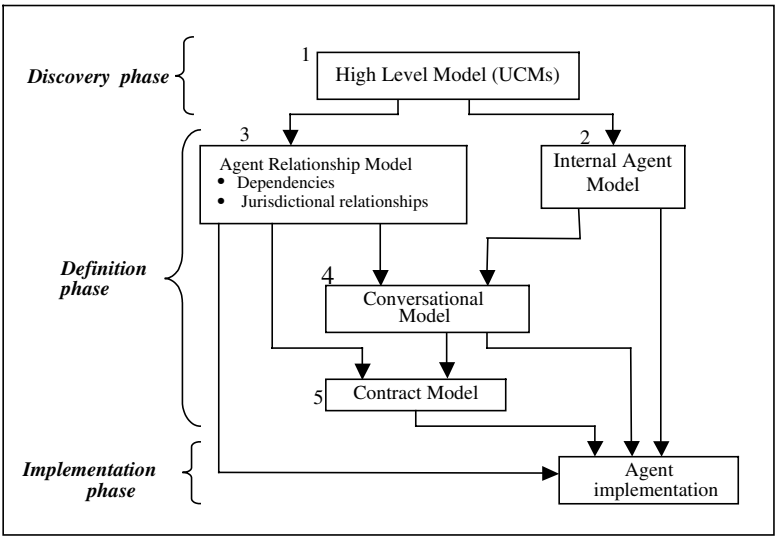


Fig. 1. Models of The HIM methodology.

The *high-level model* identifies agents and their high-level behaviour. It gives a high-level view of the system and provides a starting point for developing the details of the other models. Tracing application scenarios that describe functional behaviour, discovering agents and behavioural patterns along the way generates it. The *internal agent model* describes the agents in the system in terms of their internal structure and behaviour. It captures agent aspects such as goals, plans, tasks and beliefs. The internal agent model is derived directly from the high level model. The *relationship model* describes agent relationships: *dependency* and *jurisdictional*. The *conversational model* describes the coordination among the agents. The *contract model* defines a structure that captures commitments between agents. Contracts can be created when agents are instantiated or during execution as they are needed.

The high-level model is the only model that is associated with the discovery phase. The rest of the models are associated with the definition phase, which is not the topic of this paper.

3 Medical Diagnosis System

The diagnostic process is a distributed and cooperative work involving many different specialists, departments, and knowledge sources.

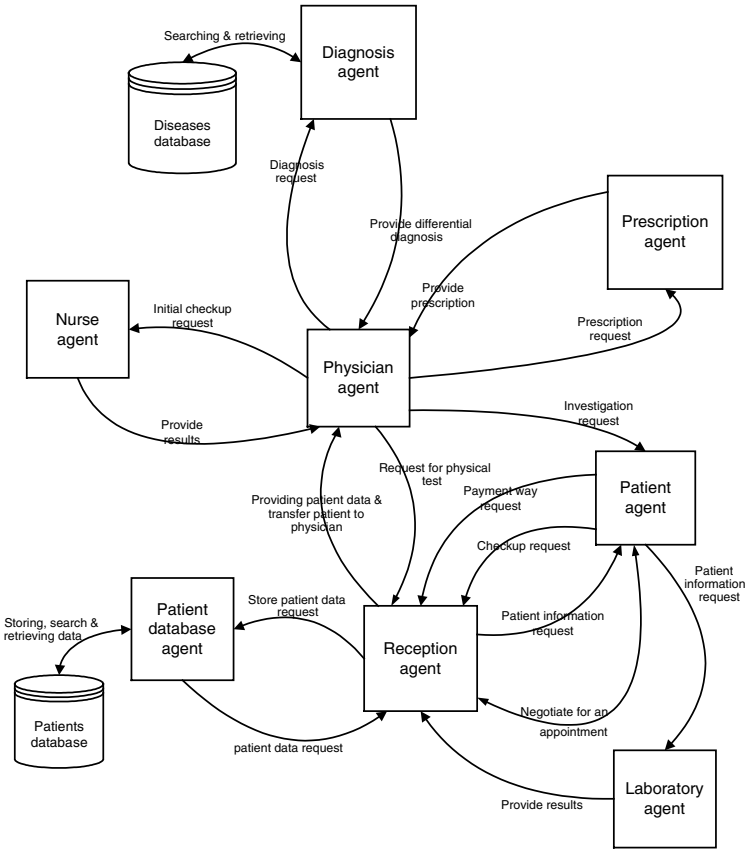


Fig. 2. General view of Multi-agent medical diagnostic system.

Before analyzing the cooperative relations in medical diagnosis, we should first describe the agent system as a whole and then we should discuss the diagnostic process. Figure 2 describes a general view of Multi-agent medical diagnostic system. It consists of set of agents, which cooperate together to achieve the diagnosis process. Behind the practice of medical diagnosis is the theory of *differential diagnosis*. Differential diagnosis operates within an uncertainty-based theoretical framework. It is a highly developed theory and is also frequently used. There are two important phases to the diagnostic procedure: hypothesis formation and evidence gathering. The hypothesis formation phase begins as soon as the patient enters the doctor's office. The patient may complain of more than one key symptom.

However, the selection of the key symptoms is tentative, because evidence gathered further along the diagnostic process may cause the doctor to rethink the selection. It is important to emphasize that a key symptom is usually caused by more than

one disease. In fact, a one to one relationship is rare. Because of its rarity, diagnosis is fundamentally a process of finding evidence to distinguish a probable cause of the patient's key symptoms from all other possible causes of the symptom.

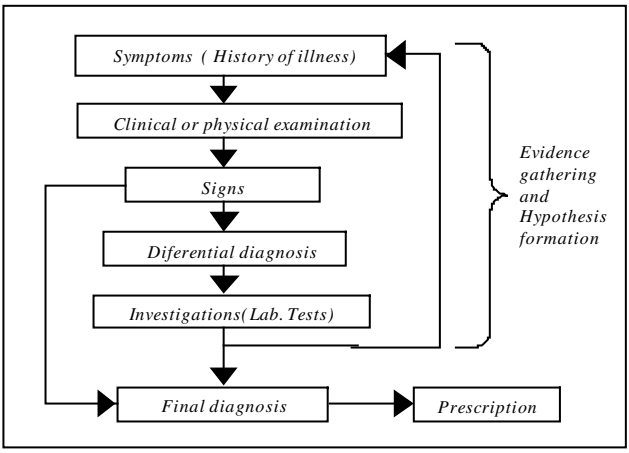


Fig. 3. Summary of the differential diagnosis.

This process is called *differential diagnosis*. The differential diagnosis may have more than one disease as a working hypothesis. In this case, extra investigations are needed to confirm which one is established as a final diagnosis. With a key symptom in hand, the doctor refers to a medical textbook, which leads the doctor to have a set of disorders. In turn, each disorder has a set of possible hypotheses as to the cause of the disease. In choosing one cause from the set or list, the doctor sets one disease or cause apart as the most likely cause of the symptom. This becomes the doctor's working hypothesis, and it is a tentative designation.

The doctor then starts to gather evidence in support of the working hypothesis, always keeping in mind the set of alternative hypotheses. The *evidence-gathering* phase is divided into three parts: *history*, *physical examination* and *investigations such as (laboratory tests, X-rays, MRI and so on)*.

The history of the illness involves the doctor questioning the patient about the history of the symptoms, the medical history of the patient, and other factors.

During *the physical examination* part, the doctor looks for *signs* of the disease, which are obtained through a physical examination of the patient. *Signs* are manifestations of the disease, which the doctor can see or feel.

If the evidence is not conclusive, the doctor may call for investigations such as *laboratory tests,* (both tests of body tissues, fluids etc.) and photographic monitoring such as (*X-rays*, *computed tomography*, *image scans* and the like). This step-by-step procedure in differential diagnosis allows the doctor to establish a *working hypothesis*, thus giving the procedure a structure and a direction for evidence gathering.

The gradual step-by-step decrease in the number of hypotheses the doctor considers keeps the doctor's mind open to all the evidence from the beginning to the end of the procedure. The diagnostic procedure is based on maximizing uncertainty so that all options are considered in the diagnosis. Figure 3 illustrates the summary of the differential diagnosis steps.

4 Agent System View (High Level Model)

Developing and understanding complex systems is not easy to achieve by traditional software systems that concentrate on low level details. The main goal of the use of a high level view is to understand the entire system and its structure without referring to any implementation details. We use Use-Case Maps, which are suitable for high-level visual representations, as a starting point for generating more detailed visual descriptions. UCMs are used to model the high-level activities because of their ability to simply and successfully depict the design of complex systems, and provide a powerful *visual* notation for a review and detailed analysis of the design.

4.1 Use Case Maps (UCMs)

UCM notation helps persons to visualize, think about and explain the overall behavior of a whole system. The center of attention of UCM is not the details. It describes scenarios in terms of causal relationships between responsibilities. It also emphasizes the most relevant, interesting and critical functionalities of the system.

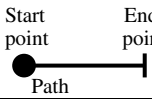
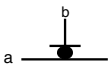
UCMs describe the complex systems at a high level of abstraction. The complex system requirements are captured as UCM scenarios integrated in one model with stubs and plug-ins. Table 1 shows basic UCM symbols [4] of the type that are used within this paper.

4.1.1 UCM Notation with a Simple Example

UCMs are precise structural entities that contain enough information in highly condensed form to enable the visualization of system behavior. It provides a high level view of causal sequences in the system as a whole, in the form of paths. The causal sequences are called scenarios. In general, UCMs may have many paths. The figures in this paper only show one, for reasons of simplicity. Figure 4 shows an example of UCMs. The simple example is called "**Money withdrawal using ATM *Automatic Teller Machine***" where a scenario starts with a triggering event or a pre-condition (filled circle labeled *Customer wants to withdraw money)* and ends with one or more resulting events or post-conditions (bar labeled *logon rejected*, *withdraw rejected*, *slip printed* and *money withdrawn*). The path starts with a filled circle, which indicates a starting point of a path, the point where stimuli occurs causing movement to start progressing along the path until the end point of a path is reached. Paths define causal sequences between start and end points. The causal sequences connect stubs and responsibilities, indicated by named points along the paths *Insert card, Ask PW, Enter PW, Validate, Enter amount, Chk, Debit, Print, Notify and Provide*. Think of responsibilities as tasks or functions to be performed, or events to occur.

In this example, the activities can be allocated to abstract components: *Customer, ATM interface, Account, Printer* and *Dispenser*, which can be seen as objects, agents, processes, databases, or even roles or persons. Paths may cross many components and components may have many paths crossing them. When maps become too complex to be represented as one single UCM, UCM may be decomposed using a generalization of responsibilities called stubs. *Stubs* link to sub-maps called *plug-ins*. Stubs may be positioned along paths like responsibilities but are more general than responsibilities in two ways: they identify the existence of sub-UCMs and they may span multiple paths (not shown). A stub can be static or dynamic.

Table 1. Basic UCM symbols.

UCM Notation	Notation Explanation
	**Path:** Represents flow of events in the system, path, connects start points, stubs, responsibilities, forks, and end points of UCM. The start-point represents preconditions. The end-point represents post-conditions.
	**Responsibility point:** Represents the functions to be accomplished by the system at that point of the path.
	**Or Fork:** An OR fork means the path proceeds in only one out of two or more directions.
	**Or Join:** it means two or more paths merged it in one single path.
	**And Fork:** it means that a single path is distributed at the same time into many concurrent paths.
	**And Join:** it means that several concurrent Paths merged at the same time into a single path.
	**Static stub:** associated with one plug-in (Sub UCM) as task to be achieved by the system, used as decomposition of complex maps.
	**Dynamic stub:** associated with several plug-ins, whose selection can be determined at run-time according to selection policy (often described with preconditions). It is also possible to select multiple plug-ins at once (sequentially or parallel).
	**Wait point**: Path a waits for an event from path b.
	**Agent:** Software component representing a software agent.

Static stubs contain only one plug-in and enable hierarchical decomposition of complex maps. *Dynamic stubs* are shown as dashed outline to distinguish them from stubs that are used for static stubs. Dynamic stubs may represent several plug-ins, whose selection can be determined at run-time according to a selection policy often described with pre-conditions.

It is also possible to select multiple plug-ins at once sequentially or in parallel. A plug-in may involve additional system components not shown in the main UCM. Start points may have pre-conditions attached, while responsibilities and end points can have post-conditions.

In Figure 4, the *Validate* stub has two outgoing ports **a** and **b**. Port **a** which means authentication was accepted. Port b means authentication was not accepted. There are two plug-ins associated with the *Validate* stub: *Fingerprint* and *Password*.

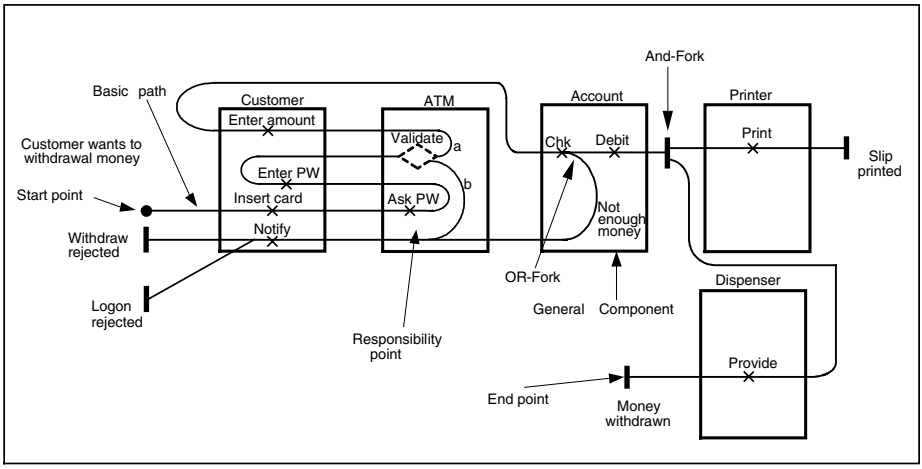


Fig. 4. UCM scenario for money withdrawal with stubs.

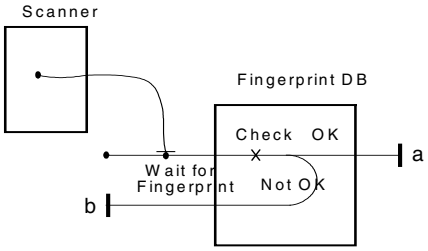


Fig. 5. Fingerprint plug-in for the validate stub.

The *Fingerprint* plug-in illustrated in Figure 5 describes the behavior when the validation is performed by fingerprint. The plug-in starts with a *wait* for *fingerprint*, which waits till the customer enters his fingerprint.

Then the path proceeds to the *Check* task, which is followed by an or-fork in the path if the entered fingerprint was found to match the stored fingerprint, the path labeled *Ok* is followed to the end point **a**. Otherwise, the path labeled *not Ok* is followed to the end point **b**.

The other plug-in is the *Password* plug-in shown in Figure 6, which is used when the customer enters his password instead of his fingerprint. The plug-in starts with *Wait for PW*, which waits for the customer to enter a password and then the *Check PW* task is performed. The path is split into three paths after the *Check PW* responsibility. The first fork (labeled *PW OK*) is followed when the entered password matches the customer's stored password. The second fork (labeled *PW Not OK*) is followed when the entered password is incorrect. The third fork is followed if the customer is allowed to retry to enter the password after it is found to be incorrect.

4.1.2 High-Level Model (Medical Diagnostic System Scenarios)

In the high-level model, we describe use case maps of the medical diagnostic system and we state how UCMs can be used to represent agent systems. Also we apply

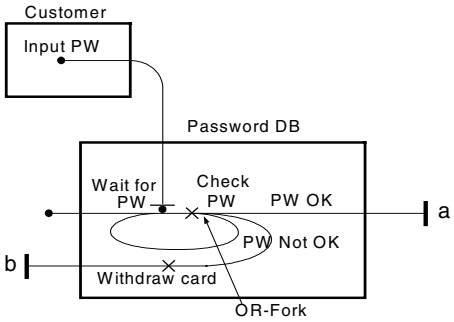


Fig. 6. Password plug-in for the validate stub.

UCMs to capture an agent based disease diagnosis system and explain how UCMs describe the system scenarios in visual views. The following scenarios represent interactions between some agents in the system. Examples of interactions shown are patient agent with reception agent, patient agent with physician agent and physician agent with diagnosis agent. By tracing application scenarios the high-level model is derived. These scenarios describe functional behavior, as UCM paths within the system. This discovers agents, responsibilities, and plug-ins along the way. Generally, one starts with some use cases and some knowledge of the agents required to realize them. This model maintains the most important steps:

- Identify scenarios and major components involved in the system.
- Identify roles for each component.
- Identify pre-conditions and post-conditions to each scenario.
- Identify responsibilities and constraints for each component in a scenario.
- Identify sub scenarios and replace them with stubs.
- Identify agent collaborations for the major tasks.

4.1.2.1 Patient Agent and Reception Agent UCM

The UCM, shown in Figure 7 represents a basic scenario between a patient agent and a reception agent in our system. Patient agent represents the patient in the application environment, and the reception agent represents the reception. The precondition for the patient agent and reception agent scenario is that the patient needs a checkup. The scenario starts with the *Checkup Request* stub, which hides the detailed information of the checkup request process.

Preconditions:	*Postconditions:*
• Patient needs a checkup.	• Patient transferred to physician. • Patient refused

The checkup request can be achieved in several ways. For example, it can be done by phone call or by Email. Therefore, the *Checkup Request* stub is represented as a dynamic stub. Figure 8 illustrates the plug-ins for the *Checkup Request* stub.

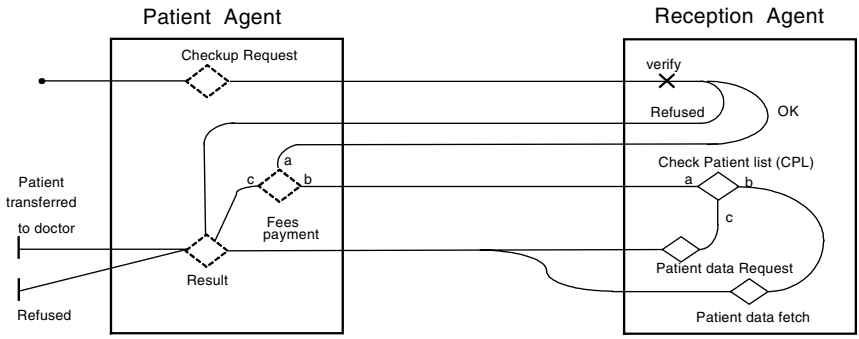


Fig. 7. UCM scenario between Patient agent and Reception agent.

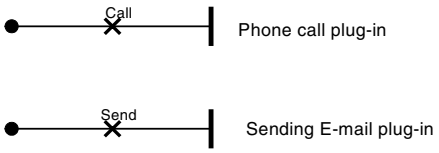


Fig. 8. Plug-ins for *Checkup Request* stub.

After all responsibilities for the checkup request process are performed, the path leads to the reception agent where there is a responsibility called *verify* which verifies whether the specialist whom the patient requests is available or not. Then the path leads to an or-fork immediately after the *verify* responsibility which indicates alternative scenario paths. One path leads to refuse the checkup request, e.g. because there is no specialist available who can examine the patient. Then the path leads to the result stub to inform the patient that the checkup is refused.

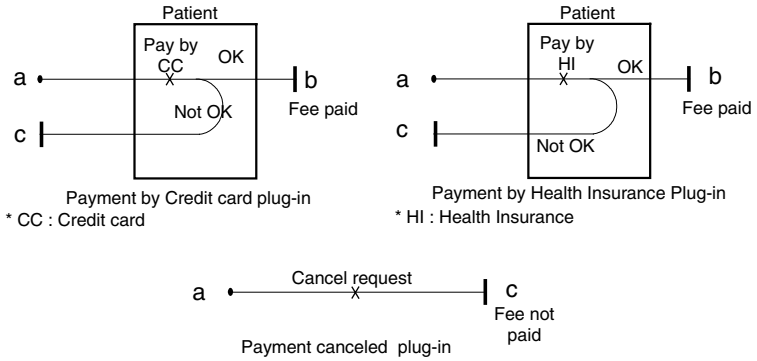


Fig. 9. Plug-ins for Fees payment stub.

The other path leads to accept the checkup request and the path proceeds to the Fees payment stub, which is concerned with the payment of the checkup fees. The *Fees payment* stub has two outgoing ports b and c. When fees are not paid, Port c is followed. In this case the path leads to the result stub to confirm that the checkup request is refused. Port b is followed when the fees are paid.

There are three plug-ins associated with the *Fees payment* stub illustrated in Figure 9: *Payment by Credit card* plug-in, *Payment by the Health Insurance* plug-in *and Payment canceled* plug-in.

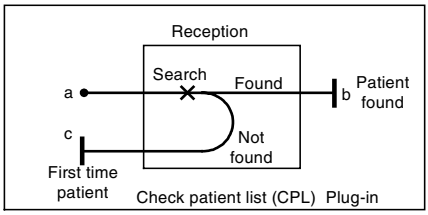


Fig. 10. Check patient list Plug-in.

After that the path leads to the *Check patient list* CPL stub, which is responsible for searching the patient database to find the history of the patient and all related information.

The CPL stub has two outgoing ports. If the patient is found, port b will be followed, which means that there is no need to enter the patient information again. Otherwise port c is followed, which means that this is the first visit of the patient. And then the path leads to the result stub, which is responsible to inform the patient about the results by sending E-mail or by a phone call. The result stub has two plug-ins shown in Figure 11: *Notify by phone* plug-in and *Notify by E-mail* plug-in.

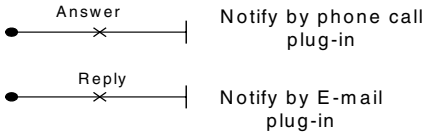


Fig. 11. Plug-ins for Result stub.

4.1.2.2 Patient Agent and Physician Agent UCM

In figure 12, the UCM shows the scenario between patient agent and physician agent. The physician agent represents the physician in the application environment. The precondition for starting the scenario is that the patient is transferred to the physician.

This scenario starts at the physician agent with the *Review History* (RH) stub. In this stub, the physician agent performs a review of the patient history in order to get the process of disease diagnosis started.

After that the path leads to the *ask* responsibility in the physician agent in which the physician agent starts to ask the patient agent about the symptoms the patient is aware of. The patient agent replies to the physician agent by the *answer* responsibility in the patient agent. After that the path leads to the *eval* responsibility in the physician agent, which causes the evaluation of the patient answers.

Pre-conditions:	*Post-conditions:*
Patient transferred to physician.	Patient successfully diagnosed and informed.

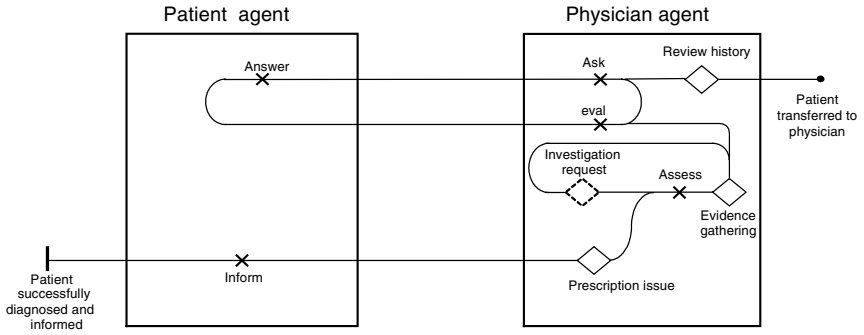


Fig. 12. Patient agent and Physician agent UCM.

After the *eval* responsibility there is an or-fork, which indicates alternative scenario paths. One fork path leads again to the cycle of the *ask* responsibility in case the physician agent wants to ask more questions.

When the physician agent determines that no further questions are needed, the other fork path leads to the *Evidence gathering* stub is followed. In this stub, the physician agent collects all evidence related to the disease to form the possible hypothesis of the disease. Also the *Evidence gathering* stub hides the connection between the physician agent and the diagnosis agent. Figure 13 shows the *Evidence gathering* plug-in. The plug-in starts with the *Diagnosis request* responsibility, which asks the diagnosis agent to examine the current patient case. Then the path leads to the stub "hypothesis formation" in the diagnosis agent, which is responsible for searching in the disease database for a match for the patient case symptoms.

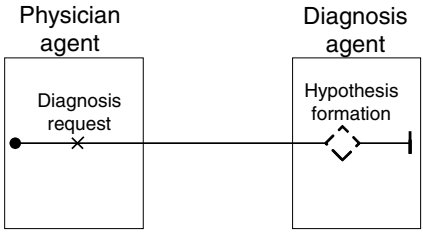


Fig. 13. Evidence gathering plug-in.

After that, the path leads to the *assess* responsibility which assesses whether the collected evidence is enough. If it is, the path leads to the *prescription* stub, which is responsible for issuing a prescription to the patient agent, and then informs the patient what the problem is, which is the post-condition of this scenario.

If the collected evidence is not enough, the path leads to the *Investigation request* stub, which performs some investigations that are required by the physician agent. Figure 14 shows the plug-ins for the *Investigation request* stub: *appointment for physical exam* plug-in, *Lab-work* plug-in and *X-Ray* plug-in. After the investigations, the path leads again to the *Evidence gathering* stub to assess whether the collected evidences are not enough.

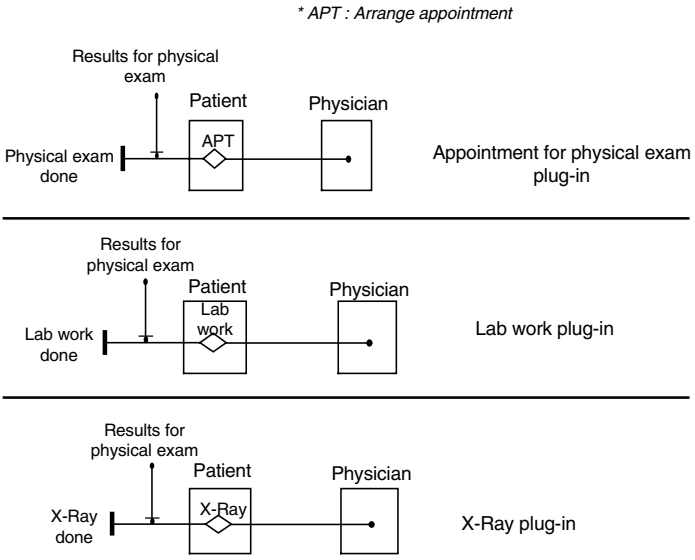


Fig. 14. Plug-ins for Investigation request stub.

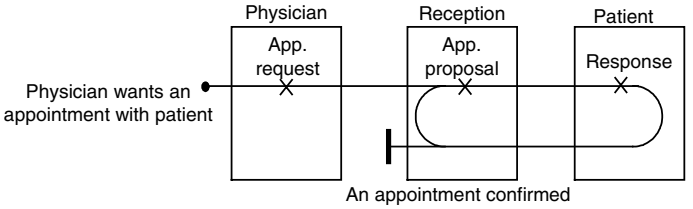


Fig. 15. Patient agent, Reception agent and Physician agent UCM.

4.1.2.3 Patient Agent, Reception Agent and Physician Agent UCM

This scenario starts when the physician wants to meet the patient to make a physical exam.

The physician agent asks the reception agent for an appointment request with the patient agent. The path leads to the reception agent, which negotiates with the patient agent for an appointment. Figure 15 shows the UCM scenario for that.

5 Conclusion

In this paper, we developed a Multi-agent medical diagnostic system using a notation called use case maps. The notation was found to be one of the most significant techniques for the process of developing complex systems. The approach chosen for the development is proven to be a practical solution for the problem of constructing systems, which are required to be flexible and working in dynamic environments. We have shown that the visual high-level view helps in visualizing, understanding, and defining the behaviour of a Multi-agent medical diagnostic system.

Currently we are developing models that capture the internal details of agents and their relationships. The models being developed capture the goals, beliefs, plans, relations, conversations, and agent commitments.

6 Future Work

In this paper, we described the high-level model that captures the high level behaviour of agent systems. In the future, we will try to develop the definition phase, which produces intermediate models that facilitate the implementation of a multi-agent system. And how we map from UCMs to the internal agent model and agent relationship model in straightforward manner. The high-level model, supplemented by other information, is used to generate these models. These models express the full functional behaviour of an agent system by identifying aspects of agents such as goals, beliefs, plans, jurisdictional and dependency relationships, contracts, and conversations.

References

1. Elammari, M. and Lalonde, W., *An Agent Oriented Methodology: High Level and Intermediate Models*, in Proceedings of the 1'st International Workshop on Agent Oriented Information Systems (AOIS 99), Heidelberg, Germany, June 1999.
2. Gunter Mussbacher, Daniel Amyot, *Acollection of patterns for Use Case Maps,* Mitel Networks, 350 Legget Dr., Kanata (ON), Canada.
3. D. Amyot, L. Logrippo, R.J.A. Buhr, and T. Gray (1999), *Use Case Maps for the Capture and Validation of Distributed Systems Requirements.* In: Fourth International Symposium on Requirements Engineering (RE'99), Limerick, Ireland, and June 1999.
4. R.J.A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski (1998), *High Level, Multi-Agent Prototypes from a Scenario-Path Notation: A Feature-Interaction Example. In: H.S. Nwana and D.T. Ndumu (Eds),* Third Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'98), London, UK, pp. 255-276
5. R.J.A. Buhr, M. Elammari, T. Gray, S. Mankovski, *A High Level Visual Notation for Understanding and Designing Collaborative, Adaptive Behaviour in Multi-agent Systems,* Hawaii International Conference on System Sciences (HICSS'98), Hawaii, January 1998.
6. Timothy Finin, CFP: 1st Int. Conference on Multi-agent Systems - ICMAS '95 Fri, 5 Aug 1994.
7. Jennings, N., Sycara, K., and Wooldridge, *M. A Roadmap of Agent Research and evelopment.* Int. Journal of Autonomous Agents and Multi-Agent Sys tems, l(l):7-38 (1998).
8. Use Case Maps Web Page and UCM User Group, 1999. http://www.UseCaseMaps.org
9. Buhr, R.J.A.: "*Use Case Maps as Architectural Entities for Complex Systems*". In: Transactions on Software Engineering, IEEE, December 1998, pp. 1131-1155. http://www.UseCaseMaps.org/UseCaseMaps/pub/tse98final.pdf
10. R.J.A. Buhr, R.S. Casselman, *Use Case Maps for Object-Oriented Systems,* Prentice Hall, 1996.
11. J.A. Barondess, C.C.Carpenter. *Differential Diagnosis*, Lea and Febiger, Philadelphia, PA, 1994.
12. M. Wooldrige, N. R. Jennings and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems*, volume 3, pp 285-312, Kluwer Academic Publishers, The Netherlands, 2000.
13. G. Bush, S. Cranefield and M. Purvis: *The Styx Agent Methodology*. In The Information Science Discussion Paper Series, Number 2001/02, January 2001, ISSN 1172-6024 http://citeseer.nj.nec.com/bush01styx.html

14. P. Burrafato, M. Cossentino - "*Designing a multi-agent solution for a bookstore with the PASSI methodology*" - Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) - 27-28 May 2002, Toronto (Ontario, Canada) at CAiSE'02 (www.csai.unipa.it/cossentino/paper/AOIS02.pdf)
15. Tropos web site http://www.cs.toronto.edu/km/tropos/
16. AgentUML web site http://www.auml.org/auml/
17. Iglesias, C., Garijo, M., and Gonzalez, J. *A Survey of Agent-Oriented Methodologies*. In Intelligent Agents V - Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg. 1999
18. Wooldridge, M. and Ciancarini, P. Agent-Oriented Software Engineering: The State of the Art. In Agent-Oriented Software Engineering. Ciancarini, P. and Wooldridge,M. (eds), Springer-Verlag Lecture Notes in AI Volume 1957, 2001.