A MULTI-AGENT SYSTEM FOR ADMINISTERING THE PRESCRIPTION OF ANTI-RETROVIRAL AND ANTI-TB DRUGS

Wilhelmina Johanna Kuyler

Dissertation submitted in fulfilment of the requirements for the Degree

MAGISTER TECHNOLOGIAE:

INFORMATION TECHNOLOGY

in the

School of Information and Communication Technology

Faculty of Engineering, Information and Communication Technology

at the

Central University of Technology, Free State

Supervisor: Prof. J.D.M. Kinyua, Ph.D.

Bloemfontein

November 2007

Declaration of Independent Work

I, WILHELMINA JOHANNA KUYLER, identity number [redacted], and student number 20469543, do hereby declare that this research project submitted to the Central University of Technology, Free State for the Degree MAGISTER TECHNOLOGIAE: INFORMATION TECHNOLOGY, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology, Free State; and has not been submitted before to any institution by myself or any other person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

W.J. Kuyler	DATE

Acknowledgements

I would like to express my gratitude to:

- Prof Johnson Kinyua, my supervisor, for encouraging me to become involved in research and for his ongoing support and positive feedback. I wish to thank him for his dedication and all the time he spent directing me and reading through my work.
- Dawid, for his encouragement and for always believing in me.
- Admill, Francois and Jenny, for making everything worthwhile.
- Everybody that contributed to the project by giving me ideas, information, or any kind of assistance.
- My heavenly Father, the Giver of all wisdom, for giving me the strength to complete this work.

Summary

Multi-agent systems (*MAS*) consist of a number of autonomous agents that communicate among themselves to coordinate their activities in order to solve collectively a complex problem that cannot be tackled by any agent individually. These kinds of systems are appropriate in many domains where problems that are complex, distributed and heterogeneous require communication and coordination between separate autonomous agents, which may be running on different machines distributed over the Internet and are located in many different places.

In the health care domain, *MAS* have been used for distributed patient scheduling, organ and tissue transplant management, community care, decision support, training and so on. One other promising area of application is in the prescription of antiretroviral and anti-TB drugs. The drugs used to treat the two diseases have many and similar side effects that complicate the prescription process. These factors have to be considered when prescribing medication to a person co-infected with HIV and tuberculosis. This is usually done manually using drug recommendation tables, which are complicated to use and require a great deal of decision-making. The design and implementation of a multi-agent system that assists health care staff in carrying out the complex task of combining antiretroviral and anti-TB drugs in an efficient way is described.

The system consists of a number of collaborating agents requiring the communication of complex and diverse forms of information between a variety of clinical and other settings, as well as the coordination between groups of health care professionals (doctors, nurses, counsellors, etcetera.) with very different skills and roles. The agents in the system include: patient agents, nurse agents, lab agents, medication agents and physician agents. The agents may be hosted on different machines, located in many different places distributed over the Internet. The system saves time, minimises decision errors and increases the standard of health care provided to patients.

Opsomming

Multi-agent stelsels (*MAS*) bestaan uit ‘n aantal outonome agente wat onder mekaar kommunikeer om hulle aktiwiteite te koördineer en sodoende gesamentlik ‘n komplekse probleem op te los wat enige agent op sy eie nie sou kon aanpak nie. Hierdie soort stelsels is geskik vir baie terreine waar die probleme kompleks, verspreid en heterogeen is. Dit benodig kommunikasie en koördinasie tussen afsonderlike outonome agente wat op verskillende rekenaars verspreid oor verskillende plekke op die Internet mag wees. Op die terrein van gesondheidsorg word *MAS* reeds gebruik vir verspreide skedulering van pasiënte, bestuur van orgaan- en weefseloorplanting, gemeenskapsorg, ondersteuning van besluitneming, opleiding ensovoorts. ‘n Ander belowende toepassingsgebied is die voorskryf van antiretrovirale en anti-TB middels. Die geneesmiddels wat gebruik word om die twee siektetoestande te behandel, het baie newe-effekte, wat ook baie ooreenkom en die keuse van medikasie vir ‘n voorskrif kompliseer. Hierdie faktore moet in ag geneem word wanneer medikasie voorgeskryf word vir ‘n persoon wat geïnfekteer is met beide MIV en tuberkulose. Dit word gewoonlik deur mense gedoen met behulp van medikasie-aanbevelingstabelle, wat ingewikkeld is om te gebruik en baie besluitneming verg. Die ontwerp en implementering van ‘n multi-agent stelsel wat gesondheidsorgpersoneel ondersteun in die uitvoering van die ingewikkelde taak om antiretrovirale en anti-TB middels op ‘n doeltreffende wyse te kombineer, word hier beskryf. Die stelsel bestaan uit ‘n aantal agente wat saamwerk en die kommunikasie van komplekse en diverse vorms van inligting tussen ‘n verskeidenheid kliniese en ander omgewings benodig, sowel as die koördinasie tussen groepe beroepslui in gesondheidsorg (geneeshere, verpleegsters, beraders, ens.) met vaardighede en rolle wat baie van mekaar verskil. Die agente in die stelsel sluit in: pasiënt-agente, verpleegster-agente, laboratorium-agente, medikasie-agente en geneesheer-agente. Die agente kan op verskillende rekenaars wat verspreid oor verskillende plekke op die Internet geleë is, gehuisves word. Die stelsel bespaar tyd, verminder besluitnemingsfoute en verbeter die standaard van gesondheidsorg wat aan pasiënte verskaf word.

Table of Contents

Chapter 1

C h a p t e r 1

1 INTRODUCTION

1.1 Overview

Recently, agent-oriented software development has been used as a paradigm for software engineering in a wide variety of applications (Jennings & Wooldridge, 1998). The agent concept constitutes a powerful abstraction tool for dealing with software development. It also provides a good development paradigm in domains that are complex, open and distributed, such as the Internet, which requires agents to act autonomously, learn, compete, and even cooperate (Wooldridge & Jennings, 1995).

Agents are typically defined by considering the characteristics that they should have, which include the following: autonomy, reactivity, sociability, mobility, rationality, and pro-activity (Wooldridge, 2002). In most cases, the solution to complex distributed problems developed using a Multi-agent system (*MAS*) approach is composed of multiple interacting agents acting as computing elements to achieve the system's goals. Agents are being used in an increasingly wide variety of applications, such as intelligent user interfaces, personal assistants and for handling large amounts of information, such as e-mail and Internet searches. They are also used in electronic commerce, process control, air traffic control, enterprise resources planning, health care and so on (Jennings et. al, 1998). In the health care domain they have been used for distributed patient scheduling, organ and tissue transplant management, community care, decision support, training and so on (Nealon & Moreno, 2004).

The health care domain at all levels (local, regional, national and international), is characterised as being a vast, open environment where health care professionals such as doctors and nurses, with different skills and roles, manage patients via shared and distributed decision making. Such environments require the communication of complex and diverse forms of information between a variety of clinical and other settings. Health care professionals in such environments, in particular, require information that is both timely and error-free, and recommendations or decisions offered by software systems have to be secure and trustworthy. Software agents can be used to provide information to doctors, nurses and patients. There are information agents (also called Internet agents) that are specialised in retrieving information from different sources, analysing the data, selecting the information of interest to the user, filtering redundant or irrelevant information, and presenting the appropriate information to the user.

The components of a multi-agent system may be running on different machines, located in many different places distributed over the Internet. Each of the agents may keep part of the knowledge required to solve the problem, such as patient records held in different departments within a hospital or in several hospitals, clinics, surgeries, or in a government department.

Methodologies to aid in the development process, tools and platforms that facilitate the implementation of agent systems have also been developed. The methodologies currently available include: *Gaia* (Zambonelli et. al 2003), *MaSE* (Wood, 2000), *Tropos*, *PASSI*, *Agent-UML*, and so on. See (Weiss, 2002) for a survey of Agent-Oriented Software Engineering (*AOSE*) methodologies and the field of *AOSE* in general. Many agent platforms are also available and good examples include the following: *ZEUS*, *JAS* (Java Agent Services API), *JADE*, *Aglets*, *Concordia*, *Voyager*, *ADK* (Agent Development Kit), *FIPA-OS*, *Madkit*, *agentTool*, *AgentAcademy*, *BlueJADE*, *Odyssey*, *Jumping Beans*, *Grasshopper*, *Swarm* and *JAMES* (Bordini et. al, 2005).

According to the World Health Organization Report published in 2005 (UN Chronicle Online Edition, 2005), it is estimated that one third of the 40 million people living with HIV/AIDS worldwide are co-infected with tuberculosis (TB).

HIV affects the immune system, increases the likelihood of people acquiring TB infection, and promotes the progression of latent TB infection to an active disease. People that are HIV-positive are up to fifty times more likely to develop tuberculosis in a given year than those who are HIV-negative. Without proper treatment, approximately 90 per cent of those living with HIV die within months of contracting TB. The two diseases speed up the progress of each other and TB kills up to 50% of all AIDS patients worldwide.

Joint HIV/TB intervention is an item on the WHO list of key areas for HIV/AIDS programming in the health sector (World Health Organization, 2005). The recommended interventions provided in this list are considered essential and have proven highly effective in restricting the HIV epidemic in many locations. The extent of the TB and HIV/AIDS epidemics is of such a nature that in the future additional health care staff will be needed and existing staff will need to be trained or re-trained to ensure that adequate joint interventions take place.

Anti-TB drugs may cause some side effects such as jaundice, hepatitis, rash, fever, bleeding, decreased vision and dizziness, etc. The side effects of antiretroviral drugs are numerous and include liver failure, pancreatitis, metabolic complications, fever, rash, flu-like symptoms, dizziness, insomnia, diarrhoea, nausea, headache, muscle aches, etcetera. Many of the side effects of the drugs used in the treatment of the two conditions are the same and may be increased when a combination of drugs is taken. The complexity of the matter is increased by the fact that many of these symptoms may be caused by TB or AIDS. Interaction also occurs between antiretroviral and anti-TB drugs, which prohibits the use of certain combinations of drugs, or call for a dose change.

All these factors have to be considered when prescribing medication to a person co-infected with HIV and tuberculosis. This is done according to drug recommendation tables, which are complicated to use and require a great deal of decision-making.

1.2 Statement of the Problem

The complicated nature of the two medical conditions has to be considered when prescribing medication to a person co-infected with HIV and TB. At present most clinics and hospitals in South Africa treat HIV and TB patients according to the PALSA Plus (Practical Approach to Lung Health South Africa Plus ART) guidelines for treating patients with TB and HIV.

The first diagnosis of HIV is usually done at a primary health care clinic. If the patient is found to be HIV positive, he is sent to an HIV clinic for further tests, to establish if he qualifies for ARV treatment. If the patient qualifies for ARV treatment, the patient visits a doctor who will do a baseline assessment. Before starting ARV treatment, the patient has to complete a drug readiness training programme for three weeks. Then the doctor prescribes the medication, and the prescription is given to the primary health care clinic that has to provide the medicine. The prescription is sent through to the pharmacist, who will verify that the combination and dose of medicines are correct. The medicine is dispensed and sent to the clinic, where the patient will receive it.

The patient has to visit the doctor and clinic at regular intervals. At each visit the doctor or nurse will do some tests. Thereafter the doctor will assess the condition of the patient and revise the prescription if necessary.

All TB treatment in South Africa is done at TB clinics and combination pills are used. At this stage TB clinics and HIV clinics work in parallel, without interaction.

Health workers at clinics and government hospitals as a rule suspend anti-retroviral treatment until TB treatment is completed. They are not able to treat the two conditions simultaneously, because it is too complicated and requires a great deal of decision-making. If a patient already receiving antiretroviral drugs also contracts TB, the patient has to be treated by a health worker with special knowledge of the medication interactions. Prescriptions are then prepared according to drug recommendation tables, which are complicated to use and require a great deal of decision-making.

1.3 Motivation for the Research

The complicated task of prescribing anti-retroviral and anti-TB drugs can be made easier by using a computer application that is able to assist in complex decision making. Intelligent agents and multi-agent systems are a new technology for the development of complex, distributed and heterogeneous information systems. The health care domain is a well suited environment for applying multi-agent systems.

1.4 Hypothesis

The prescription of anti-retroviral and anti-TB drugs can be made significantly faster and more accurate by deploying a multi-agent system to generate prescriptions to save time and minimise decision errors.

1.5 Research Goals and Objectives

The goal of this research is the development of a *MAS* system for administering the prescription of anti-retroviral and anti-TB drugs, in order to assist health care staff in carrying out the complex task of combining antiretroviral and anti-TB drugs in an efficient way. The system is expected save time, minimise decision errors and increase the standard of health care provided to these patients.

The objectives related to this goal are to:

1. Consider the application of agents in different fields.
2. Design a multi-agent system for the prescription of anti-retroviral drugs.
3. Implement the system and show how the system is used.

1.6 Research Methodology

The system has been developed using the *MaSE* methodology for agent-oriented software engineering. This methodology covers the whole development life cycle and has a support tool called *agentTool*. The *JADE* (Java DEvelopment Framework) was chosen for the implementation.

1.7 Organisation of Dissertation

The rest of this dissertation is organised as follows: In Chapter Two the concept of an agent, its characteristics and the application domain characteristics for Multi-Agent Systems are described. Chapter Three concentrates on related work, where the application of *MASs* in various domains is reviewed. It includes applications of multi-agent systems in different domains such as industrial, information management, e-commerce and health care. Chapter Four discusses the design of the *MedAgent* system. The method of analysis and design is described in detail. The implementation of *MedAgent* in *JADE* is presented in Chapter Five. The conclusions and suggestions for further work are presented in Chapter Six.

C h a p t e r 2

2 MULTI-AGENT SYSTEMS

2.1 The Concept of Software Agents

Computers are increasingly connected to one another and to larger networks, including the Internet. This is beneficial, in the sense that information and communication are much more easily available than in the past. However, the disadvantage is that there is sometimes too much information and everything becomes cluttered. Software is also becoming bigger, more complex and difficult to manage. As software evolves, it attempts to represent the real word as closely as possible, which means that programmes have to be developed to work in a dynamic environment, susceptible to change, and where information is uncertain and incomplete. The World Wide Web is an example of an information environment that is complex, open, distributed and heterogeneous.

There is a growing need for tools to help with computer and network systems management. Very large and complex software projects can be made more comprehensible by decomposing them into smaller components, probably independent of each other. In order to handle dynamic and open systems it is important to have autonomous software systems that work independently, but also communicate with one another. These two concepts, software components as well as autonomous software, are both part of the agent technology.

The concept of intelligent agents is a fairly recent development in software engineering. Agents became a buzzword in the popular computing press in 1994 (Nwana et al., 1999) and it was seen as “the new revolution in software”. This was also approximately the year when the wider public started to use the World Wide Web more generally. The first International Conference on Multi-

Agent Systems was held in San Francisco in 1995 (Wooldridge & Decker, 2000).

Shoham (Shoham, 1997) describes a software agent as a software entity that functions continuously and autonomously in a particular environment often inhabited by other agents and processes. Agents need the ability to communicate and interact with other agents, that is, to be social (Wooldridge, 2002). Agents may be stationary or mobile, which means that they either reside on individual computer systems or travel from host to host across the Internet to carry out different tasks.

Recently, agent-oriented software development has been used as a paradigm for software engineering in a wide variety of applications (Jennings et al., 1998). Agents have found application in numerous domains as explained in Section 2.4 below. For example, agents have been used for intelligent user interfaces, personal assistants and for handling large amounts of information, such as e-mail and Internet searches. They are also being used in electronic commerce and other complex, distributed domains.

2.2 The Concept of Multi-Agent Systems

A multi-agent system (*MAS*) is defined (Durfee & Lesser, 1989) as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver. In a broader sense, all systems in which agents interact can be called multi-agent systems.

A *MAS* has the ability to solve problems that are too large for a single agent and also enhances speed and reliability. These problem-solving activities have requirements such as coordination, negotiation and communication.

Zambonelli et al. (Zambonelli et al., 2003) distinguish between two main classes of *MASs*, that is, distributed problem solving systems in which the component agents are explicitly designed to cooperatively achieve a given goal; and open systems in which agents are not co-designed to share a common goal. Distributed problem solving systems are applied to problems that are computationally intensive, where each agent can be assigned a specific portion of the task. In this class of *MAS* the system is closed, the agents know and trust each other in their interactions, and they collaborate to accomplish the goal of the system as a whole. Open systems consist of agents with different objectives that dynamically enter and leave the system. In these systems agents have to make use of services, knowledge and capabilities found in other agents spread throughout the network, for example e-commerce agents and web service agents. In these cases agents must take into account that other agents may be self-interested or competitive and cannot always be trusted.

A *MAS* must provide a general communication mechanism to enable agents to coordinate their actions and to negotiate with each other. This has to be considered in the analysis and design of the system.

2.3 Characteristics of Agents

There is no one standard definition of an agent and no real agreement on the question of exactly what an agent is. They are typically defined by considering their characteristics (Wooldridge, 2002). According to Wooldridge, an intelligent agent is a system that is capable of flexible, autonomous action. Flexibility means that a system is reactive, pro-active and social.

According to Faltings (Faltings, 2000), agents turn software components into processes that are *autonomous*, *proactive, embedded* and *heterogeneous.* He describes a number of characteristics beyond these, which are associated with *intelligent* agents, namely: *adaptive*, *learning*, *rational* and *communicating*.

The characteristics of intelligent agents that are also mentioned on the web site www.openclinical.org (OpenClinical, 2006) include the following: autonomy, pro-activity, reactivity, communication and co-operation, negotiation and learning.

Some of the above-mentioned characteristics are discussed in more detail below.

2.3.1 Autonomy

An agent is situated in a certain environment where it acts and executes its specific task to achieve a certain goal. It has its own internal thread of execution, and it decides for itself which actions should be performed at what time without direct instructions from outside. An agent has the ability to act on behalf of users without the direct intervention of humans. It can monitor events, make observations about changes within its environment and react to them without requiring explicit commands. It has control over its own actions and internal state.

2.3.2 Reactivity

A reactive system is one that maintains an ongoing interaction with its environment. It can examine changes in the environment and react to external events. According to the changes they observe, intelligent agents can adapt their behaviour and make appropriate decisions about the tasks that have to be carried out to accomplish their goals. The responses to the changing environment have to be done in time for them to be useful.

2.3.3 Sociability

Sociability in agents is the ability to behave socially, to interact and communicate with other agents via some kind of agent communication language. Agents exchange information, receive instructions and give responses, and co-operate when it helps them to fulfil their own goals. They can also negotiate by means of organised conversations to achieve co-operation with other agents.

2.3.4 Pro-activity

Being pro-active is more than just reacting to the environment. An agent that is pro-active will not only be driven by events or act in response to its environment, but will take the initiative to make changes where appropriate, and also deliver them in a timely manner. Agents often act in a dynamic and unpredictable environment, where they are capable of flexible and autonomous action in order to meet their design objectives. To be flexible, agents have to be pro-active. They should have the ability to anticipate the information and knowledge needs of users and to adapt accordingly. They should use the opportunity to accomplish new goals at the appropriate time.

2.3.5 Learning

Intelligent agents have the ability to learn from the environment in which they are embedded while they are executing tasks. As a result of this interaction an agent can learn from previous experiences to improve its performance on the same task over time.

2.4 Agent Application Domain Characteristics

According to Jennings and Wooldridge (Jennings & Wooldridge, 1998), new technology can only be considered to be useful in the market place if it can offer one of two things:

- It must solve new problems that could not be solved using existing technology, or problems that would be too difficult, time-consuming, or risky to solve using existing technology; or

- It must have the ability to solve problems for which solutions already exist in a significantly better way that will make it much cheaper, solve it in a more natural way or provide an easier, more efficient or faster solution.

The use of agents can be justified if they can be applied to accomplish any of the above-mentioned.

2.4.1 Agents used for solving new Problems

Reactive systems, which maintain an ongoing, independent interaction with some environment, are complex and difficult to design and implement correctly. To develop this kind of system new techniques are required. These systems can broadly be divided into three classes:

- Open systems, in which the structure of the system itself is capable of dynamically changing. The components of these systems are not known in advance and may be highly heterogeneous. The Internet is an example of an open software environment. A computer system that must operate on the Internet, must be able to handle the changes without constant guidance by users.

- Complex systems, which are difficult to develop. Two tools for handling complexity in software development are modularity and abstraction. Agents make systems modular, because they are specialised components that cooperate with one another. Each agent solves a particular problem in the most appropriate way and is not forced to use a common uniform approach. The idea of autonomous agents provides the abstraction needed in a complex system, which makes it a concept that is appropriate for the development of complex systems.

- Ubiquitous systems in which the computer system should cooperate with the user as an equal partner to achieve a goal instead of receiving instructions from the user to the smallest detail. To achieve this, systems need to be autonomous, proactive, responsive and adaptive. Intelligent agents have all these properties and can be used to develop these kinds of systems.

2.4.2 Agents used for improving the Efficiency of Software Development

Agent technology can provide a way of conceptualising and implementing a given application that is an improvement on previous technology. Three important domain characteristics are often cited as a rationale for adopting agent technology in existing systems (Bond and Gasser, 1988):

- Data, control, expertise or resources are inherently distributed.
- The system is naturally regarded as a society of autonomous cooperating components.
- The system contains legacy components, which must be made to interact with other, possibly new, software components.

These domain characteristics are discussed below.

2.4.2.1 Distribution of Data, Control, Expertise or Resources

Real-world entities are often distributed and they need to interact with one another to solve problems. Distributed autonomous agents with their own resources and expertise provide a natural way of modelling this problem. They can carry out significant amounts of processing at the data source and can react with one another to accomplish certain goals.

2.4.2.2 The System regarded as a Society of autonomous cooperating Components

The concept of an autonomous agent can be used to present certain software functionality. A programme that filters e-mail can be presented to the user as a personal digital assistant, while other software (such as meeting scheduling) acts as an autonomous, social agent that interacts with similar agents.

2.4.2.3 Legacy Components

It is very difficult and expensive to update all legacy components of a system, but it is necessary to maintain the functionality of a system. A way to keep legacy software useful is to incorporate them into a wider cooperating community, that is, to build an agent wrapper around the software, which will exploit it to communicate with other systems.

2.5 The Applications of Multi-Agent Systems

Agents are being used in an increasingly wide variety of applications, for example, intelligent user interfaces, personal assistants and for handling large amounts of information such as e-mail and Internet searches. There are information agents (also called Internet agents) that are specialised in retrieving information from different sources, analysing the data, selecting the information of interest to the user, filtering redundant or irrelevant information, and presenting the relevant information to the user.

They are also being used in electronic commerce, process control, air traffic control, enterprise resources planning, health care and so on (Jennings et al., 1998). In the health care domain, they have been used for distributed patient scheduling, organ and tissue transplant management, community care, decision support, training and so on (Nealon & Moreno, 2004). Further descriptions of the application of agents in various domains are given in Chapter 3.

The specification of agent behaviour is an important part of multiagent system development. The agents in a *MAS* need to work together as a group to solve a problem, or communicate and negotiate. It is important that agents must be able to find each other and use a communication protocol that is familiar to all participating agents.

2.6 *AOSE* Methodologies that aid in the Development of *MAS*

New methodologies and tools had to be developed to aid in the development of multi-agent systems, resulting in agent-oriented software engineering (*AOSE*). Several methodologies have been described in the literature. These methodologies include *AUML* (Bauer et al., 2001), *MaSE* (DeLoach et al., 2001), *PASSI* (Cossentino, 2005), *Gaia* (Zambonelli et al., 2003), *RETSINA* (Sycara et al., 2003), *ROADMAP* (Juan et al., 2002), *Tropos* (Bresciani et al., 2004) and several others.

The Unified Modelling Language (*UML*) (Object Management Group, 2007) is used widely for object-oriented software engineering. Agent *UML* (*AUML*) contains extensions to *UML* to accommodate the requirements of agents.

The *Gaia* methodology (Zambonelli et al., 2003) is a methodology for agent-oriented analysis and design. *Gaia* has probably been the most popular methodology for the analysis of a system as a society/organisation; it consists of a set of roles that are later assigned to agents. It uses the following abstractions that characterise a computational organisation: the environment in which the *MAS* is immersed; the roles to be played by the different agents in the organisation; and the interaction between the roles. It also considers organisational rules and organisational structures.

The *ROADMAP* (Juan et al., 2002) (Role Oriented Analysis and Design for Multi-Agent Programming) methodology is an extension of *Gaia*. It includes formal models of knowledge and the environment, role hierarchies, explicit representation of social structures and relationships based on *AUML* interaction diagrams, while also incorporating dynamic changes.

The Multi-agent Systems Engineering (*MaSE*) (DeLoach et al., 2001) methodology contains all the steps the designer has to follow from the initial set of system specifications up to the implemented agent system. The *agentTool* system, which is a graphics-based interactive tool, was designed to support the *MaSE* methodology.

RETSINA (Reusable Task Structure Based Intelligent Network Agents) (Sycara et al., 2003) is a general-purpose modelling framework which proposes goal, role, context and attitude as first class objects for modelling multi-agent systems in an open world. *PASSI* (Cossentino, 2005) (a Process for Agent Societies Specification and Implementation) is a step-by-step methodology for designing and developing multi-agent societies.

The *Tropos* methodology (Bresciani et al., 2004) is based on the notions of agents and their goals, and plans are used in all phases of software development, from the early analysis to the actual implementation. *Tropos* also covers the very early phases of requirements analysis.

The system described in this study is a *MAS* for the prescription of Antiretroviral and Anti-TB drugs. The *MaSE* methodology was chosen for the analysis and design of the system, and the *agentTool* system was used in the process.

C h a p t e r 3

3 RELATED WORK

Agents are used in a variety of application domains where complex, distributed and heterogeneous information systems are appropriate. They have been used for personal assistants, intelligent user interfaces and for managing electronic mail. Other domains for the use of agents and *MASs* include telecommunication, industrial, commercial, entertainment and medical applications.

3.1 User Agents or Personal Agents

User or personal agents are intelligent agents that take action on a person's behalf. Intelligent agents can check a user's e-mail, sort it according to his priority, and alert him when important messages come through. They can assemble customised news reports, find information of the user's choice, automatically fill in forms and store the user's information for future reference, and scan Web pages by looking for and highlighting text in which the person is interested, etcetera. The website www.user-agents.org (List of User-Agents, 2007) contains an extensive list of user agents under the categories spiders, robots, crawlers and browsers. Agents can also be used in transport logistics for market-based transport scheduling and coordination, or as personal travel assistants.

3.2 Industrial Applications

Process control is a natural application for intelligent agents and multi-agent systems, as controllers are autonomous reactive systems. Agents are used in manufacturing, where the manufacturing process is modelled as a number of work cells that are functioning together. Parunak (Parunak, 1987) describes the YAMS (Yet another Manufacturing System) that adopts a multi-agent approach, by means of which each factory and factory component is represented as an agent. Ljungberg and Lucas describe a sophisticated air traffic control system known as OASIS (Ljungberg et al., 1992), in which agents are used to represent both aircraft and the various air traffic control systems in operation. Agents can also be used in transport logistics for market-based transport scheduling and coordination, or as personal travel assistants.

3.3 Commercial Applications

3.3.1 Information Management

The amount of information available to us in our everyday lives has grown immensely over the last couple of decades, which has resulted in information overload. The Internet and World Wide Web is a good example of this problem. Agents can be used to manage information effectively in two ways, that is, information gathering, to find information that meets our requirements; and information filtering, to sort out the relevant and important information. Maes (Maes, 1994) describes an electronic mail filtering agent called Maxims as well as an Internet news filtering programme called Newt that has the ability to learn from examples. An agent-based digital library, Zuno Digital Library (Ferguson et al., 1997), was developed using consumer agents to represent the user's interests. These agents maintain models of users and use these to provide the information they require. The system acts both as an information filter and information gatherer.

3.3.2 Electronic Commerce

Owing to the growth of the Internet and World Wide Web, online commerce has progressed substantially. Currently many e-commerce applications are still driven by human interactions, and information about traders, products and services are still collected and interpreted by humans. These activities can be very time consuming, and could be automated by utilising agents. Buying and selling agents can interact, bargain and make decisions regarding transactions. Chaves and Maes (Chavez A. & P. Maes, 1996) describe a simple electronic market place called Kasbah, which creates buying and selling agents for each transaction. Each commercial transaction takes place by the interaction of these agents.

Intelligent buyer agents (shopping bots) have been developed and are used widely by consumers to search for product and pricing information on the Web. Some systems are closed, such as Shopping.Yahoo.com (Yahoo! Shopping, 2007) and shop.AOL.com (AOL Shopping Main – Online Shopping Made Easy, 2007). In these systems buyer agents direct users to retailers who become part of the closed system through subscription. In open systems the buyer agents travel around the Internet, retrieving information about goods and services.

The website http://ecommerce.hostip.info/pages/938/Shopping-Bots.html describes buyer agents that compare the prices of different online merchants to find the lowest prices for consumers. One of the most popular sites, mySimon.com (mySimon – Price Comparison shopping, 2007) gathers information of more than 2000 retailers, and uses Virtual Learning Agent software. Users can search for a specific product by keyword, or browse through categories. When shoppers reach a decision, they are routed to the website selling the item.

Another important buyer agent is PriceSCAN, which also includes offers from merchants without websites. The BookFinder.com agent (BookFinder.com: Search for New & Used Books, 2007) searches Amazon, Antiqbook, Barnes and Noble, Bibliofind and other online book retailers. It uses a combined database of roughly 15 million books, to find the cheapest book prices for shoppers.

3.3.3 Business Process Management

Managers have to gather information from different sources to make informed decisions. The gathering of relevant, consistent and current information is a complex and time consuming process. A number of IT systems have been developed to assist in business process management. A system named *ADEPT* (Advanced Decision Environment for Process Tasks) (Jennings et al., 1996) employs a number of negotiating agents that each provides one or more services. Each agent plays a distinct role, and when an agent requires a service from another agent, they negotiate to obtain a mutually acceptable agreement. The proactive nature of the agents makes it possible to have just-in-time scheduling of services.

3.4 Entertainment

Computer games, interactive theatre and virtual reality applications used in entertainment are full of autonomous, animated characters, which can be implemented as agents.

Computer games are increasing in complexity and reality. The Sims (The Sims – Official Site, 2006) is a well-known system consisting of different agents displaying varied behaviours within an interesting environment.

3.5 Medical Applications

Medical informatics is an important growth area in computer science, and new applications for computers in the health industry are being found every day.

The health care domain at all levels (local, regional, national and international) is characterised by being a vast, open environment where health care professionals, such as doctors and nurses, with different skills and roles, manage patients via shared and distributed decision making. Such environments require the communication of complex and diverse forms of information between a variety of clinical and other settings, as well as the coordination between groups of health care professionals with very different skills and roles (Nealon & Moreno, 2004).

Health care professionals in such environments, in particular, require information that is both timely and error-free, to ensure that recommendations or decisions offered by the software systems are secure and trustworthy.

Because the above-mentioned characteristics match the customary properties of intelligent agents, the health care domain is well suited for the development of flexible, intelligent computer systems.

Intelligent agents are already being used in different fields within the health care domain. Below are some examples of application areas in the health care domain where agents have been deployed.

3.5.1 Distributed Patient Scheduling within a Hospital

Medical procedures have become very complex and include tests and treatments that are interconnected. The different tasks to be performed on a hospitalised patient have to be executed in a specific chronological order, which also include medical restrictions among the procedures. It is very difficult to schedule these actions manually or to use traditional software solutions. This can be improved considerably with the use of multi-agent systems (Decker et al., 1998), (Kumar et al., 1989), (Marinagi et al., 2000). An agent framework for building cooperative software agents in medical applications was developed by Lanzola et al., (Lanzola et al., 1999) in which they proposed a generic computational model that may be specialised to support the different requirements of a Hospital Information System.

3.5.2 Organ and Tissue Transplant Management

In this field the matching of available organs with the list of waiting patients to find the most appropriate recipient in a hospital or across a certain region requires optimal efficiency. When an organ becomes available, it is important that the most suitable patient must receive the organ as soon as possible. The time used to do this can be improved significantly with multi-agent systems. A *MAS* has been designed and implemented to coordinate the management of organ transplant in Spanish hospitals (Moreno, 2003). It consists of agents that have different knowledge and play different roles, which coordinate their activities to deal with organ transplant management at a national level. Vázquez-Salceda et al. (2003) developed an agent-based system called *Carrel* that assists in managing the allocation of tissues stored in hospital tissue banks.

3.5.3 Community Care

Intelligent agents can be used to continuously monitor physiological signs in high-risk citizens, such as the elderly and people with chronic diseases. This can replace occasional reviews by health care staff. More complete information on the medical condition of patients will enable both patients and health professionals to understand better how to manage their activities in order to avoid or anticipate problems, instead of responding to extremely costly health crises. In the *INCA* (Intelligent Community Alarm) elderly care management system (Beer et al., 2002), an agent is associated with each elderly patient. This agent receives medical data, gives reminders to the person and alerts the medical centre if something goes wrong. An architecture for an inexpensive support system based on the Internet and using stationary as well as mobile agents was developed by Camarinha-Matos and Vieira (1999). Rialle et al. (2003) propose an intelligent *MAS* for telemonitoring patients at home. They recommend a system using in-home bio-signal sensors connected to a local area network, remote server and the carers' computers, as well as software agents with different levels of knowledge on the different devices.

Another *MAS* described by Moreno (2003) is the *PalliaSys* project. This system is used to improve the management of patients in a Palliative Care Unit by telemonitoring them continuously. This unit specialises in dealing with people with terminal illnesses, and its aim is to ease their pain in the final phase of their lives.

3.5.4 Information Access

Communication of complex and varied forms of information between a variety of settings, as well as the coordination between groups of health care professionals with very different skills and roles, is necessary. Information agents have been developed to collect and organise the vast amount of medical information available on the Internet. These intelligent and specialised search engines are used to make searches more efficient and usable.

The Health on the Net Foundation and the Molecular Imaging and Bioinformatics Laboratory at Geneva Hospital have developed Multi-Agent Retrieval Vagabond on Information Networks (*MARVIN*) (Boyer et al., 1997), (Baujard et al., 1998), which searches sites and documents related to a specialised scenarion. *MARVIN* has already been used in the medical field to index documents according to their relevance to the health and medical field. This specific information is then used by a medical search engine, *MedHunt*. One field in which *MARVIN* has been implemented is in managing the UK National Electronic Library for Communicable Diseases (Kostkova et al., 2002).

A multi-agent system has also been implemented within the AgentCities.NET European project to build a worldwide network of agent-based platforms that provide interesting services to citizens and visitors to a city. This has been applied in the medical field by providing patients with information about medical centres satisfying certain properties, access to their medical records and doctors available in a certain centre, on a certain day, to make an appointment. While examining a patient, the doctor also accesses and updates the patient's medical record (Moreno, 2002).

Lieberman and co-workers have developed a browsing assistant, *Letizia* (1995), which records a user's choices in a Web browser and compiles a profile of the user's interests. It "follows you around" on the Web and updates this profile dynamically. They also developed a software agent, *Aria*

(Lieberman et al., 2001) that detects opportunities for annotation and retrieval of images in the context of the user's everyday work. A combination of these two systems has been used in the management of health care information.

3.5.5 Decision Support Systems

Intelligent agents can be used to continuously monitor physiological signs in high-risk patients such as diabetics or heart patients. Comprehensive information on the condition of patients makes it easier to notice a trend and to predict potential problems. A *MAS* can notify a patient or healthcare professional that certain action has to be taken to prevent a potential crisis, which could be very costly, or even fatal (Barro et al., 1999). The tasks of monitoring and diagnosing intensive-care patients require knowledge and skill, and demand correct action in complex, unexpected, and time-critical situations. A system called *Guardian* (Hayes-Roth et al., 1992), developed at Stanford University's Knowledge Systems Laboratory, is an example of such a knowledge-based system designed to perform these tasks for post-cardiac surgery patients. An intelligent interface agent with medicine-related common sense reasoning was proposed to ease the difficult task of gathering useful patient data and making the correct diagnosis (Hsu et al., 1999). Another system called *Patient Advocate* assists maternity patients who are at risk of gestational diabetes to monitor their own behaviour and physiological condition (Miksch et al., 1996).

Susan L Mabry et al. (2003) describe a system that utilises intelligent agents to assist in patient health care. In their system multiple intelligent monitor agents coordinate as a team, with each agent performing specialised tasks of monitoring, performing diagnostics and establishing the appropriate intervention needed. The design includes a single *Patient Agent* that contains a GUI and a *Data Collection Unit* that gathers data about the patient. Multiple *Monitor Agents* are created, each narrowly focusing on a specific monitoring goal. Each monitor agent contains a *Dynamic Decision Module*, which comprises different modules for analysing and predicting a trend, and performing *symptomatic* and *systemic* diagnostics. Depending on these diagnostics, an agent may activate another agent. The agents are mobile in order to roam the network, collect data and process sensitive patient data at remote locations. A prototype implementation of their system helps with emergency trauma centre care of hemorrhagic shock and focuses on fluid resuscitation and arterial blood gas stabilisation.

A secure and distributed architecture for the monitoring of medical protocols was developed by Alsinet et al. (Alsinet et al., 2003). They define a system for the assistance and supervision of the real-time application of medical protocols in distributed hospital environments with computer-based medical records. They formalise the communication language of the agents and use the language to specify a real medical protocol for detecting and controlling hypertension. A patient, physician and nurse participate in this protocol. They use a graphical framework called *JAFDIS* (Java Framework for Dialogical Institution Specification) (Alsinet et al., 1998) to specify dialogical institutions.

3.5.6 Training

Agents can be used in distance-learning tutoring systems to improve medical training and education.

An agent-based intelligent tutoring system, *Ines* (Intelligent Nursing Education Software), was developed by Hospers and et al. (2003). It provides a generic teaching environment that uses agents to support learning and is applied to nurse education.

AMPLIA, as described by Vicari et al. (2003), is a multi-agent intelligent learning environment designed to support training of diagnostic reasoning and modelling of domains with complex and uncertain knowledge, which focuses on education in the medical environment.

3.5.7 Drug Prescriptions

A multi-agent system for monitoring the prescription of restricted use antibiotics is proposed by Godo et al. (2003). They describe a system to assist the revision of medical prescriptions containing antibiotics of restricted use in the pharmacy department of the hospital. A patient agent is attached to each patient to check the medical aspects of the prescribed therapy. A pharmacy agent will analyse the information and suggest alternative antibiotic treatment. These agents work in collaboration with human agents within the hospital environment.

In this research, we have identified the prescription of anti-retroviral and anti-TB drugs, which are usually required by people infected with HIV/AIDS, as an area where a *MAS* system could be deployed.

Joint HIV/TB intervention is an item on the World Health Organisation list of key areas for HIV/AIDS programming in the health sector (World Health Organization, 2005). The recommended interventions provided in this list are considered essential and have proven highly effective in restricting the HIV epidemic in many locations. The extent of the TB and HIV/AIDS epidemics is of such nature that additional health care staff will be needed and existing staff will have to be trained or re-trained to ensure that adequate joint interventions take place.

Anti-TB and anti-retroviral drugs may cause numerous side effects. The side effects caused by the drugs used in the treatment of the two conditions are often similar and can increase when a combination of drugs is used. Most of these symptoms may also be caused by TB or AIDS itself, which complicates the situation even more.

Interactions between the different drugs also prohibit the use of certain combinations of drugs, or require a change of dose. All these factors should be considered when prescribing medication to a person co-infected with HIV and TB. Doing this manually is a complex and time consuming task..

The goal of our system is to assist health care staff in carrying out this task in an efficient way in order to save time, minimise decision errors, and increase the standard of health care.

C h a p t e r 4

4 DESIGN OF MedAgent

4.1 Introduction

The *MedAgent* system was designed to assist in the administering of prescriptions that combine antiretroviral and anti-TB medications. This combination is a complicated and time-consuming process, which can be modelled as a multi-agent system. A number of entities such as patients, pharmacies, nurses, counsellors, doctors and pharmacists are involved in the process.

We initially proposed the following agents as part of the system: a patient agent, a physician agent, a nurse agent, a pharmaceutical agent, a pharmacist agent, a medication database (MDB) agent and a medication agent. The full extent of the agent society emerged at the end of the design process as described in the remainder of this chapter. The initial generic system architecture is shown in Figure 4.1.

The patient agent is used to collect and contain the medical conditions and symptoms of a particular patient, which is unique for each person and changes during treatment. The information can be obtained from existing medical records or supplied by health care professionals according to their observations. All access to the medical data of a specific patient is handled by the patient agent associated with that patient.

A nurse agent is used to assist the nurse to follow the correct procedures in the treatment of a patient. It reminds the nurse of the tests that have to be performed at a certain point in time and it interacts with the medication database (MDB) agent to obtain the prescription which is generated by the medication agent in consultation with a doctor. The nurse provides information to the nurse agent about observations made during visits, which are communicated to the patient agent. It also communicates information to the medication database agent about medicine issued to a patient. The nurse agent also informs the lab agent when a blood sample for a given patient has been dispatched for testing. Later, after the blood testing is completed, the lab agent sends the results to the patient agent.

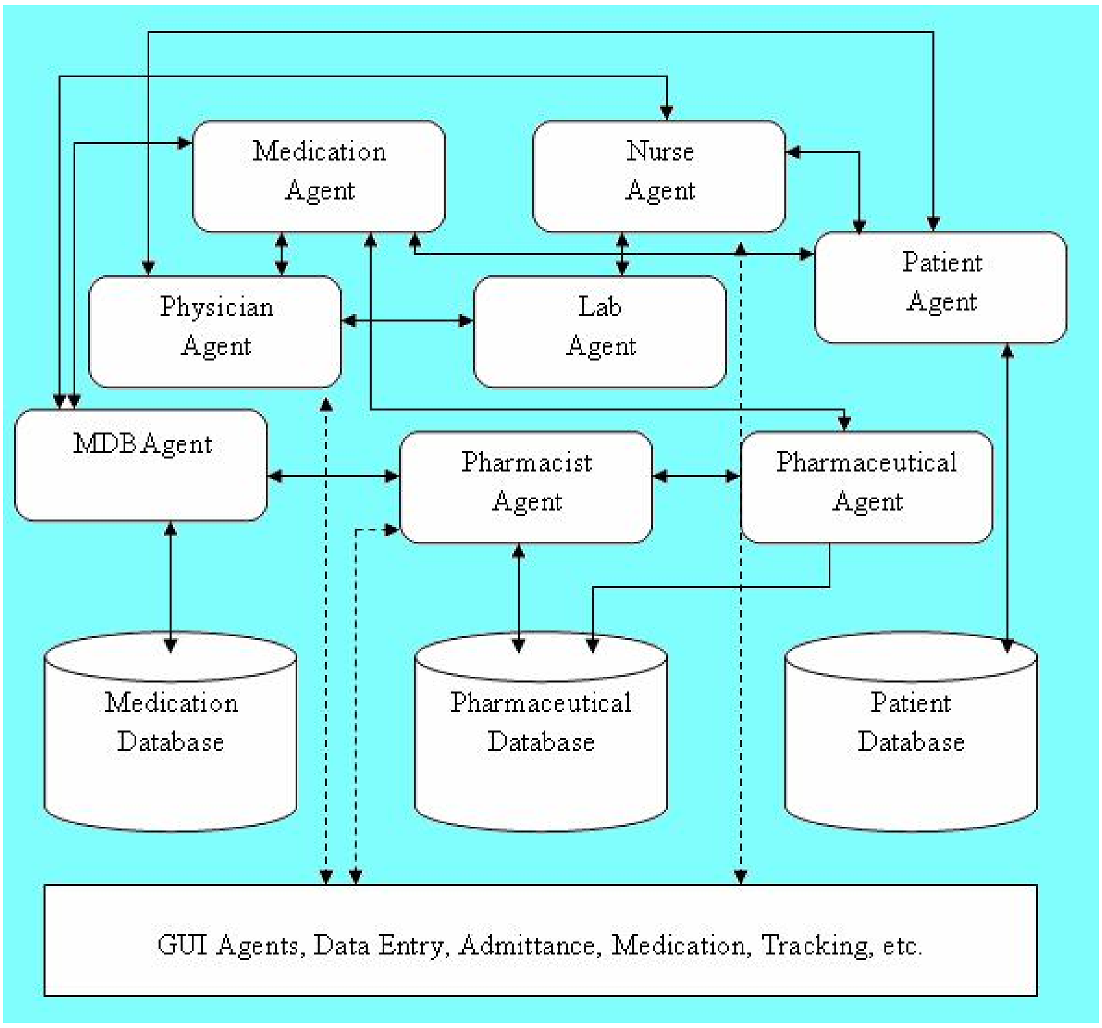


Figure 4.1: Overview of Generic System Architecture

The medication database agent (*MDBAgent*) mediates all access requests to the medication database by the medication agent, pharmacist agent, pharmaceutical agent, physician agent and nurse agent. The medication database contains information about prescriptions and drug issues for all patients, as well as information about available drugs, usage instructions, side effects and interactions.

The major task of the medication agent is to generate prescriptions for patients based on their medical conditions. However, a prescription must undergo certain validation steps before it can be saved on the prescription database. The doctor must be consulted, who will either accept the prescription or request alternatives. The medication agent communicates with the pharmaceutical agent as well as the patient agent to determine the medication regime for a specific patient. It obtains information about the effect of a specific drug, as well as the contra-indications. The pharmaceutical agent has to decide whether the substance is compatible with the other medications given to the patient. The pharmaceutical agent does this by consulting the pharmaceutical database and the pharmacist agent, if necessary.

The medication agent communicates with the patient agent to find out if any condition in the patient prohibits the use of that medication, or warns the health care professional that caution has to be taken. If no contra-indications exist, the medication agent adds the medication to the list of prescribed medications. The completed combination of medications are forwarded to the physician to confirm and finalise. The prescription is then saved to the medication database, where it can be accessed by the pharmacist, physician and the nurse.

The pharmacist agent is responsible for maintaining the pharmaceutical database and has access to patients' prescriptions via the *MDBAgent*. When the patient visits the pharmacist, the pharmacist agent will access the medication database to retrieve the prescription so that the medicines can be issued. The prescription stored in the medication database is also updated to reflect the fact that the medicines have been issued.

In the system, each doctor is represented by a physician agent. The physician agent communicates with the patient agent to obtain data about a given patient and to update the patient data. It communicates with the medication agent to request a prescription, which must be confirmed by the doctor via the physician agent. It also communicates with the *MDBAgent* to obtain patient prescriptions, as well as the pharmacist agent, whenever there is a need for consultations with regard to medicines.

The initial generic system architecture, which is incomplete at this stage, is shown in Figure 4.1. In Section 4.2, the *MaSE* methodology is described. In the following sections the application of *MaSE* to the analysis, design and deployment of *MedAgent* is elaborated on.

4.2 The MaSE Methodology

The Multi-agent Systems Engineering (*MaSE*) methodology (DeLoach et al, 2001) was chosen for the analysis and design of the system, because *MaSE* is a complete methodology for developing multi-agent systems and it leads the designer from the initial system specification to the implemented agent system. It produces a set of formal design documents in graphically based styles, some of which are in *UML*. *MaSE* is independent of a particular multiagent system architecture, agent architecture, programming language or message-passing system, since a system designed in *MaSE* can be implemented in several ways from the same design. It is also possible to track changes to the process, and every design object can be traced backward and forward through the different phases of the methodology.

The *MaSE* methodology consists of two main phases namely analysis and design, as depicted in Figure 4.2 below (DeLoach, 2001). When using *MaSE*, iteration can be applied in both phases.

4.2.1 Analysis

The analysis phase consists of three steps: capturing goals, applying use cases and refining roles. The first step, capturing goals, takes the initial user requirements, which can be technical documents, other documents or verbal instructions, and turns them into a structured set of top-level system goals shown in the form of a *Goal Hierarchy Diagram (GHD)*. These goals are analysed and structured into a form that can be passed on and used in the design phase.

In the second step these system level goals are taken and *Use Cases* are extracted. A use case is a description of a sequence of events that defines a certain behaviour of the system. It determines the minimum set of messages that must be passed between roles. If a message is passed between two roles, there must be a corresponding communication path between them, and this means that a conversation must be constructed between the agents that play those roles. The possible scenarios that are created from sequence diagrams are represented in the form of *Sequence Charts*. This step defines an initial set of system roles and communications paths.

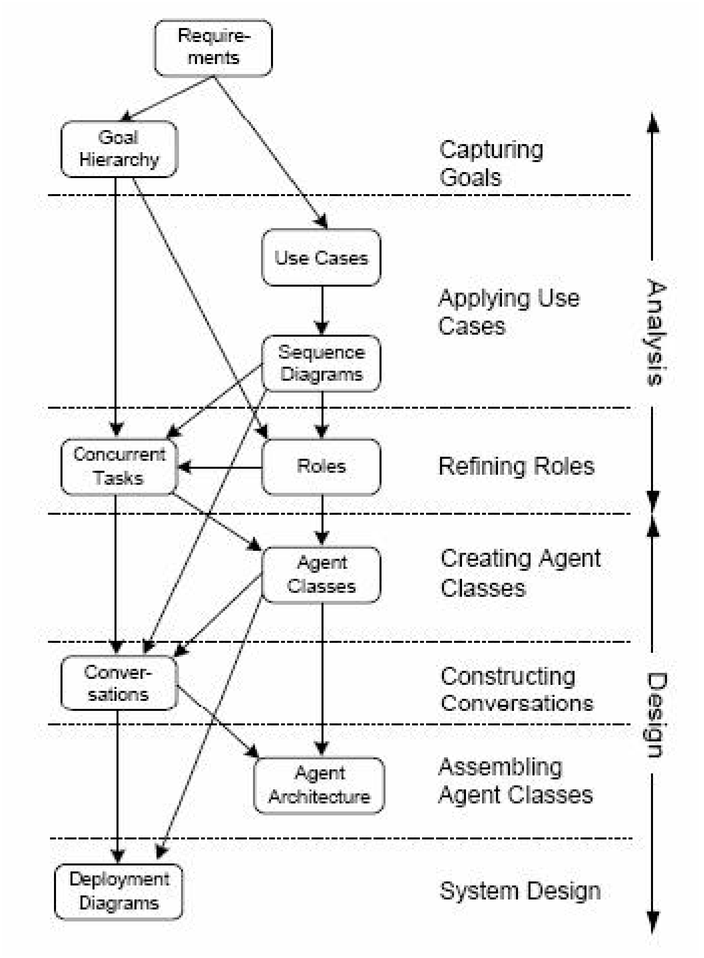


Figure 4.2: The *MaSE* Methodology

The third step is to refine the structured goals of the *GHD* into roles, where a role is defined as a description of an agent's expected function. In general, each goal specified in the first step maps to a specific role, but goals that are similar or related may be combined into a single role. The roles are captured in a *Role Model*, which includes communication paths between roles, derived from the sequence diagrams in the previous step.

After roles are created, tasks are associated with each role and every goal associated with a role can have a task that details how the goal is accomplished. A task is a structured set of communications and activities, depicted as a *State Diagram*.

4.2.2 Design

In the design phase, we transform the analysis models into constructs useful for actual implementation in the multi-agent system. The design phase has four steps: creating agent classes, constructing conversations, assembling agent classes and system design. In the first step of the design phase, creating agent classes, specific agent classes are defined to fill the roles defined in the analysis phase. The product of this phase is an *Agent Class Diagram*, where agents consist of two components: roles and conversations.

After determining the number and types of agent classes in the system, we can construct conversations between those agent classes in the 'Constructing Conversations' step and define the internal components that comprise the agent classes in the 'Assembling Agent Classes' step. These two steps are closely linked and may be done in parallel. A conversation in *MaSE* defines a coordination protocol between two agents, and consists of two *Communication Class Diagrams*, one each for the initiator and the responder. The internals of agent classes are created in the 'Assembling Agent Classes' step.

The final phase of the *MaSE* methodology uses a *Deployment Diagram* to instantiate the agent classes as actual agents. It shows the number of individual agents, their locations and other system specific items.

4.3 The Use of agentTool

MaSE is supported by a software engineering tool called *agentTool* (DeLoach et al., 2001), which simplifies the development process. It is a Java-based, graphical development environment in which the system designer defines high-level system behaviour graphically by using the *MaSE* methodology.

In the *MaSE* analysis phase, *agentTool* is used to create a goal hierarchy diagram. In the next step use cases are added and a sequence diagram is defined for each use case. The third step creates a role diagram. It implements the roles by assigning all goals in the hierarchy diagram to a specific role.

The first diagram created in the design phase is the *Agent Template Diagram*, in which agents and their conversations are declared. In the next step, the agent architecture of each agent in this diagram is then completed by defining the components of the agent and the connections between the components. Properties, attributes and methods are defined for each component. Conversations are defined to accomplish communication between agents.

Finally all instances of agents can be added to the deployment diagram to define a working system.

4.4 Development of MedAgent System

4.4.1 Analysis

4.4.1.1 Capturing Goals

The main source of information used in the analysis of our system, is the PALSA Plus (Practical Approach to Lung Health South Africa Plus HIV) guidelines for treating patients with TB and HIV. These guidelines are used by most clinics and hospitals in South Africa. The forms that are completed by health care professionals at the different visits of the patient to the clinics and hospital, are another source of information.

As discussed in chapter 1, the first diagnosis of HIV is usually done at a primary health care clinic. If the patient is found to be HIV positive, he is sent to an HIV clinic for further tests, to establish if he qualifies for ARV treatment. If the patient qualifies for ARV treatment, the patient visits a doctor, who will do a baseline assessment. Before starting ARV treatment, the patient has to complete a drug readiness training programme for three weeks. The doctor then prescribes the medicine and the prescription is given to the primary health care clinic, which will provide the medicine. The prescription is sent through to the pharmacist, who will verify that the combination and dose of medicines are correct. The medicine is dispensed and sent to the clinic, where the patient will receive it.

The patient has to visit the doctor and clinic at regular intervals. At each visit the doctor or nurse will do some tests, and the doctor will assess the condition of the patient and revise the prescription if necessary. All TB treatment is done at TB clinics and combination pills are used. At this stage TB clinics and HIV clinics work in parallel, without interaction.

These requirements were used to identify the system goals, as included in a goal hierarchy diagram as shown in Figure 4.3. An iterative process was applied to break down the main goals into sub goals.

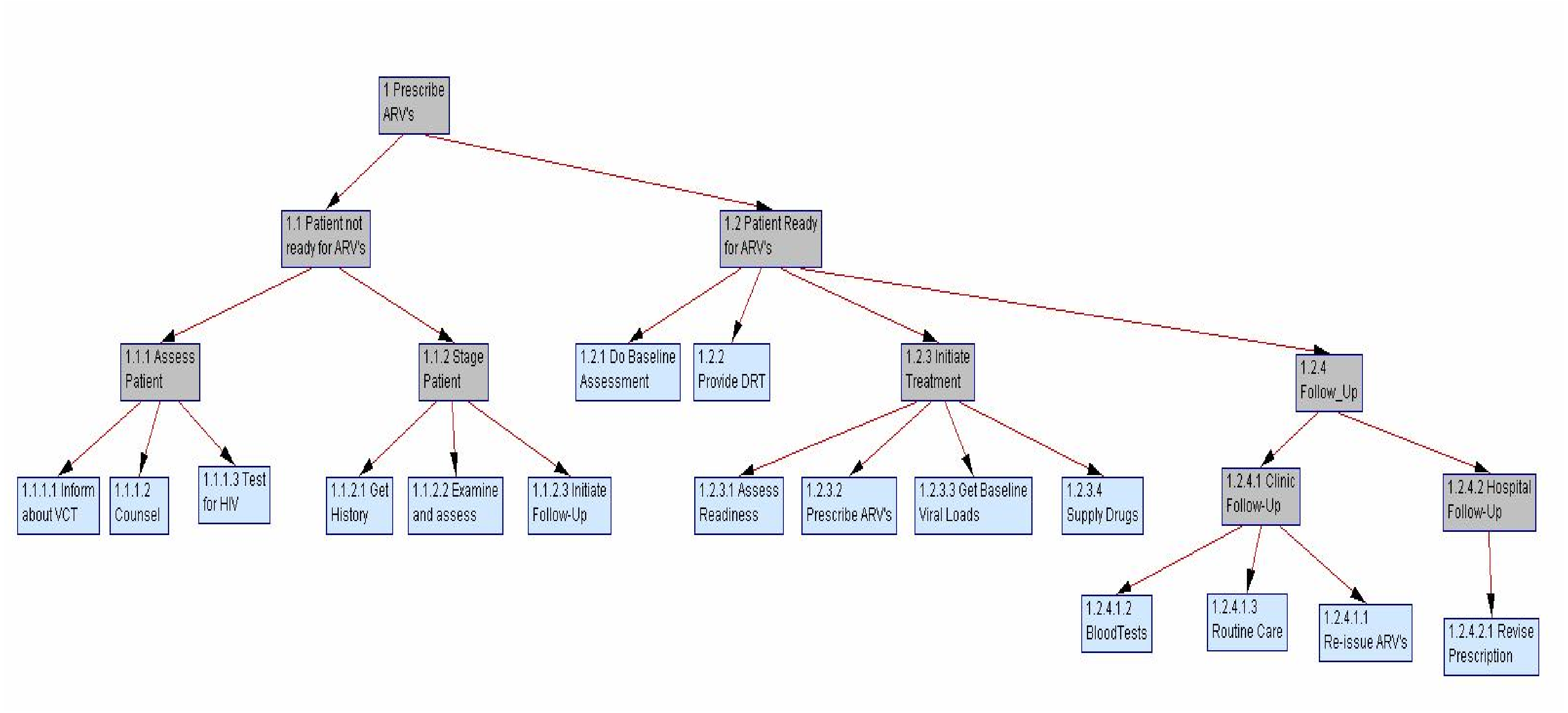


Figure 4.3: Goal Hierarchy Diagram

4.4.1.2 Applying Use Cases

In the Applying Use Cases phase, the following use cases of the system were identified:

- Inform About VCT
- Counsel and Test for HIV
- Get History
- Examine and Assess
- Baseline Assessment
- Drug Readiness Training
- Assess Readiness
- Prescribe ARVs
- Baseline Viral Loads
- Supply Drugs
- Clinic Follow-Up
- Hospital Follow-Up

Each use case was described in detail using sequence diagrams.

Figure 4.4 shows the sequence diagram for the *prescribeARVs* use case. The use cases of the system include: *InformAboutVCT, CounselTestHIV, GetHistory, ExamineAssess, DrugReadinessTraining, AssessReadiness, PrescribeARVs, BaseLineTests, SupplyDrugs, ClinicFollowup, HospitalFollowup*, and so on. The *prescribeARVs* use case is one the most complex use cases in the system and is described below to illustrate the drugs prescription process. Descriptions of the relatively complex *SupplyDrugs* and *HospitalFollowup* use cases are also given below. The sequence diagrams for all the other use cases are included in Appendix A, but their descriptions will be omitted owing to space limitations.

The prescription process is done when a patient visits a doctor. The first request is submitted by the physician agent to the patient agent. The patient agent then returns the data associated with the medical conditions of this patient. This enables the doctor to make informed judgements when he/she examines the patient. Upon examination, the doctor requests a prescription for the patient based on the latest patient information. The physician agent sends the request to the medication agent to generate a prescription for this patient. Together with this request, the physician agent forwards the data that is required in order to generate the correct prescription.

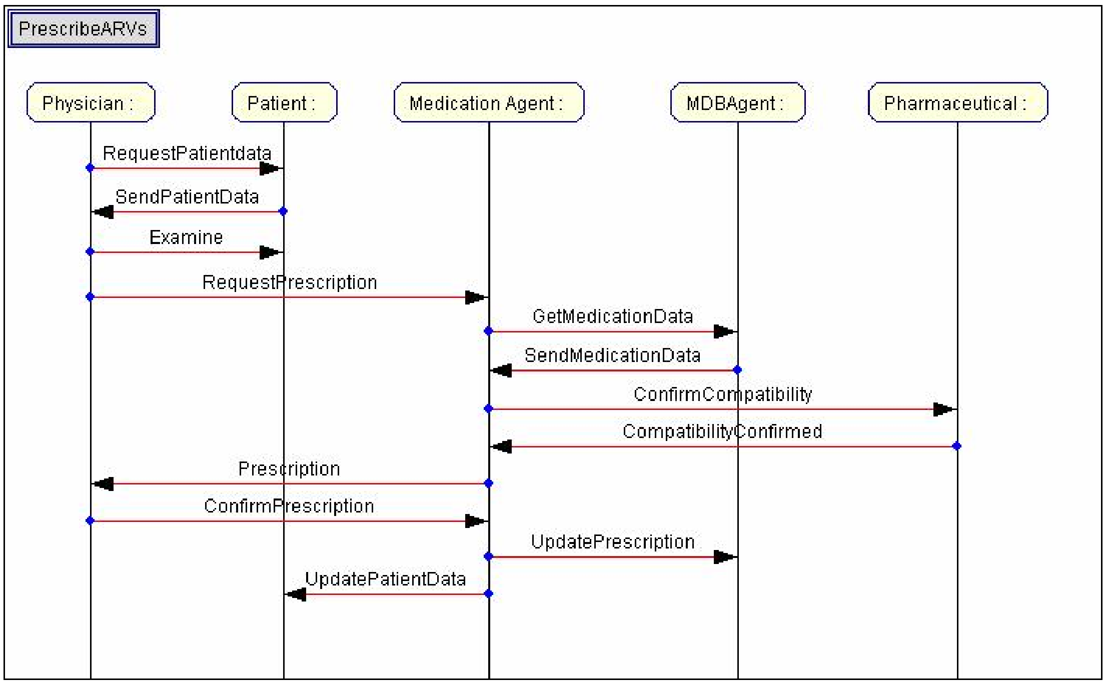


Figure 4.4: Sequence diagram for the *prescribeARVs* use case

The medication agent receives the request, and it then formulates possible prescription options. Before it can select a specific option, it must first confirm a few matters. The medication agent requests the *MDBAgent* to send the medication data so that it can decide if there are any side effects and contra-indications for the drug options it has formulated based on the patient's medical conditions. It also requests the pharmaceutical agent to confirm that the drug formulations are compatible with any other medications that the patient is currently taking. The pharmaceutical agent does this by consulting the pharmaceutical database and, if necessary, the pharmacist agent. For simplification this part of the prescription process is not shown in the *prescribeARVs* use case.

Once the compatibility between drugs, side effects and contra-indications have been confirmed, the medication agent decides on valid prescription options and forwards a valid option to the physician agent. The physician agent will then confirm the drugs prescription through the doctor. If the prescription is accepted, the medication agent requests the *MDBAgent* to update the prescriptions database. By then, the physician agent will have requested the patient agent to update the patient records. The prescription process is now considered to be complete. If the physician agent had rejected the prescription in the relevant step above, the medication agent would have sent an alternative prescription, and this would have continued until confirmation was obtained. The medication agent will only ask the *MDBAgent* to update the prescription database after confirmation has been received. During these conversations, the physician agent may consult the pharmaceutical agent if necessary.

The *SupplyDrugs* use case is shown in Figure 4.5 and its description follows.

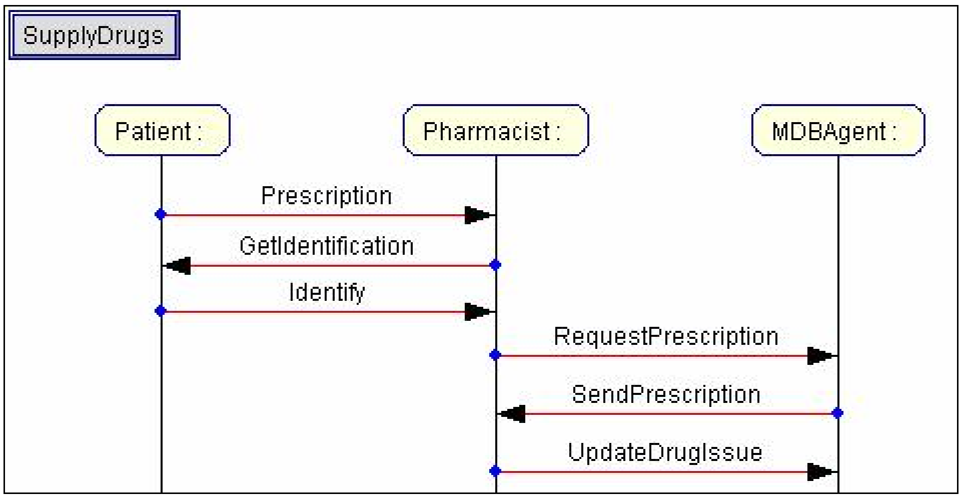


Figure 4.5: Sequence diagram for the *SupplyDrugs* use case

After a prescription has been finalised, the patient collects the medication from the pharmacist. The pharmacist retrieves identification detail of the patient via communication between the pharmacist agent and the patient agent. After identification has been established, the pharmacist agent requests the prescription detail from the *MDBAgent*. The *MDBAgent* replies by sending the prescription information to the pharmacist agent. The pharmacist then issues the drugs to the patient, and the pharmacist agent sends the information about the drugs issued to the *MDBAgent*. The *MDBAgent* then updates the number of issues completed in the prescriptions and issues database.

The *HospitalFollowup* use case is shown in Figure 4.6 and its description follows.

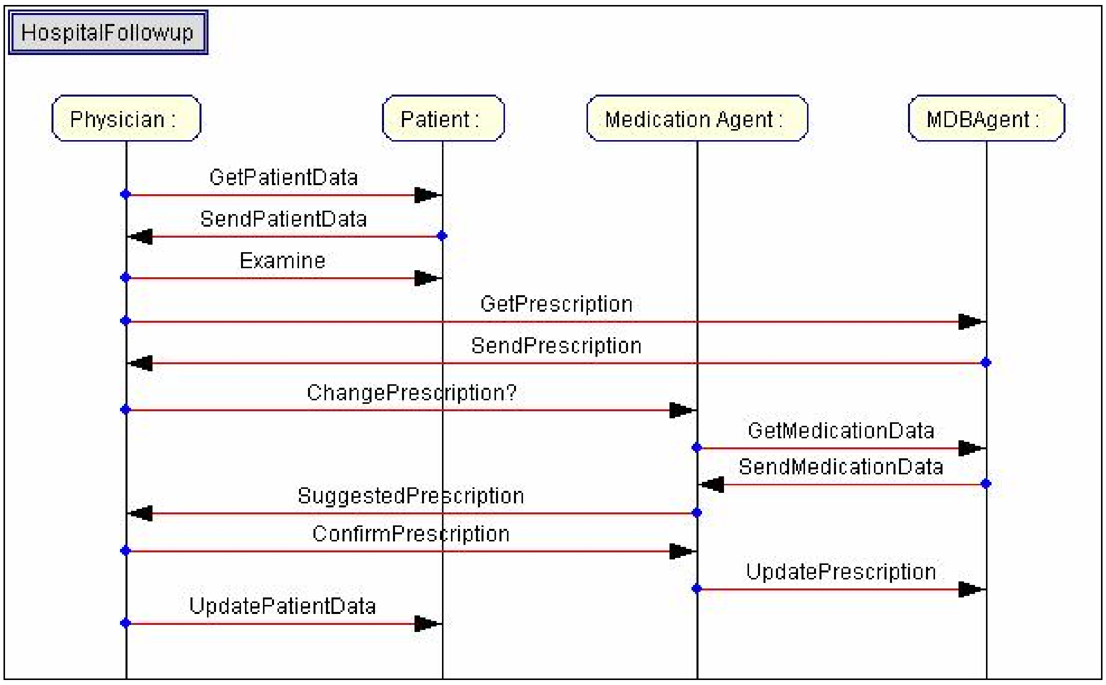


Figure 4.6: Sequence diagram for the *HospitalFollowup* use case

When the patient visits a doctor at the hospital for a follow-up visit, the first step of the process is a request from the physician agent to the patient agent to retrieve the personal and medical data of the patient. The patient agent replies by sending the data of this patient to the physician agent. The doctor will then do a physical examination of the patient and use the data obtained from the patient agent to compare the condition of the patient at a previous visit with the condition at present. The next step is to see if the prescription has to be updated. The physician agent sends a request to the *MDBAgent* for the current prescription, and the *MDBAgent* replies to the physician agent with the information. The physician consults with the medication agent to determine if the prescription has to be changed. The medication agent requests the medication data from the *MDBAgent* and the *MDBAgent* replies with the relevant data. The medication agent formulates a prescription in consultation with the pharmaceutical agent and sends it to the physician agent, who will confirm the prescription like in the *PrescribeARVs* use case. When the prescription is finalised, the medication agent asks the *MDBAgent* to update the prescription on the medication database. The physician finally sends the data obtained during the examination to the patient agent to update the patient database.

Nine roles for the Role Diagram were identified: Patient, Physician, Nurse, Counsellor, Medication, Medication Database (*MDBAgent*), Laboratory, Pharmaceutical and Pharmacist Agent. Tasks were assigned to the different roles to satisfy each goal. The counsellor role was not part of the original design, but was included to fulfil the task of counselling the patient after he has been diagnosed as HIV positive. This task could also be performed by the nurse, but it would be more logical to keep this as a separate role. The complete Role Diagram consists of the nine roles, their task allocations and external as well as internal communications among the agent roles. The role diagram gets cluttered very quickly when all these details are included. Figure 4.7 shows part of the role diagram consisting of the Physician, Patient, Medication and *MDBAgent* agent roles and the communications among them.

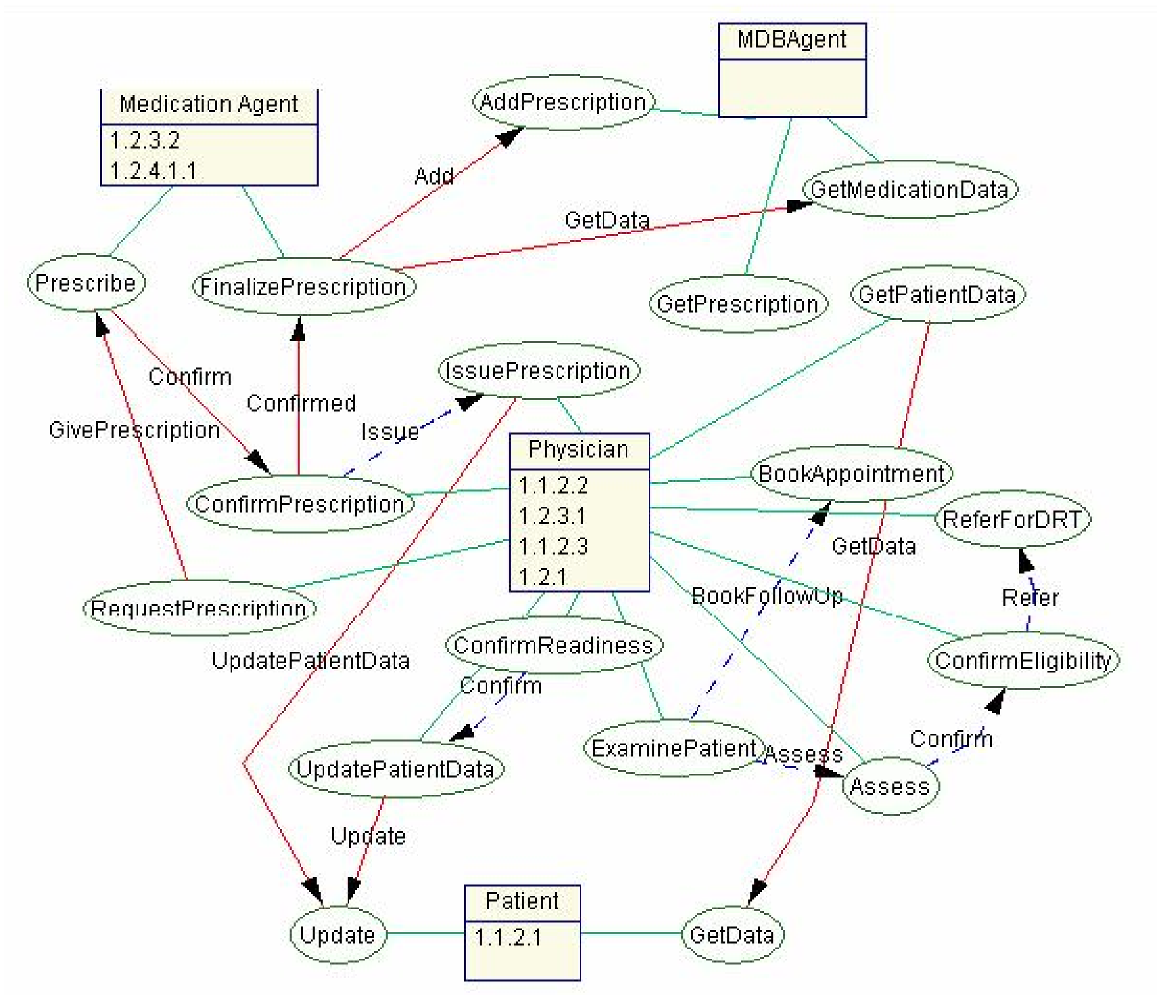


Figure 4.7: Role diagram for Physician, Patient, Medication and *MDBAgent* roles

4.4.2 Design Phase

In the design stage agent classes and their conversations were created, and then assembled to create the agent architecture. As a final step a deployment diagram was used to show the system structure.

4.4.2.1 Creating Agent Classes

Each role is represented by an agent, which manifests itself as a class or classes. A class may be associated with more than one role, but each role must be represented by at least one agent class to ensure that all system goals that were identified are captured in the design. If an agent class plays more than one role, the roles being played change dynamically at execution time. In addition, agents of the same class may play different roles simultaneously. The association between role and task is not always one-to-one, although it is the case in this design. It is important to note that roles are the foundation upon which agent classes are designed.

Roles correspond to the set of system goals defined in the analysis phase and form a bridge between what the system is trying to achieve (the analysis phase and goals) and how it goes about achieving it (the design phase of agent classes). The analyst can easily change the organisation and allocation of roles among agent classes during design, since roles can be manipulated in a modular way.

The agent class diagram depicts the agent classes and their conversations, similar to Object-Orientated (*OO*) Class diagrams. However, there are two main differences between the agent class diagram in this stage of *MaSE*, and *OO* class diagrams. The first difference is that agent classes are not defined by attributes and methods, but by the roles they play. Secondly, the semantics of the relationships between agent classes are different. All relationships in agent class diagrams are conversations that may take place between two agent classes.

Figure 4.8 shows an example agent class diagram in *MedAgent*. It contains eight of the nine agent classes, their associated roles and some of the conversations among them. The rectangles in Figure 4.8 depict agent classes and contain the class name and the role each agent plays (only one in this design). The lines with arrows show conversations and point from the initiator of the conversation to the responder. The name of the conversation is indicated next to the arrow. Not all conversations between the agents in Figure 4.8 are shown. Refer to Figure 4.7 for the possible conversations that can occur between the agents in Figure 4.8. All details of conversations between agents are elaborated on in the next step of the design process, that is, constructing conversations.

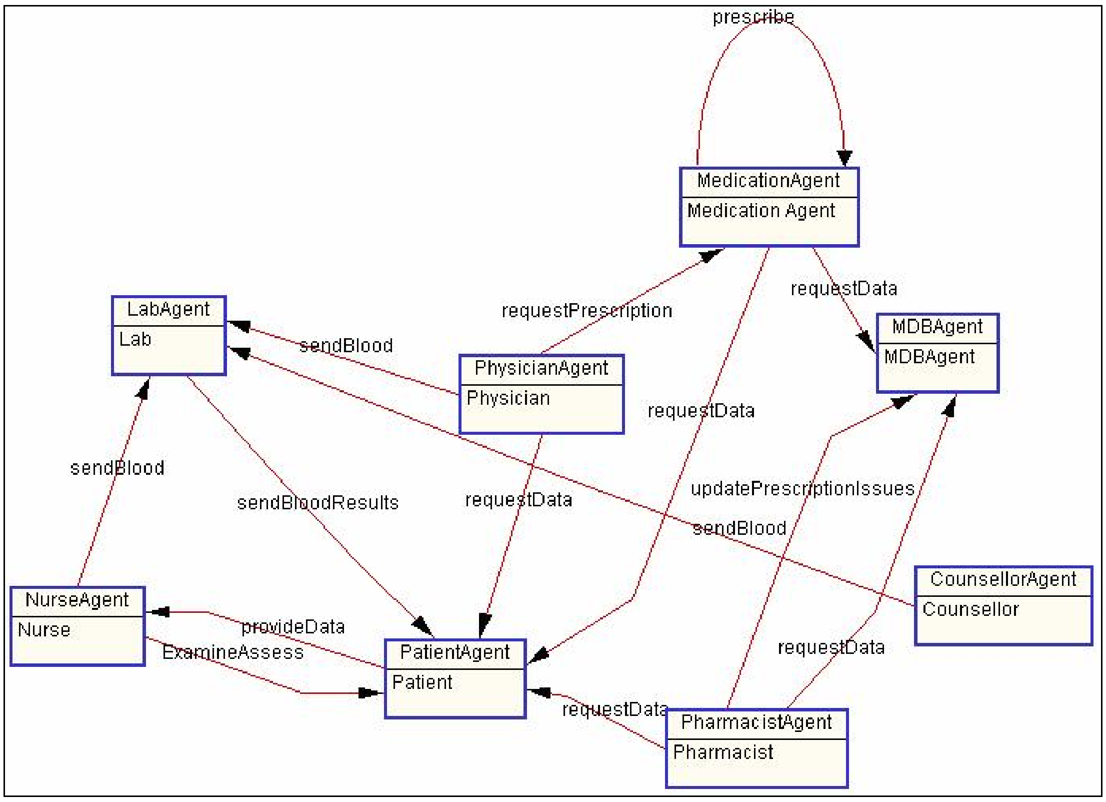


Figure 4.8: Agent Class Diagram

4.4.2.2 Constructing Conversations

Constructing Conversations is the next step in the *MaSE* design phase. Up to this point the designer may only have stated the conversations that exist between agents without defining the communications any further. In this step the details of conversations are defined and communication class diagrams are drawn for all conversations between agents. Each of these diagrams is a finite state machine that defines the conversation states of the two participant agent classes, that is, the initiator and the responder. The diagram drawn for the *RequestPrescription* conversation as shown from the perspective of the initiator, that is, the *PhysicianAgent*, is shown in Figure 4.9.

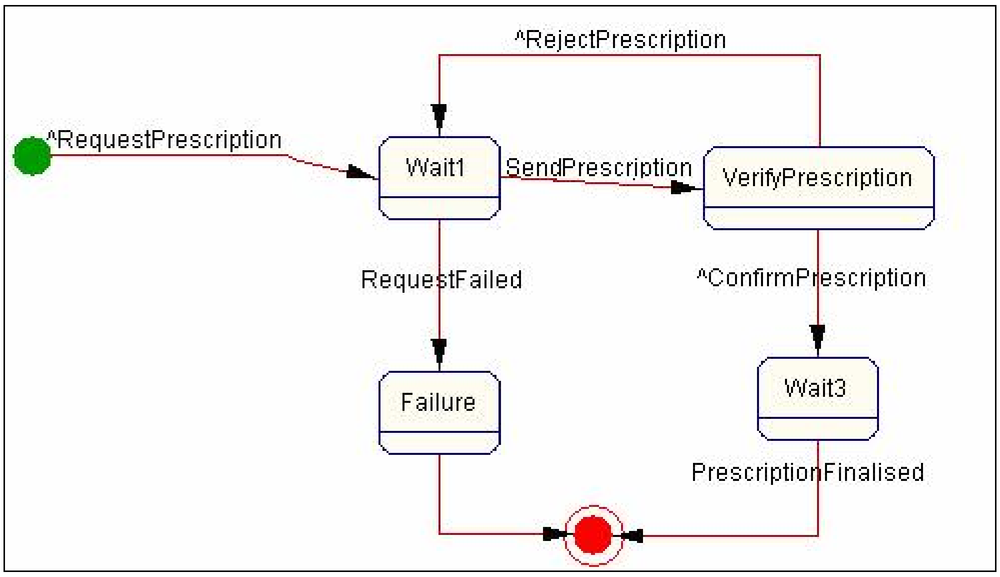


Figure 4.9: Communication Class Diagram from perspective of initiator

The *PhysicianAgent* begins the conversation by sending the first message, *RequestPrescription* to the *MedicationAgent*, and then it enters state *Wait1* until it receives message *GivePrescription* or message *RequestFailed.* If the *GivePrescription* message is received, it enters the state *VerifyPrescription* and after verification, it either sends message *RejectPrescription* and returns to state *Wait1* or it sends the message *ConfirmPrescription*, and enters stage *Wait3* until it receives the message *PrescriptionFinalised*, and the conversation is ended. If message *RequestFailed* is received, it enters the *Failure* state and the message is ended.

The same conversation as shown from the perspective of the responder, that is, the *MedicationAgent*, is shown in Figure 4.10. In this diagram the conversation is started when the responder, that is, the *MedicationAgent* receives the *RequestPrescription* message and enters the *Prescribe* state. Then it either sends the message *SendPrescription* and enters the state *Wait1*, or it sends the message *PrescriptionFailed* and enters the *Failure* state, after which the conversation is ended. In the *Wait1* state it will either receive the message *PrescriptionRejected* and return to the *Prescribe* state, or receive the message *PrescriptionConfirmed*, and send the message

PrescriptionConfirmed, after which it will enter the *Finalise* state, send the message *PrescriptionFinalised* and the conversation is completed.

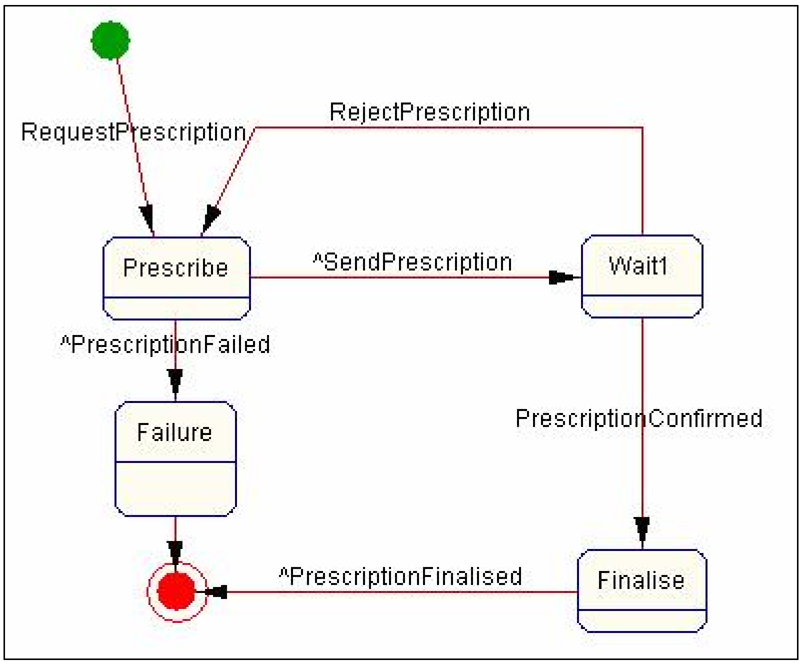


Figure 4.10: Communication Class Diagram from perspective of responder

4.4.2.3 Assembling Agents

During this step of the design phase the internal details of the agent classes are created. This is accomplished by defining the agent architecture and defining the components that make up the architecture. These architectural components consist of a set of attributes and methods. The components are represented by boxes in the architecture diagram, which are connected to inner- or outer-agent connectors. Inner-agent connectors are shown as thin arrows and define the visibility between components, whereas outer-agent connectors are shown as thick dashed arrows, which define connections with external resources such as other agents and databases.

The architecture and the internal definition of components must be consistent with the conversations defined in Section 4.4.2.2. At the very least, each action or activity defined in a communication class diagram (for example in

Figure 4.9), should be defined as an operation in one of the internal components.

The architectures of the *PhysicianAgent* and the *MedicationAgent* are shown in Figures 4.11 and 4.12 below. The actions depicted in Figures 4.9 and 4.10 manifest themselves as methods in the different components of the agent architecture. The message that initiates the conversation is shown as method *requestPrescription* in *Physician Examiner*, which is a component of the *PhysicianAgent*. In the same way all the other messages shown in the communication class diagrams of this conversation are also implemented as methods of the agents involved in the conversation (See Figures 4.9 to 4.12).

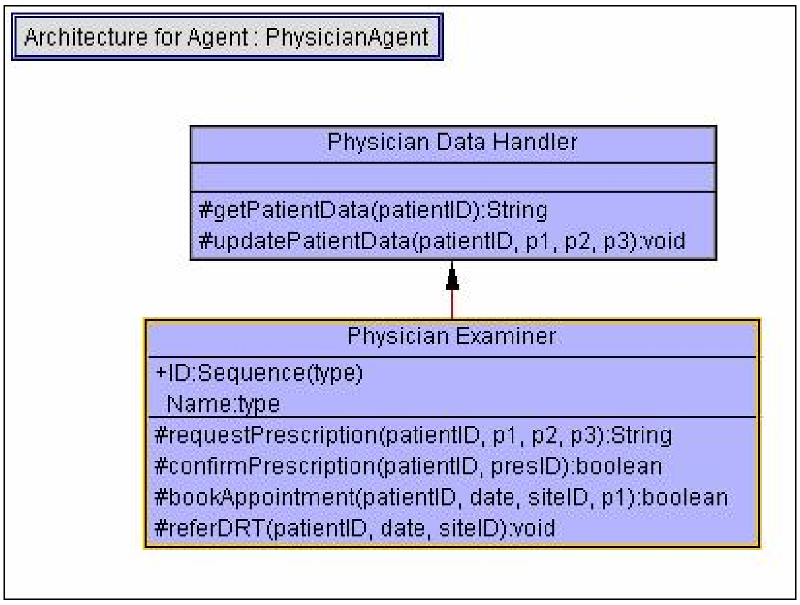

Architecture for Agent : PhysicianAgent
Physician Data Handler
#getPatientData(patientID):String
#updatePatientData(patientID, p1, p2, p3):void
Physician Examiner
+ID:Sequence(type)
Name:type
#requestPrescription(patientID, p1, p2, p3):String
#confirmPrescription(patientID, presID):boolean
#bookAppointment(patientID, date, siteID, p1):boolean
#referDRT(patientID, date, siteID):void

Figure 4.11: Architecture of *PhysicianAgent*

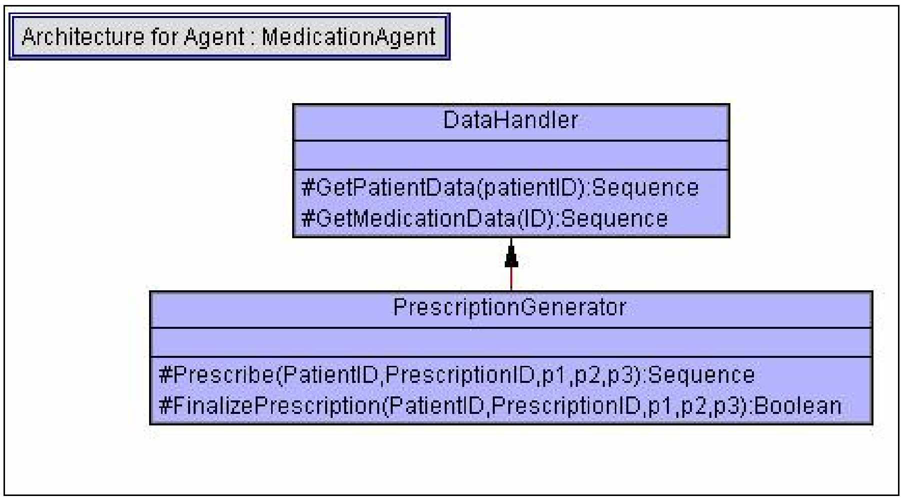

Architecture for Agent : MedicationAgent
DataHandler
#GetPatientData(patientID):Sequence
#GetMedicationData(ID):Sequence
PrescriptionGenerator
#Prescribe(PatientID,PrescriptionID,p1,p2,p3):Sequence
#FinalizePrescription(PatientID,PrescriptionID,p1,p2,p3):Boolean

Figure 4.12: Architecture of *MedicatonAgent*

4.4.2.4 System Design

The final step of the system design involves using the deployment diagram to show the system structure. In constructing the diagram it has been taken into account that many instances of the same agent class can be running in the system at the same time. For example, many instances of the *PatientAgent*, *PhysicianAgent*, *NurseAgent* and *PharmacistAgent* will be running in the system at a given time. On the other hand, only one instance of the *MedicationAgent* and *MDBAgent* will be required. More than one instance of the *LabAgent* may be required in actual deployment, but in this system there will only be one instance of this agent. Figure 4.13 shows this deployment diagram.

This system will be running on a number of hosts at different geographical locations connected by a computer network. These may be spread within a town or city and between towns/cities, and even rural community clinic locations.

The main part of the system will run on a server in a central location named Server 1 in Figure 4.13. A patient agent for each patient will live on the server, and access the information about the patient and his medical conditions. The medication agent, pharmaceutical agent and *MDBAgent* will also reside on the server, where information about a patient's condition and information about drugs will be used to generate a prescription. The physician agents will be living on computers used by doctors, either in a hospital or a private practice, for example, Host 1 in Figure 4.13. The nurse agents will be on computers used by nurses at clinics in a town or a rural area, or at a hospital. These computers are shown as Host 2 and Host 3 in Figure 4.13. A computer used by a pharmacist in a pharmacy will host the pharmacist agent. There may be multiple pharmacist agents and their environments are shown as Host 4 and Host 5.

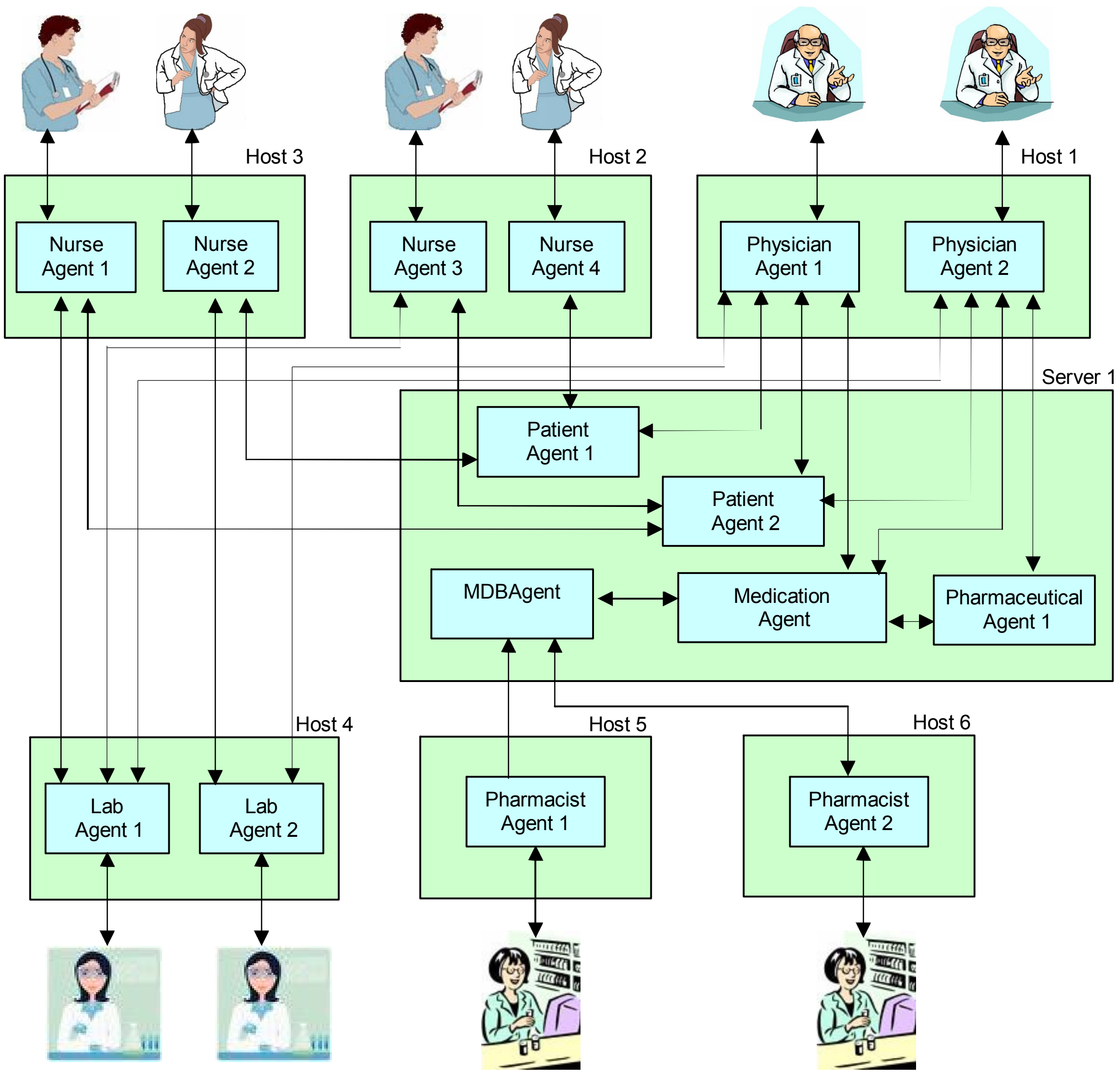


Figure 4.13: Deployment diagram

4.4.3 Database Design

The database design as shown in Figure 4.1 was not implemented, because of time constraints, but it was simplified to a single database.

The following tables were identified to be part of the database:

- ARVInfo
- ARVIssues
- ARVPrescriptions
- HealthCentres
- HealthWorkers
- MedicalConditions
- OtherMedicationIssues
- OtherPrescriptions
- PatientPersonalInfo

4.4.3.1 Table Information

ARVInfo

This table stores the information about the different types of ARV. It contains the following attributes:

ARVID is an abbreviation of ARV name, which is used by health workers. This is the primary key of the table.

ARVName contains the full name of the ARV drug.

ARVDoseless40, *ARVDose40to60*, and *ARVDosemore60* contain the dose of the specific ARV for a person weighing less than 40 kg, between 40 and 60 kg, and more than 60 kg respectively. *ARVMessage* contains usage instructions for the ARV.

ARVIssues

Each record in this table contains information about the ARVs issued by a specific health worker at a specific time. The field *issueID*, a unique identifier, is the primary key.

The fields *patientID* and *prescriptionID* identify the patient that received the medicine and the prescription from which it was issued. The field *healthWorker* indicates the pharmacist or nurse that issued the ARVs. The *date* field contains the date of the issue. The four attributes *ARV1*, *ARV2*, *ARV3* and *ARV4* contain the abbreviated names of the ARVs that were issued.

ARVPrescriptions

This table contains the information regarding each prescription. The *prescriptionID* is a unique number that identifies the prescription, which is the primary key. It also contains a *patientID*, the ID of the *doctor* who issued the prescription and the date of the prescription (*prescriptionDate*). There are four fields *ARV1*, *ARV2*, *ARV3* and *ARV4* that contain the IDs of up to four different ARVs that may be part of one prescription. For each of these ARVs there are two counters: one that indicates the number of months for which the ARV is prescribed and another that indicates the number of months for which the medicine has been issued. The value *ARVDose* indicates the category in which the dose is calculated according to the weight of the patient.

HealthCentres

This table stores the names of the clinics and hospitals that are part of the system. It contains a unique counter (*centreNo*) as well as the name of the hospital or clinic (*centreName*).

HealthWorkers

This table stores the names (*healthworkerFirstname*) and surnames (*healthworkerFirstname*) of health workers, as well as their role (*healthworkerRole*).

MedicalConditions

This table contains information about the medical conditions of a specific patient at a specific visit to the clinic or hospital. The counter *visitCount* combined with the *patientID* provides a unique identification of the visit. Other information about the consultation is stored in field *healthWorker*, which is the ID of the health worker who did the examination and the *visitDate* that contains the date of the visit.

The values stored in this table include the physical aspects of *weight*, *temperature* and *respiratoryRate* as well as the results of blood tests: the CD4 count (*CD4*), viral load (*VL*), *ALT*, *Hb*, *fastingGlucose*, *fastingCholesterol* and *Triglycerides*.

OtherMedicationIssues

This table stores the same type of information as that in *ARVIssues*, and contains Boolean values to indicate whether the patient was issued with *Cotrimoxazole*, *Fluconazole*, *INH* or *TBTreatment*. These are medications that may be issued by a nurse during a clinic visit, and no prescription is needed.

OtherPrescriptions

This table stores prescriptions other than ARVs that may be issued by a doctor, and each record contains information about a single medication. The fields *prescriptionID*, *patientID*, *doctor* and *prescriptionDate* all have the same meaning as in *ARVPrescriptions*, but each prescription only has a single attribute, *Medication*, that contains the medication name, dose and duration of one prescribed drug.

PatientPersonalInfo

The table is used to store the personal information about a particular patient. The *patientID* attribute is a unique identification of the patient.

The *treatmentStage* indicates what tests have been completed, if the patient is ready for ARV treatment, or if the patient is already receiving ARVs. Other fields include *firstName*, *surname*, *dateOfBirth*, *IDNumber*, *gender*, etcetera.

4.4.3.2 Relationships

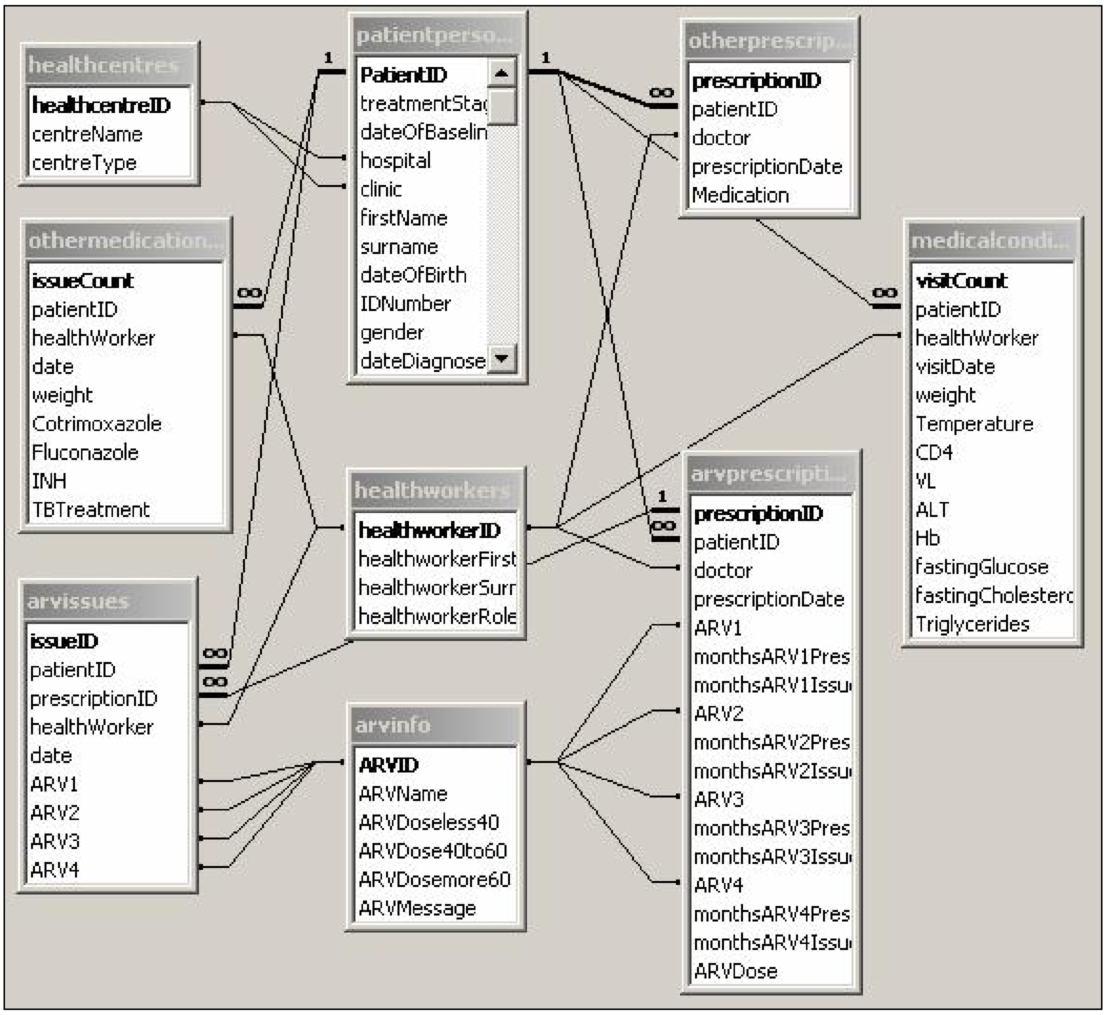


Figure 4.14: Relationships between tables in the database

The relationships between the tables are shown in Figure 4.14. For each patient whose personal information is stored in *patientPersonalInfo* there can be a number of records in *medicalConditions*, *arvPrescriptions*, *arvIssues*, *otherPrescriptions* and *otherMedicationIssues*. Multiple records will be created to contain information about the medical conditions of the patient at different stages, the ARVs and other medications prescribed, and the issues of the prescribed drugs.

For each prescription in *arvPrescriptions* there can be more than one record in *arvIssues* because a prescription is usually given for six months, but the patient is issued with one month's supply of drugs at a time. It is also possible to issue only a single drug or some components of a prescription.

The table *arvInfo* contains a single record for each type of ARV. There is a one-to-many relationship between the *arvID* in this table and the four fields *ARV1*, *ARV2*, *ARV3* and *ARV4* in both *arvPrescriptions* and *arvIssues*.

4.4.3.3 Normalisation

The design of the tables was done in such a way as to exclude the possibility of update, insertion and deletion anomalies, and to reduce data duplication and inconsistencies that could lead to loss of integrity of the database. To accomplish this goal the tables in the database were normalised to at least the third normal form.

The criteria for the first normal form are that a table must be guaranteed not to have duplicate records, and that every column must be atomic. These criteria were met by identifying a primary key for each table, and by placing only a single value in each field. The criterion for second normal form is that it does not include partial dependencies. This was accomplished by creating separate tables to store data that is only dependant on a single primary key that consists of just one attribute. Third normal form also requires that a table must contain no transitive dependencies. This was reached by placing attributes that are only indirectly dependent on the primary key in separate tables.

C h a p t e r 5

5 IMPLEMENTATION OF MedAgent

5.1 Overview

The agent classes as well as the conversations between them as discussed in the previous chapter and shown in the class diagram (Figure 4.8), were implemented. The only exception is the counsellor agent, whose role was implemented as a component of the task of the nurse agent.

5.2 The *JADE* Platform

MedAgent was implemented by using *JADE* (Java Agent DEvelopment Framework) (*JADE*, 2005). *JADE* is a software framework fully implemented in the Java programming language. To utilise this framework, agents should be implemented in Java.

JADE simplifies the implementation of multi-agent systems through a middle-ware that complies with the *FIPA* (Foundation for Intelligent Physical Agents) specifications (*FIPA*, 2005). It provides a runtime environment where agents can “live”, and a library of classes that can be used in application development. It also contains a set of graphical tools that support the debugging and deployment phases.

The agent platform can be distributed across machines, which could even use different Operating Systems and the configuration can be controlled via a remote GUI. The configuration can even be changed at run-time by moving agents from one machine to another, as and when required. The minimal system requirement for running *JADE* is version 1.4 of *JAVA* (the run time environment or the *JDK*). The latest version of *JADE* at the time of writing, *JADE* 3.3, which was released on 2nd March 2005, was used in this implementation.

JADE is composed of a number of packages including the following:

- The kernel of the system is JADE.core, which includes the Agent class that must be extended to create the different agents in the application.
- A sub-package *JADE.core.behaviours* contains a *Behaviour* class hierarchy. Behaviours implement the tasks, or intentions, of an agent. They are logical units of activity that can be composed in various ways to achieve complex execution patterns and these units can be executed concurrently.
- The *JADE.lang.acl* package processes Agent Communication Language according to *FIPA* standard specifications.
- The *JADE.domain* package contains classes that represent agent management, for example, the Agent Management System (*AMS*) agent that provides the naming service and the Directory Facilitator (*DF*) agent that provides yellow pages services.

JADE also contains tools that simplify platform administration and application development. These tools include a Remote Management Agent (*RMA*), which acts as a graphical console for platform management and control, and a number of other agents for monitoring the activities of running agents. One of these agents, called the *SnifferAgent*, is used to track and graphically display the messages sent between a set of agents selected by the user.

5.2.1 *JADE* Behaviours

The tasks of an agent are implemented in *JADE* as *Behaviour* objects. Behaviours can be added to an *Agent* class whenever they are needed to accomplish the agent's tasks. Two of the *Behaviour* classes that have been used frequently in the implementation of the *MedAgent* system are the *CyclicBehaviour* and the *OneShotBehaviour*. The *CyclicBehaviour* stays active as long as its agent is alive and will be called repeatedly after every event. This behaviour is useful to handle message reception. The *OneShotBehaviour* executes only once and dies. This behaviour is used when an agent has to execute a task only once, such as sending a request or a reply. A combination of the different behaviours can be used to create a complex series of tasks executed in a particular order.

5.2.2 *JADE* Messages

Individual agents communicate and interact with each other through the exchange of messages. *JADE* messages are all objects of the *ACLMessage* class in *JADE* that represents ACL (Agent Communication Language) messages. The attributes and interaction protocols of this class are defined according to *FIPA* standards. When a message is created, the content of the message as well as different attributes are set to describe and identify the message. Some of the frequently used attributes include the *Performative* (type), *Receiver*, *Sender* (initialised automatically) and *ConversationID* attributes.

5.3 The Agents implemented in *MedAgent* and their Behaviours

5.3.1 PatientAgent

The *PatientAgent* represents a single patient and handles all access to the personal data of that patient. It contains a number of cyclic behaviours that are executed to serve incoming requests from other agents, as well as a number of oneshot behaviours to send replies. Its cyclic behaviours are summarised below.

- *DisplayHistory:* to serve incoming requests to display the patient history. It displays a list of all visits of the patient to health workers with the visit date, the name of the health worker, as well as the results of the physical examination and blood tests. It also displays a list of all the

ARV issues that were done for the patient, together with the date and the name of the health worker that issued the ARVs.

- *GetBloodResults*: to serve incoming requests from a *PhysicianAgent* to get information of the last blood tests of the patient. The *PatientAgent* replies by sending a message to the sender of the message with the test results and the test date.

- *GetClinicVisitData:* to serve incoming requests from a *NurseAgent* to get clinic visit data. The *PatientAgent* replies by sending a message to the sender of the message with the personal data of the patient, as well as the medical conditions during the last visit, and the results of the physical and blood tests done at that point in time.

- *GetPatientData*: to serve incoming requests from a *PhysicianAgent* to get patient data. The *PatientAgent* sends a reply to the sender of the message, which contains the information about that patient that was acquired by the doctor during the baseline test.

- *GetPrescriptionData*: to serve incoming requests from the *MedicationAgent*, to get patient data necessary for the compilation of a prescription. Information about the gender, weight and physical condition that could influence the combination of medication, are sent in reply to these requests.

- *UpdateBloodResults*: to serve incoming requests to update results of blood tests. After blood tests for a specific patient have been done by a technician at a laboratory, the *LabAgent* sends the results to the *PatientAgent*. The *PatientAgent* updates the medical conditions of the patient by adding the results of the blood tests to the patient's information.

- *UpdateClinicVisit*: to serve incoming requests to update data obtained during a clinic visit. When the visit of a patient to a clinic nurse has been completed, the *NurseAgent* sends the information that was gathered during the visit to the *PatientAgent*. The *PatientAgent* creates a record of that specific visit in the database.

- *UpdatePatientData*: to serve incoming requests to update patient data acquired by a doctor during a hospital visit. After a patient has completed a visit to a doctor, the *PhysicianAgent* sends the information updated by the doctor to the specific *PatientAgent*, to update its patient database.

5.3.2 NurseAgent

The *NurseAgent* gets and updates patient data through the *PatientAgent*, and gets prescription data from the *MedicationAgent*. When the patient visits the nurse, the *NurseAgent* will determine whether the patient exists on the system. If the patient exists, the *NurseAgent* requests the personal information of the patient from the *PatientAgent.* The *NurseAgent* also requests the prescription data from the *MedicationAgent*. The nurse issues medicines from the prescription and the *NurseAgent* informs the *MedicationAgent* which drugs were issued. The *NurseAgent* uses the following behaviours:

- *CreatePatient:* oneshot behaviour to create a new *PatientAgent* when patient visits clinic for the first time.

- *VisitClinic*: oneshot behaviour to start a visit session and get information about a patient visiting the clinic who is already on ARVs.

- *PatientDataReceived*: cyclic behaviour to receive a message from a *PatientAgent* containing a patient's personal information.

- *VisitDataReceived*: cyclic behaviour to receive a message from a *PatientAgent* containing information about the previous visit of the patient to a clinic nurse.

- *PrescriptionReceived*: cyclic behaviour to receive a message from the *MedicationAgent* containing the names of the drugs prescribed to a patient.
- *PatientDataUpdated*: cyclic behaviour to receive a message from a *PatientAgent* to confirm the update of patient data.
- *UpdatePatientData*: oneshot behaviour to send a message with the personal information of a new patient to the *PatientAgent*.
- *GetPatientData*: oneshot behaviour to send a request to a *PatientAgent* to get the patient's personal and medical information.
- *GetPatientHistory*: oneshot behaviour to send a request to a *PatientAgent* and the *MDBAgent* to get the history of a patient and the medication issued to the patient.
- *SendBlood*: oneshot behaviour to send a request to the *LabAgent* to do a blood test.
- *UpdateClinicVisit*: oneshot behaviour to send a request to a *PatientAgent*, to update the data acquired or changed during the clinic visit.

5.3.3 PhysicianAgent

The *PhysicianAgent* represents a doctor at a hospital or private practice. The *PhysicianAgent* communicates with the *PatientAgent*, *MedicationAgent* and *MDBAgent*. The *PhysicianAgent* has a number of oneshot behaviours that are executed to send requests to the other agents, as well as a number of cyclic behaviours to receive replies. Its behaviours are summarised below:

- *BloodResultsReceived*: cyclic behaviour to receive the results of the last blood test of a patient from the *PatientAgent*.

- *GetPatientData*: oneshot behaviour sending a request to a *PatientAgent* to get the data of that patient stored at the previous visit.
- *IssuePrescription*: oneshot behaviour sending a request to the *MedicationAgent* to generate a prescription.
- *PatientDataReceived*: cyclic behaviour to receive data from a *PatientAgent* in reply to the message sent in the *GetPatientData* behaviour.
- *PatientDataUpdated*: cyclic behaviour to receive a reply from a *PatientAgent* to confirm the update of patient data.
- *PrescriptionReceived*: cyclic behaviour to receive the prescription from the *MedicationAgent.*
- *UpdatePatientData*: oneshot behaviour sending the data acquired during a visit to the doctor to a *PatientAgent*, to update the database for that patient.

5.3.4 LabAgent

The *LabAgent* receives requests from the *NurseAgent* or the *PhysicianAgent* to do blood tests and sends back the test results in reply to these requests. It uses the following two behaviours to accomplish this task:

- *BloodReceived*: cyclic behaviour used to receive a request to do the tests on one blood sample sent to the lab.
- *SendBloodResults*: oneshot behaviour to send the results of the blood test to the *PatientAgent* to which it belongs, to update the record on the database.

5.3.5 MDBAgent

The *MDBAgent* (Medical Database Agent) handles all access to the medical data on the system. Information about drugs, prescriptions and drug issues are manipulated by the *MDBAgent.* Other agents, for example, the *NurseAgent*, *PhysicianAgent*, *PharmacistAgent* and *LabAgent* request data from the *MDBAgent*, and send data to the *MDBAgent* to update the medical database. The behaviours used by this agent are:

- *GetMedicationData*: cyclic behaviour to receive a request for medication data from the *MedicationAgent* to be used in the preparation of a prescription.
- *GetPrescriptionData*: cyclic *behaviour* to receive a request from a *PharmacistAgent* or *NurseAgent* to send the information about the current prescription for a specific patient and the number of issues left for each drug.
- *NewPrescription*: cyclic *behaviour* to receive the information about a new prescription from the *MedicationAgent* and create a prescription record.
- *UpdateIssue*: cyclic behaviour to receive a request from a *PharmacistAgent* or *NurseAgent* to update the information about a patient's prescription and drugs issued.

5.3.6 MedicationAgent

The main task of the *MedicationAgent* is to generate prescriptions as requested by a *PhysicianAgent*. This is done by communicating with a *PatientAgent* and the *MDBAgent* to retrieve the information about the patient and the drugs that could be suitable for the patient. It does this by executing the cyclic behaviour *PrescriptionProcess* with the following steps:

1 Receive a message from a *PhysicianAgent* and send a request for the personal data of that patient to the *PatientAgent*.

2 Receive the patient's data from the *PatientAgent* and request the data about the different medications from the *MDBAgent*.

3 Receive the medication data from the *MDBAgent*, compile and send a prescription to the *PhysicianAgent*.

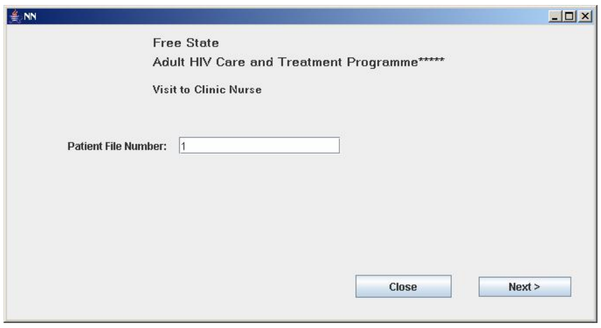
When the doctor is satisfied with the prescription, the *PhysicianAgent* sends a message to the *MedicationAgent*. The message is received in cyclic behaviour *AcceptPrescription*, which sends the prescription information to the *MDBAgent* to create a new prescription record.

5.4 Use of the System

The people who use the system are nurses, physicians, pharmacists and laboratory workers. Each person interacts with a specific agent by entering data and making choices in graphical user interfaces. This information is used by the agent that in turn interacts with other agents to make decisions, store data and keep track of events. The use of the system in terms of how a nurse, physician, pharmacist and laboratory worker interact with the system is discussed in this section.

5.4.1 Nurse

When the *NurseAgent* is activated, it executes method *newConsultation* and the interface *NurseGui* is displayed as indicated in Figure 5.1 below:


Free State
Adult HIV Care and Treatment Programme*****
Visit to Clinic Nurse

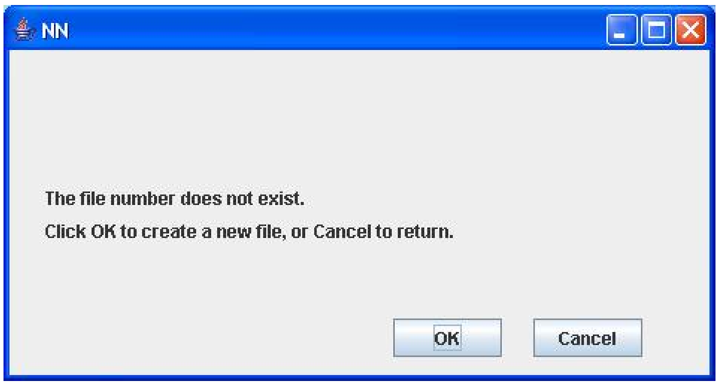
Patient File Number: 1

Figure 5.1: The *NurseAgent* GUI for entering patient ID

The nurse enters the *patientID* and the *NurseAgent* communicates with the *AMSAgent* (Agent Management System Agent) to determine if that *PatientAgent* is active on the system. If the patient is found, a sequence of tasks will be executed for an existing patient, but if the patient is not found, the nurse will have the opportunity to create a new patient. These sequences of events are described below.

5.4.1.1 New Patient

If the patient does not exist on the database, the message box shown in Figure 5.2 is displayed and the nurse can press "*OK*" to create a new patient record, or "*Cancel*" to go back to the initial interface to retype the patient number.

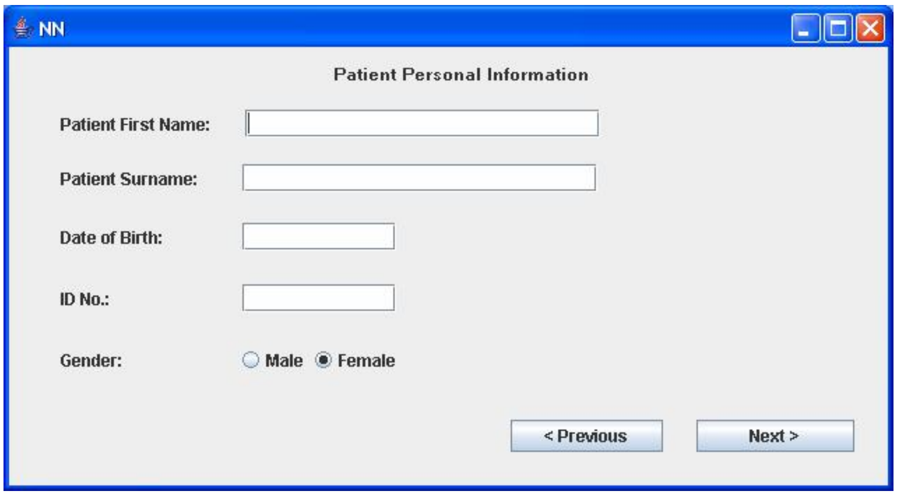

NN

The file number does not exist.

Click OK to create a new file, or Cancel to return.

Figure 5.2: Option to create new patient

When a new patient is created, the nurse has to fill in the personal information of the patient as shown in Figure 5.3.



NN

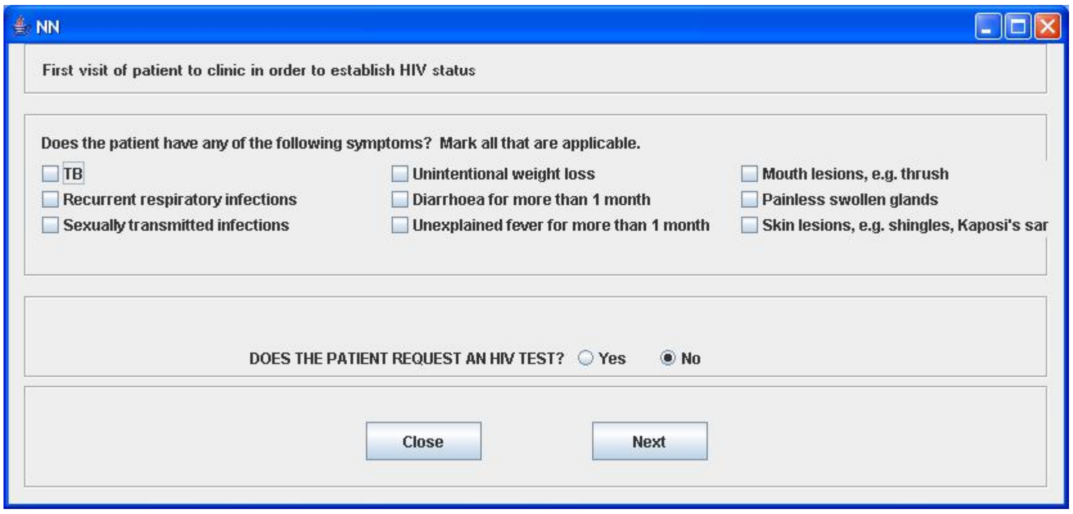
Patient Personal Information

Patient First Name:	
Patient Surname:	
Date of Birth:	
ID No.:	
Gender:	○ Male [✓ Female]

< Previous   Next >

Figure 5.3: Enter personal information of new patient

The nurse will then start to establish the HIV status of the patient by examining the patient according to the list of symptoms that could indicate a positive HIV status as displayed in Figure 5.4.



NN

First visit of patient to clinic in order to establish HIV status

Does the patient have any of the following symptoms? Mark all that are applicable.

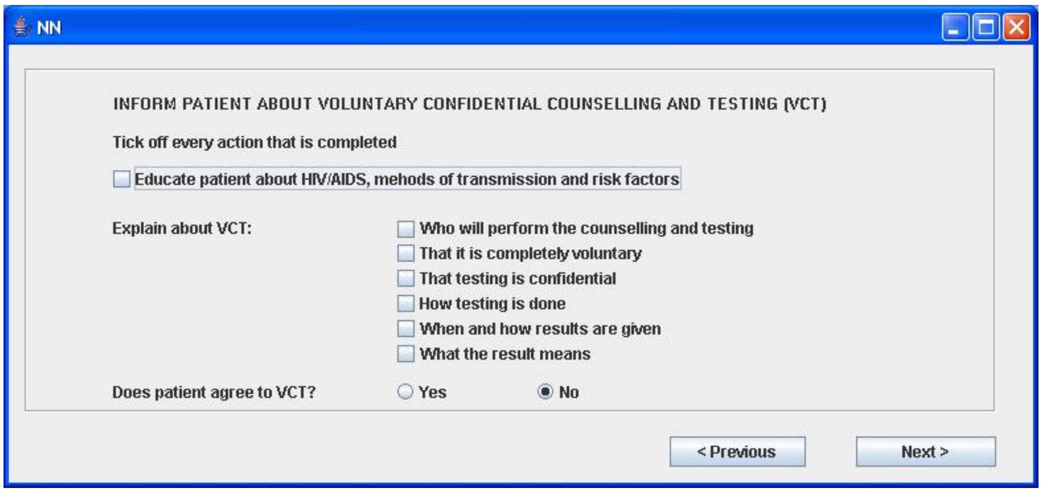
☐ TB	☐ Unintentional weight loss	☐ Mouth lesions, e.g. thrush
☐ Recurrent respiratory infections	☐ Diarrhoea for more than 1 month	☐ Painless swollen glands
☐ Sexually transmitted infections	☐ Unexplained fever for more than 1 month	☐ Skin lesions, e.g. shingles, Kaposi's sar

DOES THE PATIENT REQUEST AN HIV TEST? ○ Yes [✓ No]

Close   Next

Figure 5.4: Establish HIV status

After completion of this examination the nurse continues to inform the patient about the process of voluntary confidential counselling and testing (VCT) as shown in Figure 5.5.



NN

INFORM PATIENT ABOUT VOLUNTARY CONFIDENTIAL COUNSELLING AND TESTING (VCT)

Tick off every action that is completed

☐ Educate patient about HIV/AIDS, mehods of transmission and risk factors

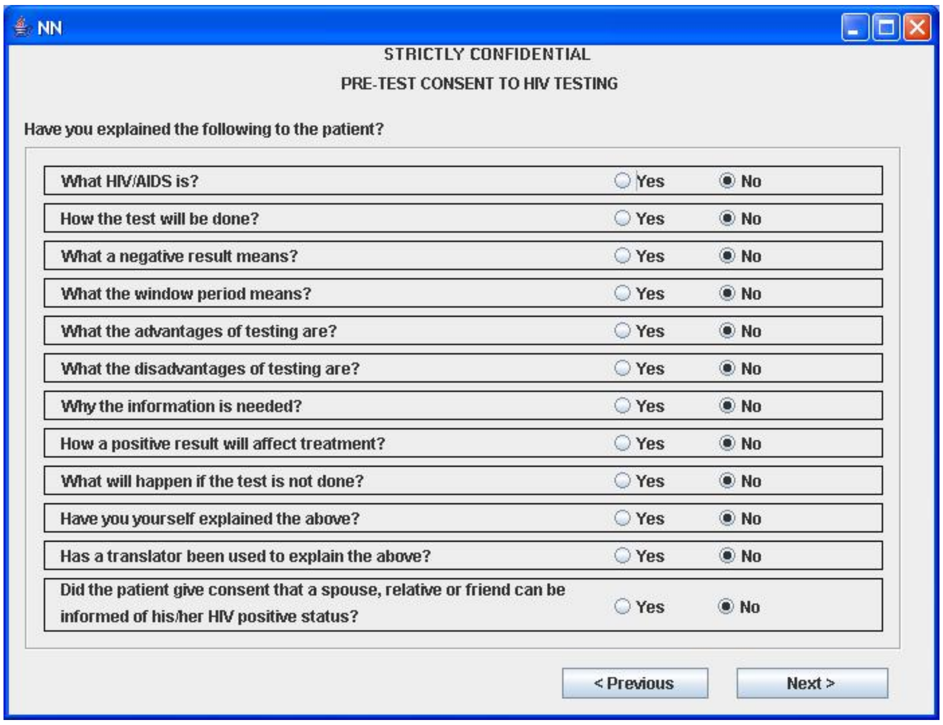
Explain about VCT:
- ☐ Who will perform the counselling and testing
- ☐ That it is completely voluntary
- ☐ That testing is confidential
- ☐ How testing is done
- ☐ When and how results are given
- ☐ What the result means

Does patient agree to VCT? Yes [✓ No]

< Previous | Next >

Figure 5.5: Inform about VCT

After explaining the VCT, the nurse has to explain the testing procedure to the patient, and tick off every item on the form as displayed in Figure 5.6.



NN

STRICTLY CONFIDENTIAL

PRE-TEST CONSENT TO HIV TESTING

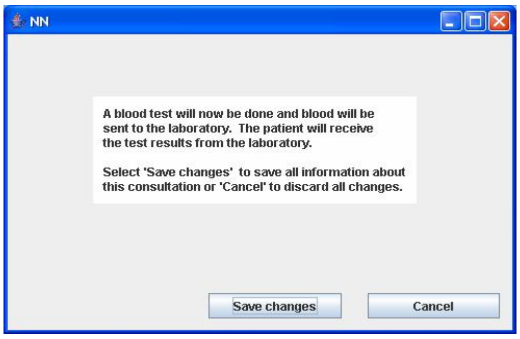
Have you explained the following to the patient?

What HIV/AIDS is?	Yes	[✓ No]
How the test will be done?	Yes	[✓ No]
What a negative result means?	Yes	[✓ No]
What the window period means?	Yes	[✓ No]
What the advantages of testing are?	Yes	[✓ No]
What the disadvantages of testing are?	Yes	[✓ No]
Why the information is needed?	Yes	[✓ No]
How a positive result will affect treatment?	Yes	[✓ No]
What will happen if the test is not done?	Yes	[✓ No]
Have you yourself explained the above?	Yes	[✓ No]
Has a translator been used to explain the above?	Yes	[✓ No]
Did the patient give consent that a spouse, relative or friend can be informed of his/her HIV positive status?	Yes	[✓ No]

< Previous | Next >

Figure 5.6: Explanation of testing procedure and pre-test consent

When the pre-test procedure is completed and the patient has agreed to be tested, the dialogue box shown in Figure 5.7 is displayed to remind the nurse that the blood test must now be done and the blood must be sent to the laboratory for testing. When the nurse clicks "*Save changes*", the *NurseAgent* sends information about the patient to the *PatientAgent* to update and it sends a message to the *LabAgent* to indicate that blood was sent to the laboratory to be tested.



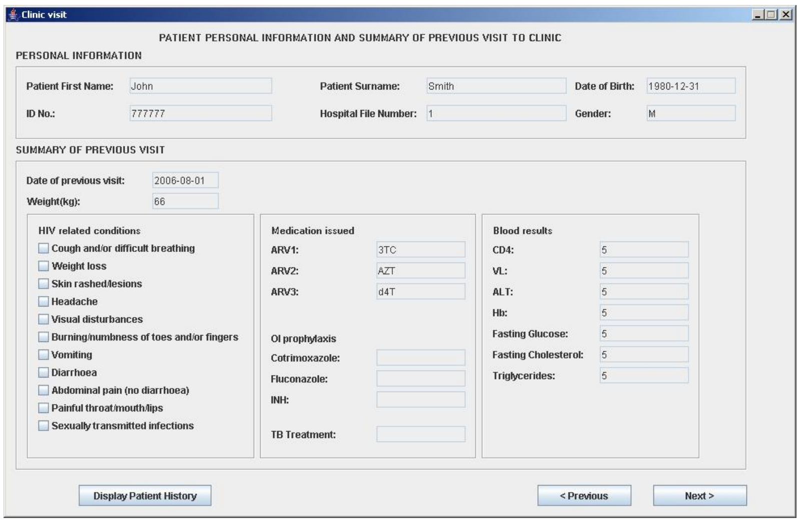
A blood test will now be done and blood will be sent to the laboratory. The patient will receive the test results from the laboratory.

Select 'Save changes' to save all information about this consultation or 'Cancel' to discard all changes.

Figure 5.7: Blood tests will be done

5.4.1.2 Existing Patient

If the *PatientAgent* exists on the system, the *NurseAgent* executes behaviour *VisitClinic* to execute the tasks that have to be performed during a follow-up visit to the nurse. The *NurseAgent* communicates with the *PatientAgent* to retrieve the personal information of the patient. It also communicates with the *MDBAgent* to retrieve information about the latest prescription for that patient, from which the nurse will issue medications. The personal information of the patient as shown in Figure 5.8 is the first interface to be displayed.



PATIENT PERSONAL INFORMATION AND SUMMARY OF PREVIOUS VISIT TO CLINIC

PERSONAL INFORMATION

Patient First Name:	John	Patient Surname:	Smith	Date of Birth:	1980-12-31
ID No.:	777777	Hospital File Number:	1	Gender:	M

SUMMARY OF PREVIOUS VISIT

Date of previous visit: 2006-08-01

Weight(kg): 66

HIV related conditions

- ☐ Cough and/or difficult breathing
- ☐ Weight loss
- ☐ Skin rashed/lesions
- ☐ Headache
- ☐ Visual disturbances
- ☐ Burning/numbness of toes and/or fingers
- ☐ Vomiting
- ☐ Diarrhoea
- ☐ Abdominal pain (no diarrhoea)
- ☐ Painful throat/mouth/lips
- ☐ Sexually transmitted infections

Medication issued

ARV1:	3TC
ARV2:	AZT
ARV3:	d4T

OI prophylaxis

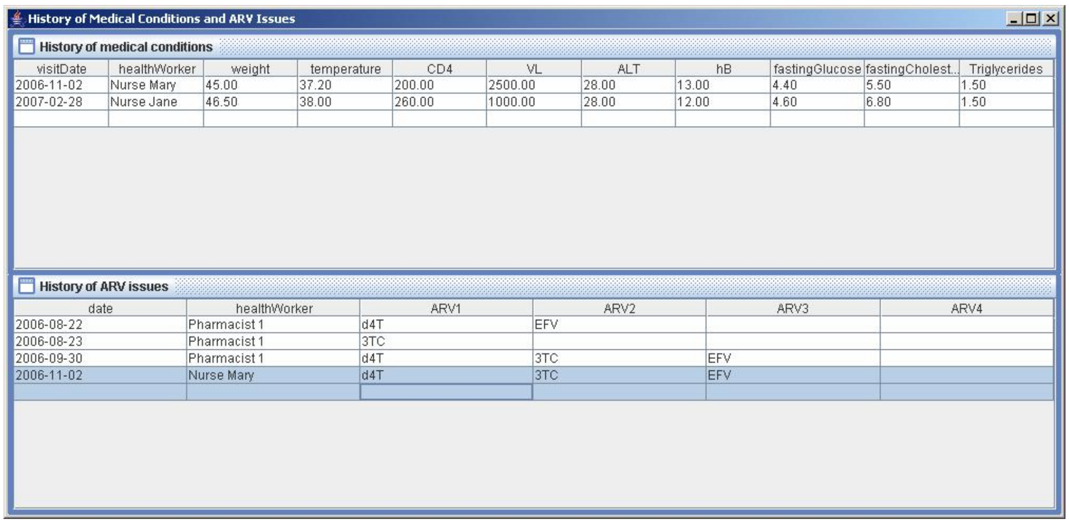
Cotrimoxazole:	
Fluconazole:	
INH:	
TB Treatment:	

Blood results

CD4:	5
VL:	5
ALT:	5
Hb:	5
Fasting Glucose:	5
Fasting Cholesterol:	5
Triglycerides:	5

Figure 5.8: The patient personal information window

The patient history, which includes the history of the medical conditions of the patient as well as the ARV issues made to the patient, can be displayed by clicking on the appropriate button *"Display Patient History"* in Figure 5.8, and the results will be as shown in Figure 5.9 below.



History of medical conditions

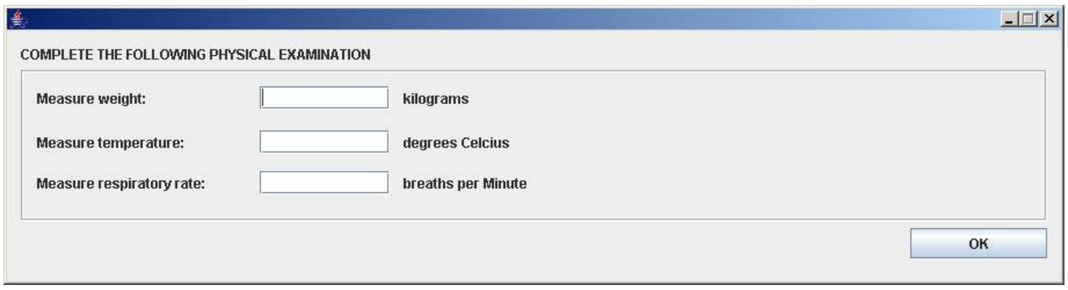
visitDate	healthWorker	weight	temperature	CD4	VL	ALT	hB	fastingGlucose	fastingCholest...	Triglycerides
2006-11-02	Nurse Mary	45.00	37.20	200.00	2500.00	28.00	13.00	4.40	5.50	1.50
2007-02-28	Nurse Jane	46.50	38.00	260.00	1000.00	28.00	12.00	4.60	6.80	1.50

History of ARV issues

date	healthWorker	ARV1	ARV2	ARV3	ARV4
2006-08-22	Pharmacist 1	d4T	EFV		
2006-08-23	Pharmacist 1	3TC			
2006-09-30	Pharmacist 1	d4T	3TC	EFV	
2006-11-02	Nurse Mary	d4T	3TC	EFV	

Figure 5.9: Window showing the history of medical conditions and ARV issues

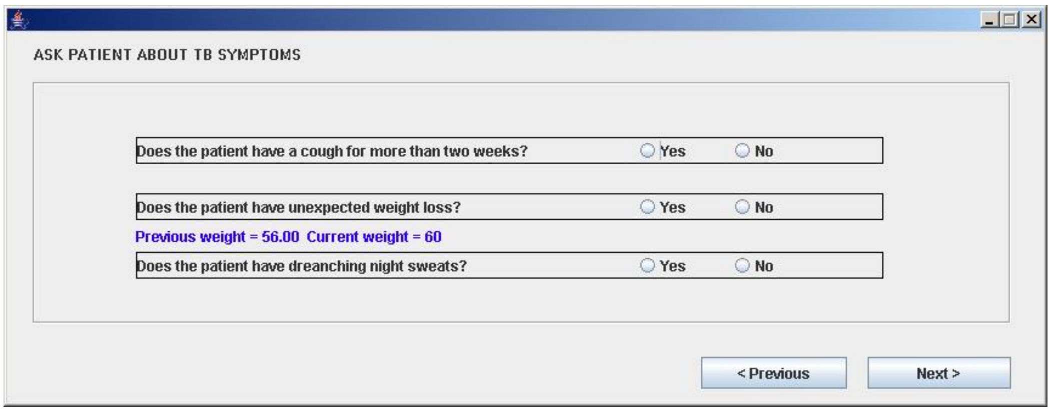
In the next step the nurse has to do a physical examination of the patient to measure the weight, temperature and respiratory rate of the patient and enter it. Figure 5.10 below shows the interface used for this step.


COMPLETE THE FOLLOWING PHYSICAL EXAMINATION

Measure weight:		kilograms
Measure temperature:		degrees Celcius
Measure respiratory rate:		breaths per Minute

Figure 5.10: Capturing physical examination data

Thereafter, the nurse will ask the patient about possible TB symptoms. A message is displayed that compares the weight of the patient at the previous visit with the current weight as depicted in Figure 5.11 below.



ASK PATIENT ABOUT TB SYMPTOMS

Does the patient have a cough for more than two weeks? ○ Yes ○ No

Does the patient have unexpected weight loss? ○ Yes ○ No

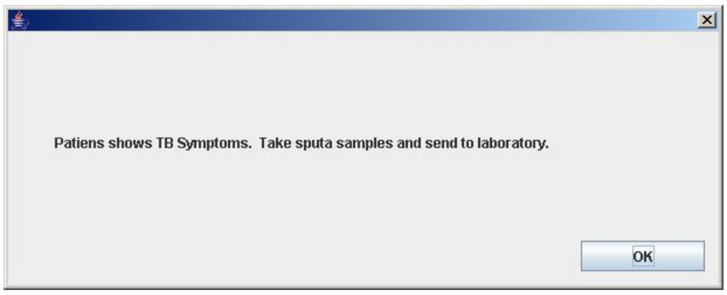
Previous weight = 56.00 Current weight = 60

Does the patient have dreanching night sweats? ○ Yes ○ No

< Previous | Next >

Figure 5.11: About patient TB symptoms

If any of the answers to the questions is positive, a message is displayed for the nurse to send sputa samples to the laboratory to test for TB as indicated in Figure 5.12.

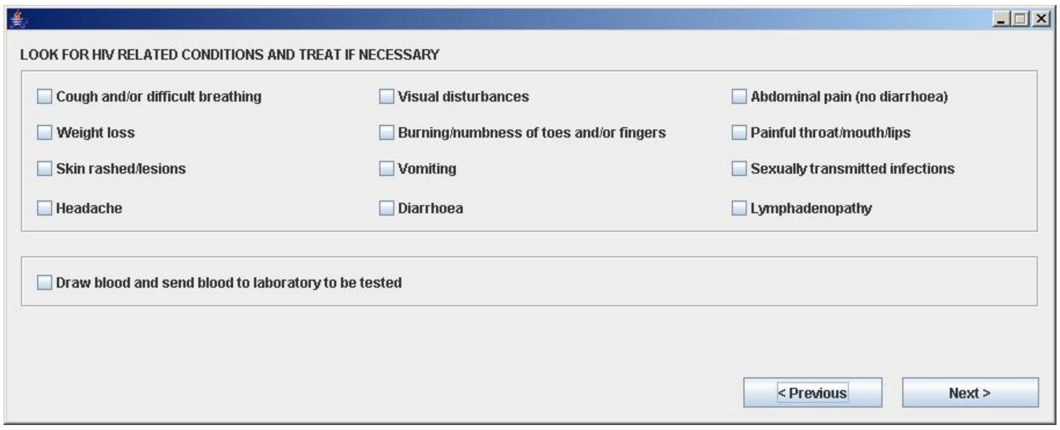


Patiens shows TB Symptoms. Take sputa samples and send to laboratory.

OK

Figure 5.12: If patient shows any TB symptoms, samples are sent to a laboratory

The next step is to look for HIV related conditions as displayed in Figure 5.13 below and treat them.



LOOK FOR HIV RELATED CONDITIONS AND TREAT IF NECESSARY

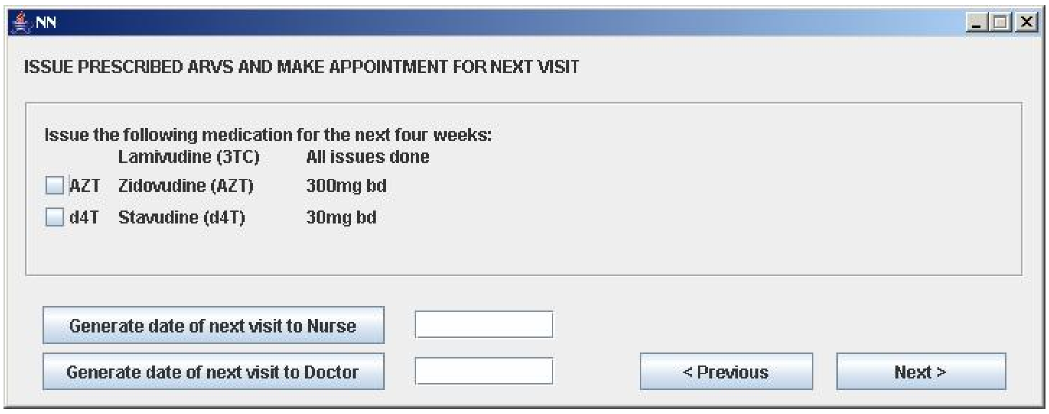
☐ Cough and/or difficult breathing	☐ Visual disturbances	☐ Abdominal pain (no diarrhoea)
☐ Weight loss	☐ Burning/numbness of toes and/or fingers	☐ Painful throat/mouth/lips
☐ Skin rashed/lesions	☐ Vomiting	☐ Sexually transmitted infections
☐ Headache	☐ Diarrhoea	☐ Lymphadenopathy

☐ Draw blood and send blood to laboratory to be tested

< Previous | Next >

Figure 5.13: Treat HIV related conditions

The next task of the nurse is to issue the ARV drugs for the next four weeks. The medications as prescribed by a doctor are displayed as shown in Figure 5.14, and the nurse may select to issue one or more medications. Only the medications for which there are issues left may be chosen by the nurse. If all issues of a certain ARV have been done, a message will be displayed to indicate this, as shown in the case of the ARV Lamivudine (3TC) in Figure 5.14. The *NurseAgent* can generate the dates for the next visit to the clinic nurse and doctor by clicking the appropriate buttons, and the nurse will communicate this information to the patient.


ISSUE PRESCRIBED ARVS AND MAKE APPOINTMENT FOR NEXT VISIT

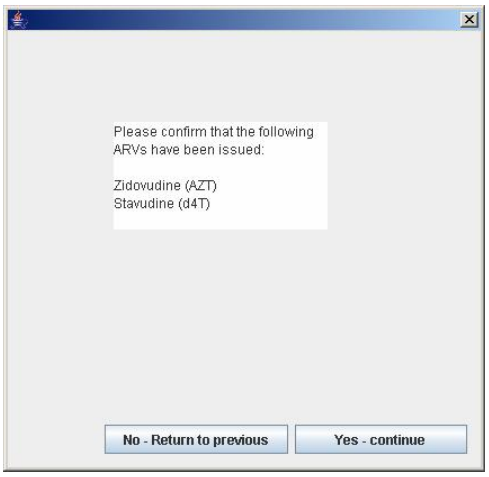
Issue the following medication for the next four weeks:

		Lamivudine (3TC)	All issues done
☐	AZT	Zidovudine (AZT)	300mg bd
☐	d4T	Stavudine (d4T)	30mg bd

NN
Generate date of next visit to Nurse
Generate date of next visit to Doctor
< Previous
Next >

Figure 5.14: Issue prescribed ARVs and make appointment for next visit

When the nurse clicks the "*Next*" button, the *NurseAgent* displays the message shown in Figure 5.15 and the nurse has to confirm that the appropriate ARVs were issued.

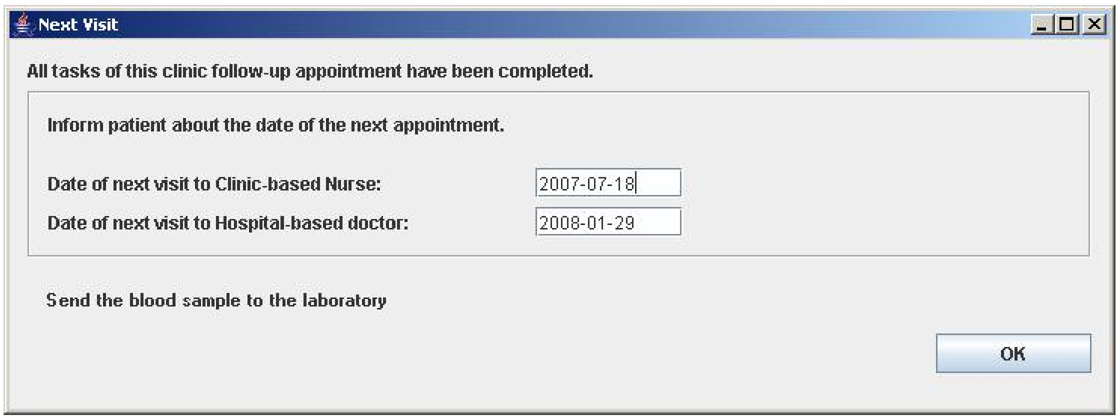


Please confirm that the following ARVs have been issued:

Zidovudine (AZT)
Stavudine (d4T)

Figure 5.15: Confirm ARV issues

Finally the *NurseAgent* informs the nurse that all tasks have been completed, and she is reminded to tell the patient the date of the next hospital and clinic visit as shown in Figure 5.16. The nurse is also reminded that the blood sample must be sent to the laboratory for testing. The *NurseAgent* now sends the patient information gathered during the visit to the *PatientAgent* to update, and it sends a message to the *LabAgent* that a blood sample has been sent and has to be tested.



Next Visit

All tasks of this clinic follow-up appointment have been completed.

Inform patient about the date of the next appointment.

Date of next visit to Clinic-based Nurse: 2007-07-18

Date of next visit to Hospital-based doctor: 2008-01-29

Send the blood sample to the laboratory

Figure 5.16: Tasks completed, patient reminded of next visit and blood sample sent to laboratory

5.4.2 Physician

The *PhysicianAgent* starts with an interface similar to the one displayed by the *NurseAgent* as was previously shown in Figure 5.1. The doctor has to type in the patient number and the *PhysicianAgent* will test whether the patient exists on the system. This is done similarly to the procedure followed by the *NurseAgent* as described in Section 5.3.1. If the patient exists, the *PhysicianAgent* determines the treatment stage of the patient and follows a set of procedures accordingly. The sequences of events for a patient coming for a first visit, as well as a patient coming for a follow-up visit, are described in the following sections.

5.4.2.1 First Visit to Doctor

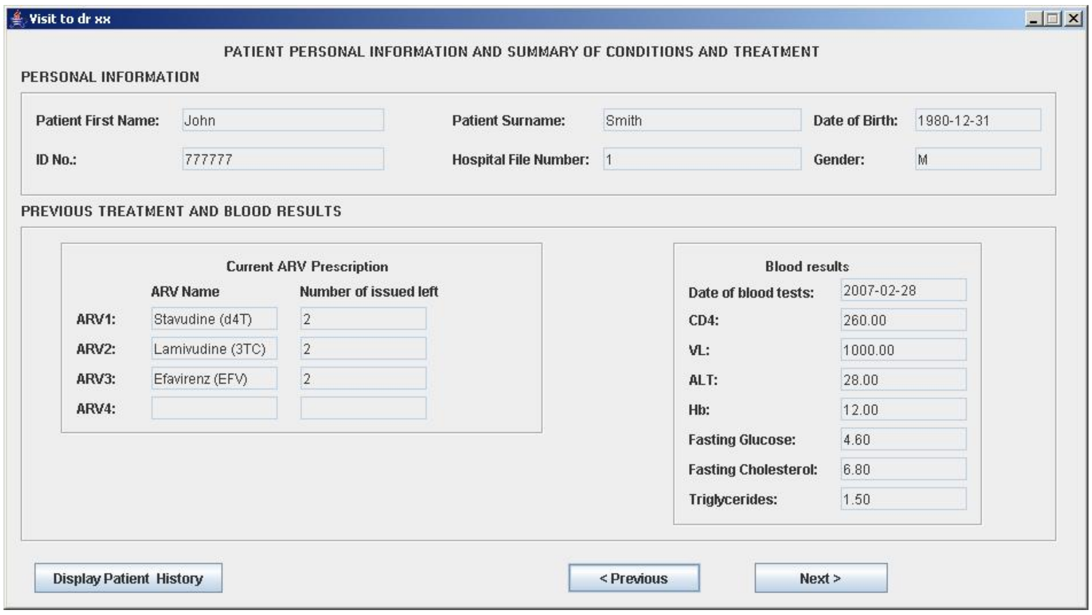
Sometimes a patient will visit the doctor without visiting the nurse at the clinic beforehand. In that case the patient does not exist on the system and the *PhysicianAgent* will create the patient. The doctor then starts by entering the personal details, followed by the information gathered during the medical examination.

If the patient does exist, the *PhysicianAgent* determines whether the current visit is the first one to a doctor, in which case the doctor will do a baseline examination, guided by the *PhysicianAgent,* which displays a series of input interfaces for the doctor to enter the results of the different baseline tests. The interfaces that are used to enter the data are shown in Appendix B.

During the visit to the doctor, the *PhysicianAgent* will determine from the test results whether the patient is at a stage where ARV treatment has to be started. If so, the doctor will send the patient for drug readiness training (DRT).

5.4.2.2 Follow-up Visit

When a patient visits the doctor for a follow-up, the personal information of the patient as well as the medical conditions and medication received previously are displayed as shown in Figure 5.17 below. The doctor examines the patient and enters the current information. The interfaces used during this examination are shown in Figures 5.18 and 5.19 below:



PATIENT PERSONAL INFORMATION AND SUMMARY OF CONDITIONS AND TREATMENT

PERSONAL INFORMATION

Patient First Name:	John	Patient Surname:	Smith	Date of Birth:	1980-12-31
ID No.:	777777	Hospital File Number:	1	Gender:	M

PREVIOUS TREATMENT AND BLOOD RESULTS

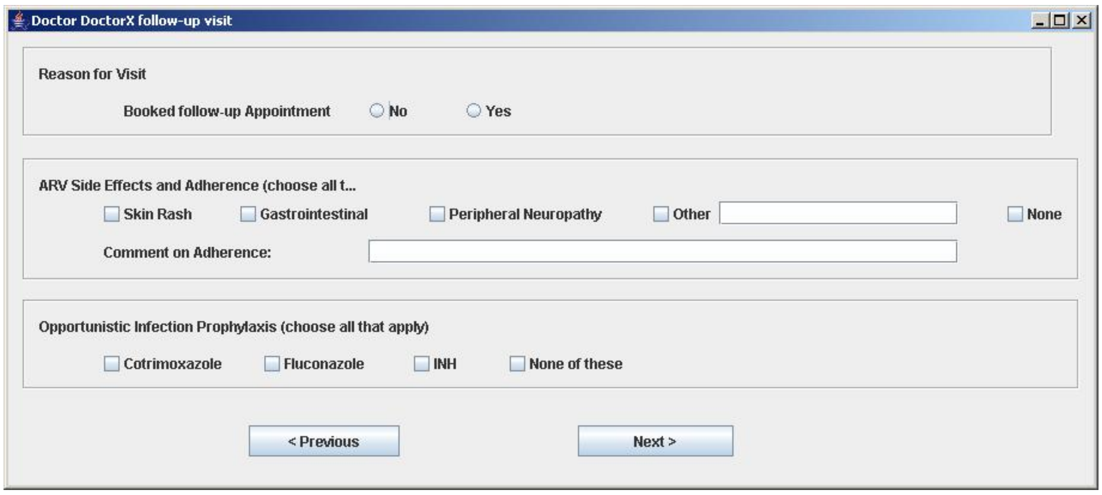
Current ARV Prescription

	ARV Name	Number of issued left
ARV1:	Stavudine (d4T)	2
ARV2:	Lamivudine (3TC)	2
ARV3:	Efavirenz (EFV)	2
ARV4:		

Blood results

Date of blood tests:	2007-02-28
CD4:	260.00
VL:	1000.00
ALT:	28.00
Hb:	12.00
Fasting Glucose:	4.60
Fasting Cholesterol:	6.80
Triglycerides:	1.50

Figure 5.17: Previously captured information shown during follow-up visit



Reason for Visit

Booked follow-up Appointment ○ No ○ Yes

ARV Side Effects and Adherence (choose all t...

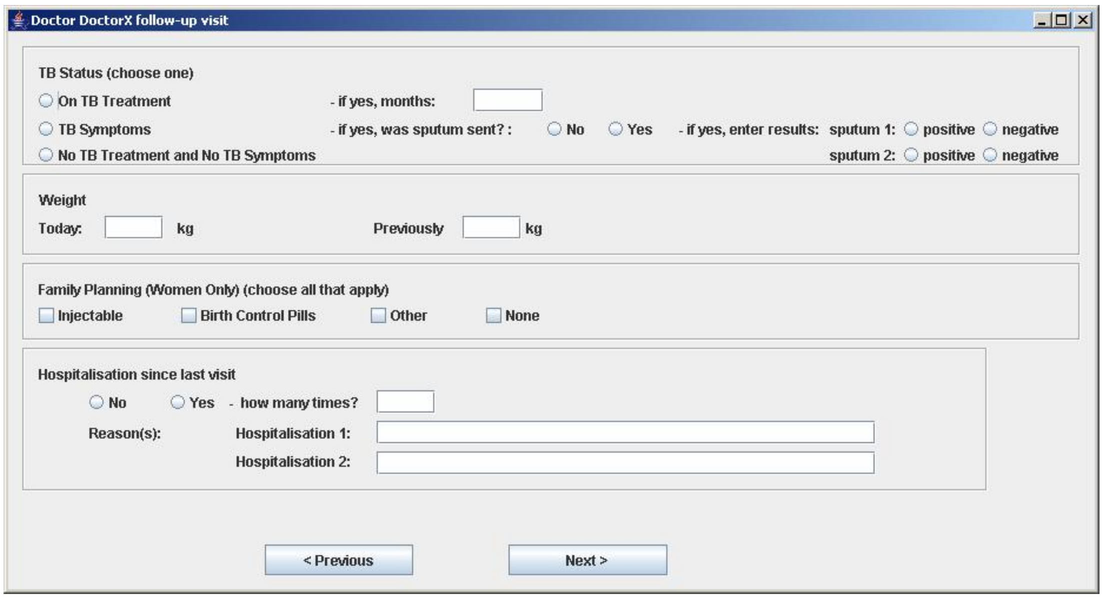
☐ Skin Rash ☐ Gastrointestinal ☐ Peripheral Neuropathy ☐ Other ☐ None

Comment on Adherence:

Opportunistic Infection Prophylaxis (choose all that apply)

☐ Cotrimoxazole ☐ Fluconazole ☐ INH ☐ None of these

Figure 5.18: First interface for entering data during follow-up visit



Doctor DoctorX follow-up visit

TB Status (choose one)

- On TB Treatment - if yes, months:
- TB Symptoms - if yes, was sputum sent? : No Yes - if yes, enter results: sputum 1: positive negative
- No TB Treatment and No TB Symptoms sputum 2: positive negative

Weight

Today: kg Previously kg

Family Planning (Women Only) (choose all that apply)

Injectable Birth Control Pills Other None

Hospitalisation since last visit

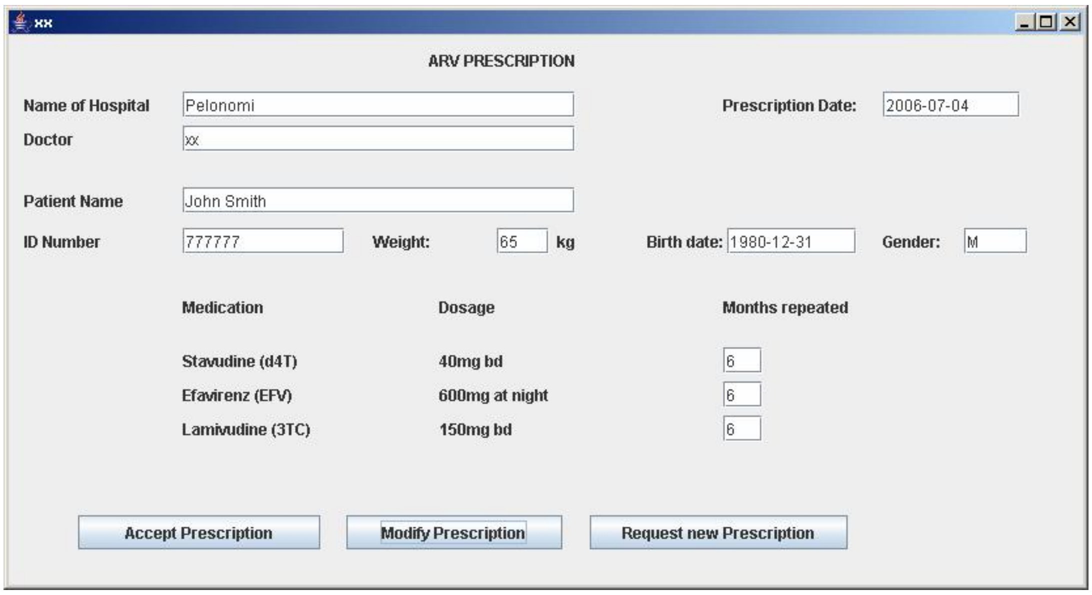
No Yes - how many times?

Reason(s): Hospitalisation 1: Hospitalisation 2:

< Previous Next >

Figure 5.19: Second interface for entering data during follow-up visit

The next step in this sequence of events is to revise the previous prescription if necessary. This is an important component of the doctor's task. The *PhysicianAgent* starts by contacting the *MDBAgent* to retrieve the current prescription, if available. The prescription is then displayed in the interface shown in Figure 5.20 below. If a prescription was not found, an empty form would be displayed.



xx

ARV PRESCRIPTION

Name of Hospital	Pelonomi
Doctor	xx
Prescription Date:	2006-07-04
Patient Name	John Smith
ID Number	777777
Weight:	65 kg
Birth date:	1980-12-31
Gender:	M

Medication	Dosage	Months repeated
Stavudine (d4T)	40mg bd	6
Efavirenz (EFV)	600mg at night	6
Lamivudine (3TC)	150mg bd	6

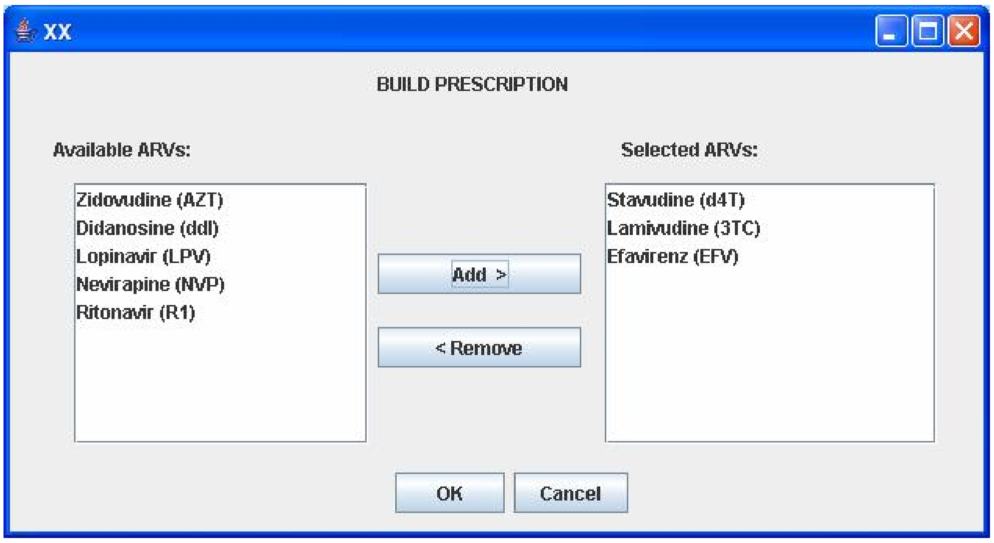
Accept Prescription Modify Prescription Request new Prescription

Figure 5.20: Current ARV Prescription

The doctor now has different options to obtain a new prescription. If the doctor selects option "*Request new Prescription*", the *PhysicianAgent* sends a message to the *MedicationAgent* to compile a new prescription. The *MedicationAgent* decides which drugs to use based on the physical and medical conditions of the patient. It replies with the new prescription, which is displayed in the ARV Prescription window.

The *MedicationAgent* retrieves information about the combination of drugs suited for the specific patient from the medication database via the *MDBAgent*. In the current system only the gender of the patient is used as the criterion for choosing a combination of drugs. In future this will be developed to include drug interactions and symptoms of the patient. Information about the doses of drugs is also retrieved from the medication database and a selection is made depending on the weight of the patient.

If the doctor is not satisfied with the rendered prescription, he can request another prescription from the *MedicationAgent*, or he could change the prescription by selecting "*Modify Prescription*" to modify the prescription manually. The available ARVs as well as the ARVs that are currently selected for the prescription are displayed as shown in Figure 5.21. The doctor may select one or more items to add or remove from the prescription. When the doctor is satisfied with the prescription and clicks "*OK*", the names of the selected drugs are displayed on the ARV Prescription form (Figure 5.20) with the appropriate dosage for the patient as retrieved from the medication database.


XX

BUILD PRESCRIPTION

Available ARVs:

Zidovudine (AZT)
Didanosine (ddI)
Lopinavir (LPV)
Nevirapine (NVP)
Ritonavir (R1)

Selected ARVs:

Stavudine (d4T)
Lamivudine (3TC)
Efavirenz (EFV)

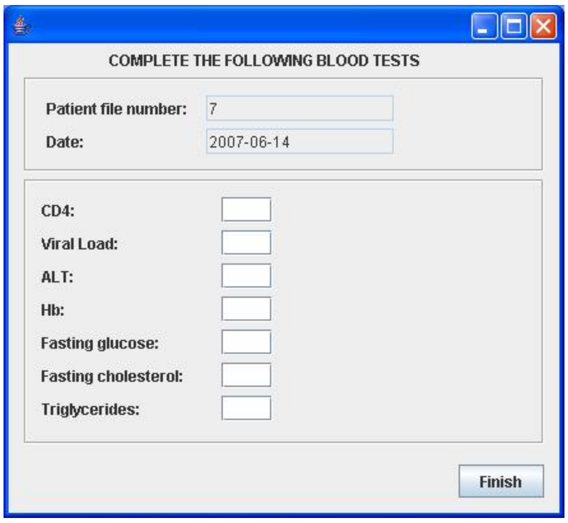
Figure 5.21: Build prescription

The default number of months that each drug is prescribed is set to six months, but the doctor has the option to change this value for each drug individually.

When the doctor is satisfied with the prescription, and clicks "*Accept Prescription*, the prescription information is sent to the *MedicationAgent*, which sends a request to the *MDBAgent* to save it on the database. A hard copy of the prescription can now be issued to the patient, and the prescription is available on the system to be accessed by the nurse or pharmacist when drug issues are done.

5.4.3 Lab Worker

When blood has been sent to the laboratory to be tested, the *LabAgent* receives the message, and the interface shown below in Figure 5.22 is displayed on the computer in the lab. When the blood tests have been completed, the lab worker enters the test results. When the lab worker clicks "*OK*", the *LabAgent* sends a message to the *PatientAgent* to add the results to the medical conditions of the patient.


COMPLETE THE FOLLOWING BLOOD TESTS

Patient file number: 7

Date: 2007-06-14

CD4:

Viral Load:

ALT:

Hb:

Fasting glucose:

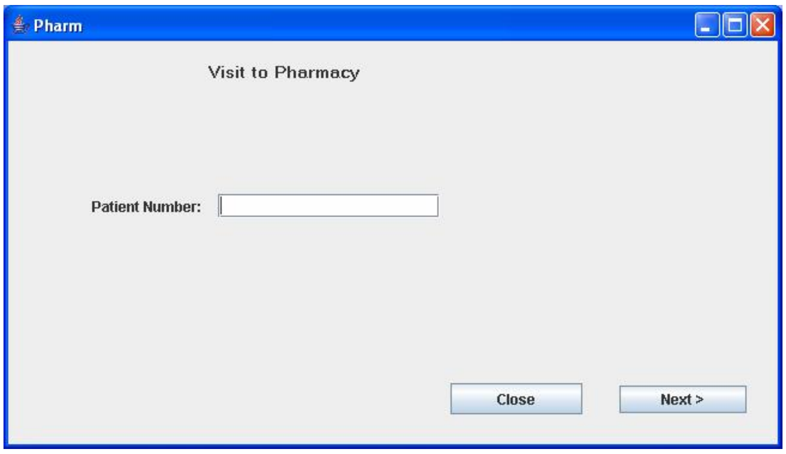
Fasting cholesterol:

Triglycerides:

Figure 5.22: Enter Results of Blood Tests

5.4.4 Pharmacist

When the *PharmacistAgent* is activated, the interface shown in Figure 5.23 is displayed. The pharmacist enters the patient number of the patient visiting the pharmacy for whom he has to issue the medicine as prescribed by a doctor. When the pharmacist clicks the "*Next*" button, the *PharmacistAgent* sends a request to the *PatientAgent* for the personal information of the patient. The *PharmacistAgent* also requests the information about the last prescription for this patient from the *MDBAgent.*

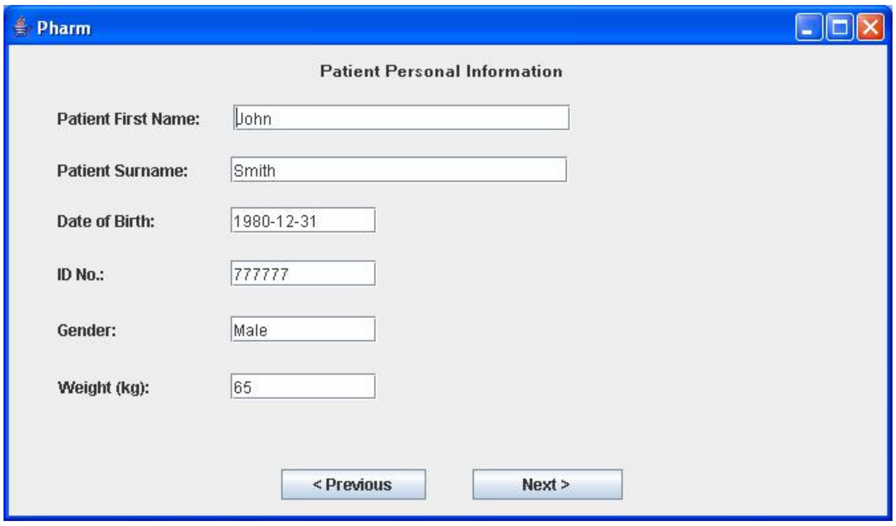

Pharm

Visit to Pharmacy

Patient Number:

Figure 5.23: Interface for Pharmacist to enter patient number

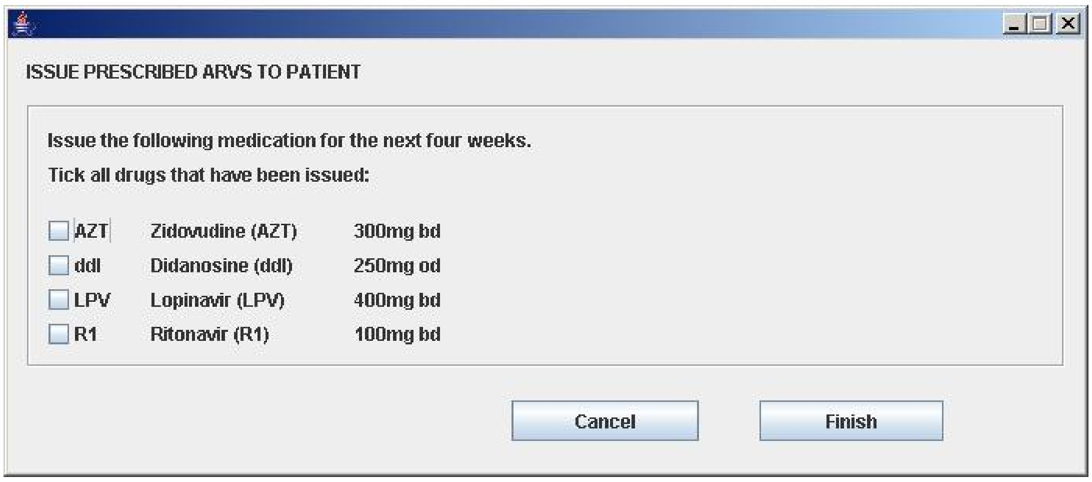
The personal information about the patient is displayed as shown in Figure 5.24. The pharmacist has to make sure if this is the correct patient by asking the patient for some form of identification, and will then click "*Next*" to continue.



Patient First Name:	John
Patient Surname:	Smith
Date of Birth:	1980-12-31
ID No.:	777777
Gender:	Male
Weight (kg):	65

Figure 5.24: Patient Personal Information

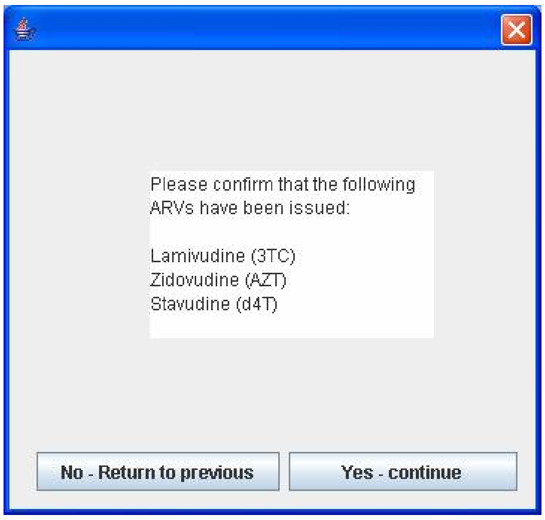
The interface showing the prescription is then displayed as shown in Figure 5.25. The pharmacist may select the ARVs that will be issued to the patient from the list of ARVs that are part of the prescription. When the pharmacist clicks "*Finish*", he will be asked to confirm whether these ARVs have been issued. This message is shown in Figure 5.26.


ISSUE PRESCRIBED ARVS TO PATIENT

Issue the following medication for the next four weeks.
Tick all drugs that have been issued:

☐ AZT	Zidovudine (AZT)	300mg bd
☐ ddI	Didanosine (ddI)	250mg od
☐ LPV	Lopinavir (LPV)	400mg bd
☐ R1	Ritonavir (R1)	100mg bd

Figure 5.25: Select ARVs to issue to patient


Please confirm that the following ARVs have been issued:

Lamivudine (3TC)
Zidovudine (AZT)
Stavudine (d4T)

Figure 5.26: Confirm that the correct ARVs were issued

When the pharmacist confirms the issue of ARVs, the *PharmacistAgent* sends the information about the issue to the *MDBAgent* to update the prescription information and the issue of drugs on the database.

5.5 The *MedAgent* Database Implementation

The database design described in Section 4.3.3 used with *MedAgent* was implemented using *MySQL*. The initial database design consisting of distributed databases, as described in Figure 4.1, was simplified and implemented as a single database because of time constraints. Some examples of the records in the different tables and the queries of the data as done by the agents are shown in this section.

5.5.1 New Records

New records are inserted into the tables of the database at different points in the system. Some examples of the creation of new records will be described in this section.

When a patient visits a nurse for the first time, the nurse gathers and enters the personal information of the patient. At the end of the visit the *NurseAgent* sends this information to the *PatientAgent* to update the database. The

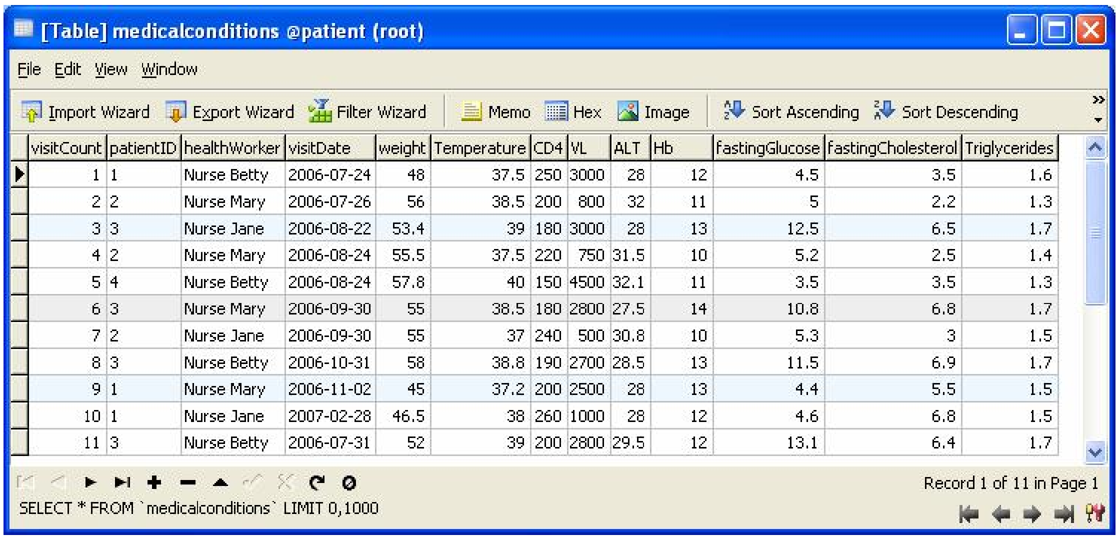
PatientAgent connects to the *MySQL* database and inserts a record into the *PatientPersonalInfo* table. After each visit to a nurse the medical conditions of a patient will also be added to the *MedicalConditions* table in a similar manner. The code that is used to build the query and execute the update to insert the record into the *MedicalConditions* table is given in Figure 5.27 below:

```
// Build MySQL query string
   String insertString = "INSERT INTO medicalconditions SET " +
   "patientID = '" + getAID().getLocalName() +
   "', healthWorker = '" + healthWorker + "', visitDate = '" +
   visitDate + "', weight = " + weight + ", Temperature = " +
   Temperature + ", CD4 = " + CD4 + ", VL = " + VL +", ALT = " +
   ALT + ", Hb = " + Hb + ", fastingGlucose = " + fastingGlucose +
   ", fastingCholesterol = " + fastingCholesterol +
   ", Triglycerides = " + Triglycerides;

// Execute update
   statement.executeUpdate(insertString);
```

Figure 5.27: Insert Record into *MedicalConditions* table

Some records of table *MedicalConditions* that were created in this way are shown in Figure 5.28 below:


[Table] medicalconditions @patient (root)
File Edit View Window
Import Wizard Export Wizard Filter Wizard Memo Hex Image Sort Ascending Sort Descending

visitCount	patientID	healthWorker	visitDate	weight	Temperature	CD4	VL	ALT	Hb	fastingGlucose	fastingCholesterol	Triglycerides
1	1	Nurse Betty	2006-07-24	48	37.5	250	3000	28	12	4.5	3.5	1.6
2	2	Nurse Mary	2006-07-26	56	38.5	200	800	32	11	5	2.2	1.3
3	3	Nurse Jane	2006-08-22	53.4	39	180	3000	28	13	12.5	6.5	1.7
4	2	Nurse Mary	2006-08-24	55.5	37.5	220	750	31.5	10	5.2	2.5	1.4
5	4	Nurse Betty	2006-08-24	57.8	40	150	4500	32.1	11	3.5	3.5	1.3
6	3	Nurse Mary	2006-09-30	55	38.5	180	2800	27.5	14	10.8	6.8	1.7
7	2	Nurse Jane	2006-09-30	55	37	240	500	30.8	10	5.3	3	1.5
8	3	Nurse Betty	2006-10-31	58	38.8	190	2700	28.5	13	11.5	6.9	1.7
9	1	Nurse Mary	2006-11-02	45	37.2	200	2500	28	13	4.4	5.5	1.5
10	1	Nurse Jane	2007-02-28	46.5	38	260	1000	28	12	4.6	6.8	1.5
11	3	Nurse Betty	2006-07-31	52	39	200	2800	29.5	12	13.1	6.4	1.7

Record 1 of 11 in Page 1

```
SELECT * FROM `medicalconditions` LIMIT 0,1000
```

Figure 5.28: Table *MedicalConditions*

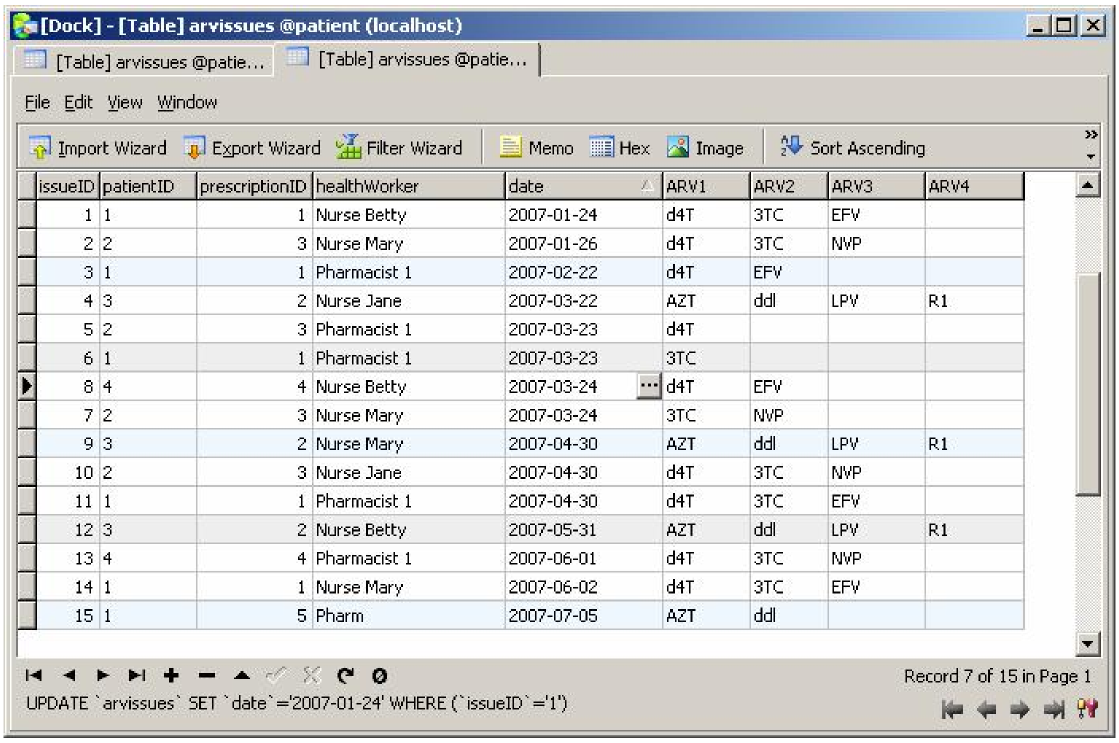
Every time when a nurse or pharmacist issues one or more ARVs to a patient, the *NurseAgent* or *PharmacistAgent* sends the information to the *MDBAgent,* which connects to the database and creates a new record in table *ARVIssues*. The code in Figure 5.29 is used to accomplish this task:

```
// Insert record into database
   String insertString = "INSERT INTO ARVIssues SET " +
   "patientID = '" + patientID +
   "', prescriptionID = " + prescriptionID +
   ", healthWorker = '" + healthWorker +
   "', date = '" + date + "', ARV1 = '" + ARV1 +
   "', ARV2 = '" + ARV2 + "', ARV3 = '" + ARV3 +
   "', ARV4 = '" + ARV4 + "'";

//Update database table
   statement.executeUpdate(insertString);
```

Figure 5.29: Insert Record into *ARVIssues* table

The content of table *ARVIssues* is shown in Figure 5.30 below:

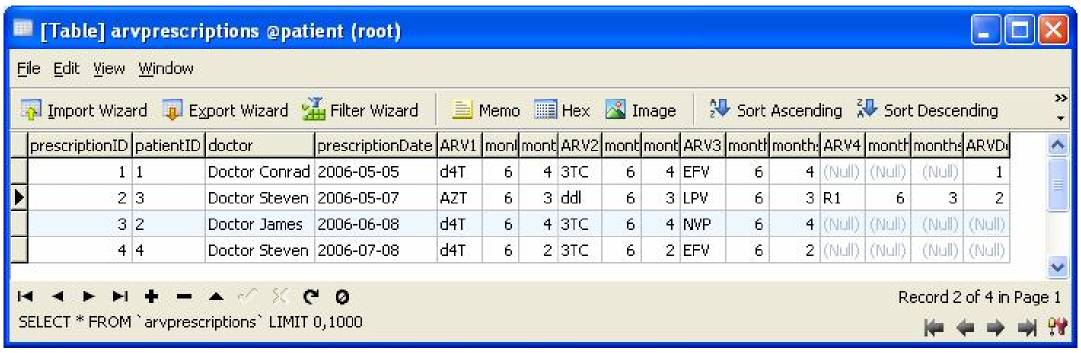


issueID	patientID	prescriptionID	healthWorker	date	ARV1	ARV2	ARV3	ARV4
1	1	1	Nurse Betty	2007-01-24	d4T	3TC	EFV	
2	2	3	Nurse Mary	2007-01-26	d4T	3TC	NVP	
3	1	1	Pharmacist 1	2007-02-22	d4T	EFV		
4	3	2	Nurse Jane	2007-03-22	AZT	ddl	LPV	R1
5	2	3	Pharmacist 1	2007-03-23	d4T			
6	1	1	Pharmacist 1	2007-03-23	3TC			
8	4	4	Nurse Betty	2007-03-24	d4T	EFV		
7	2	3	Nurse Mary	2007-03-24	3TC	NVP		
9	3	2	Nurse Mary	2007-04-30	AZT	ddl	LPV	R1
10	2	3	Nurse Jane	2007-04-30	d4T	3TC	NVP	
11	1	1	Pharmacist 1	2007-04-30	d4T	3TC	EFV	
12	3	2	Nurse Betty	2007-05-31	AZT	ddl	LPV	R1
13	4	4	Pharmacist 1	2007-06-01	d4T	3TC	NVP	
14	1	1	Nurse Mary	2007-06-02	d4T	3TC	EFV	
15	1	5	Pharm	2007-07-05	AZT	ddl		

```
UPDATE `arvissues` SET `date`='2007-01-24' WHERE (`issueID`='1')
```

Figure 5.30: Table *ARVIssues*

When the doctor finalises a prescription, the *MedicationAgent* sends the new prescription to the *MDBAgent,* which connects to the database and creates a new record in table *ARVPrescriptions*. Sample records are shown in Figure 5.31 below. A combination of up to four drugs can be used in one ARV prescription and for each drug the name of the medication, the number of months for which it has been prescribed, as well as the number of months for which the issues have been completed, is shown.



prescriptionID	patientID	doctor	prescriptionDate	ARV1	monl	mont	ARV2	mont	mont	ARV3	montl	month:	ARV4	montl	month:	ARVD
1	1	Doctor Conrad	2006-05-05	d4T	6	4	3TC	6	4	EFV	6	4	(Null)	(Null)	(Null)	1
2	3	Doctor Steven	2006-05-07	AZT	6	3	ddl	6	3	LPV	6	3	R1	6	3	2
3	2	Doctor James	2006-06-08	d4T	6	4	3TC	6	4	NVP	6	4	(Null)	(Null)	(Null)	(Null)
4	4	Doctor Steven	2006-07-08	d4T	6	2	3TC	6	2	EFV	6	2	(Null)	(Null)	(Null)	(Null)

Record 2 of 4 in Page 1

```
SELECT * FROM `arvprescriptions` LIMIT 0,1000
```

Figure 5.31: Table *ARVPrescriptions*

5.5.2 Update Queries

The database is updated at several occasions in the system when information is gathered and entered by a nurse, doctor, pharmacist or lab worker. Some examples of changes made to existing records are shown in this section.

When a nurse completes the examination of a patient, a record of the medical conditions is created as previously shown in Figures 5.27 and 5.28, but the blood drawn during that visit has to be sent to a laboratory to be tested before the results can be added to the record. After the laboratory worker has completed the tests, and entered the results, the *LabAgent* sends the results of the blood tests to the *PatientAgent*, which updates the record in table *MedicalConditions* that was created after the clinic visit, to include the blood test results.

The code in Figure 5.32 is used to generate and execute the update query:

```
statement = connection.createStatement();

// update database
    String updateStr = "UPDATE medicalconditions SET " +
      "CD4 = '" + CD4 + "', VL = '" + VL +
      "', ALT = '" + ALT + "', Hb = '" + Hb +
      "', fastingGlucose = '" + fastingGlucose +
      "', fastingCholesterol = '" + fastingCholesterol +
      "', Triglycerides = '" + Triglycerides +
      "' WHERE " +
      "(patientID = '" + getAID().getLocalName() +
      "') AND (visitCount = " + visit + ")";

    statement.executeUpdate(updateStr);
```

Figure 5.32: Update Query

The table ARVPrescriptons is updated every time after ARVs have been issued to reflect the number of issues of each drug completed. The following code is used to do this update:

```
// Generate query to retrieve information from database

    String x = "SELECT patientID, monthsARV1Issued, " +
    "monthsARV2Issued, monthsARV3Issued, monthsARV4Issued " +
    "FROM ARVPrescriptions WHERE prescriptionID = '" +
    prescriptionID + "'";

// Execute query to retrieve information from database

    ResultSet resultSet2 = statement.executeQuery(x);
        String patientID = "";

// Increment issue counts

        while ( resultSet2.next() ) {
            patientID = resultSet2.getString("patientID");
            monthsARV1Issued =

Integer.parseInt(resultSet2.getString("monthsARV1Issued"));
            if (ARV1 != "") {monthsARV1Issued += 1;}
            monthsARV2Issued =

Integer.parseInt(resultSet2.getString("monthsARV2Issued"));
            if (ARV2 != "") {monthsARV2Issued += 1;}
            monthsARV3Issued =

Integer.parseInt(resultSet2.getString("monthsARV3Issued"));
            if (ARV3 != "") {monthsARV3Issued += 1;}
            monthsARV4Issued =

Integer.parseInt(resultSet2.getString("monthsARV4Issued"));
            if (ARV4 != "") {monthsARV4Issued += 1;}
        }
```

```
// create Statement for updating database

    statement = connection.createStatement();

    String X = "UPDATE ARVPrescriptions SET " +
    "monthsARV1Issued = " + monthsARV1Issued +
    ", monthsARV2Issued = " + monthsARV2Issued +
    ", monthsARV3Issued = " + monthsARV3Issued +
    ", monthsARV4Issued = " + monthsARV4Issued +
    " WHERE " + "(prescriptionID = " + prescriptionID + ")";

// Update database
    statement.executeUpdate(X);
```

Figure 5.31: Table *ARVPrescriptions*

5.6 Messaging between Agents

Agents communicate by sending out messages to other agents and receiving messages from other agents. In the *MedAgent* system the *PatientAgent* is the only agent that has access to a patient's information on the database and the *MDBAgent* is the only one that has access to the medication database. The other agents that have user interfaces to gather information, or provide information, have to communicate with the former two agents to retrieve or save the information. The *MedicationAgent* also has to communicate with the *PhysicianAgent* to agree on the composition of a prescription. Some examples of the messaging between agents will be described in this section. The figures in this section are screenshots that were taken of the message tracking as gathered during programme execution and displayed graphically by the *JADE SnifferAgent.*

All messaging in the system are done via the FIPA messages REQUEST and INFORM. REQUEST messages are used in all cases where data is requested, and the reply is done using an INFORM message as seen in Figure 5.33.

5.6.1 Messaging when Visit to Doctor commences

When a visit to a doctor commences, the *PhysicianAgent* has to determine whether the patient exists on the system, and information about the patient, medical conditions and prescription as saved previously, have to be retrieved from the system. The messages sent between agents to accomplish this are shown in Figure 5.30 below. The agents that were tracked by the *SnifferAgent* are a *PhysicianAgent* (a), a *PatientAgent* (b), the *MDBAgent* (c) and the *AMSAgent* (d). The *PhysicianAgent* sends the first message (Request 0) to the *AMSAgent* to determine whether the *PatientAgent* exists on the system. An affirmative reply is sent back to the *PhysicianAgent.* The *PhysicianAgent* then sends a request to the *PatientAgent* (Request 1) for the personal information of the patient, another request (Request 2) to the *MDBAgent* for information about the previous prescription issued for the patient, and finally a request to the *PatientAgent* (Request 3) for the medical conditions and blood test results of the patient. The results of Requests 0, 1, 2 and 3 do not depend on one another, and the reply messages to these requests may be sent back to the *PhysicianAgent* in any order as seen in Figure 5.33.

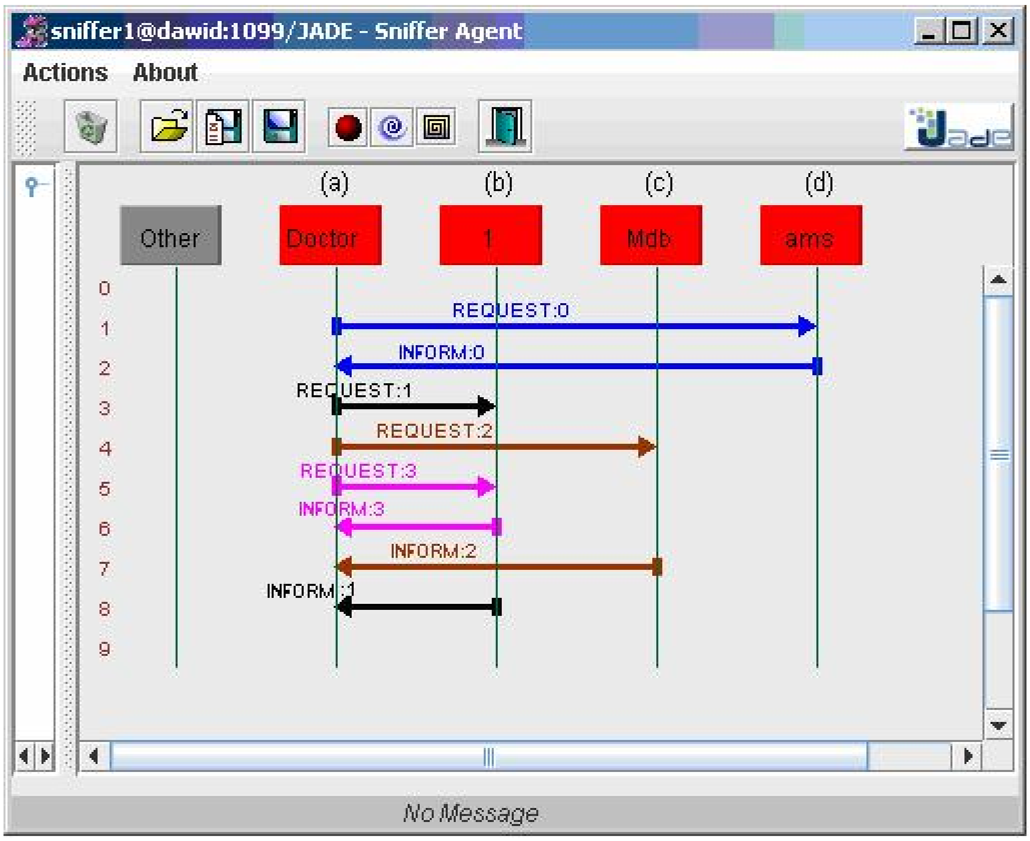


Figure 5.33: Messaging at the beginning of a visit to a doctor

A conversation ID as well as the receiver is set for each message that is sent to identify the type of message and the agent that has to respond to it. Other information when requesting data about a patient includes the ID of the patient. In the case of a request about a prescription, the prescription ID is sent to the *MDBAgent*. The reply is sent to the agent that initiated the request, and contains the data requested in that specific instance.

5.6.2 Messaging during the Prescription Process

When the doctor requests the generation of a prescription by the system, the different steps of the prescription process have to be completed in a particular order. First the patient data has to be obtained, then the medications suitable for the patient have to be retrieved and finally, the prescription is issued. The messages sent between the agents involved in the prescription process are shown in Figure 5.34 below.

The process starts when a *PhysicianAgent* (a), sends a request (Request 0) for a prescription to the *MedicationAgent* (c). The *MedicationAgent* sends a message to the *PatientAgent* (b) to request the personal and medical information of the patient (Request 1). It waits for the information from the *PatientAgent* before sending a request to the *MDBAgent* for all possible drugs suitable for the patient (Request 2). The requests, as well as the replies from the *MDBAgent* and the *PatientAgent* to the *MedicationAgent* containing the suitable information, can be seen in Figure 5.34. Finally when the *MedicationAgent* has selected a combination of drugs and the generation of the prescription is completed, the *MedicationAgent* sends the prescription to the *PhysicianAgent* in reply to the first request.

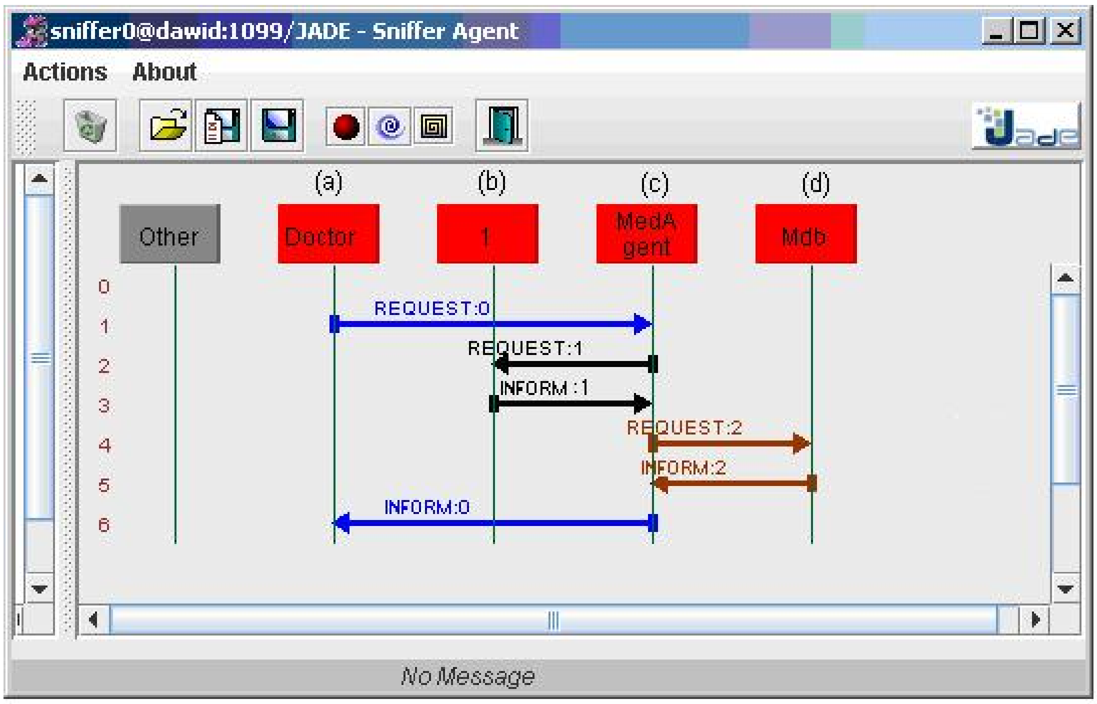


Figure 5.34: Messages sent during the prescription process

5.6.3 Messaging during Drug Issue

The messages sent between agents when drugs are issued by a pharmacist are shown in Figure 5.35. The three agents involved are a *PharmacistAgent* (a), a *PatientAgent* (b) and the *MDBAgent* (c). The *PharmacistAgent* sends a request (Request 0) to the *PatientAgent* to retrieve the personal information of the patient for identification purposes. The *PharmacistAgent* also requests the latest prescription for this patient from the *MDBAgent* (Request 1). When both reply messages have been received, the pharmacist may issue the prescribed drugs. The *PharmacistAgent* completes the process by sending the names of the drugs that were issued to the *MDBAgent* (Request 2) to update the prescription and issues done. The *MDBAgent* replies to confirm that the update has been completed.

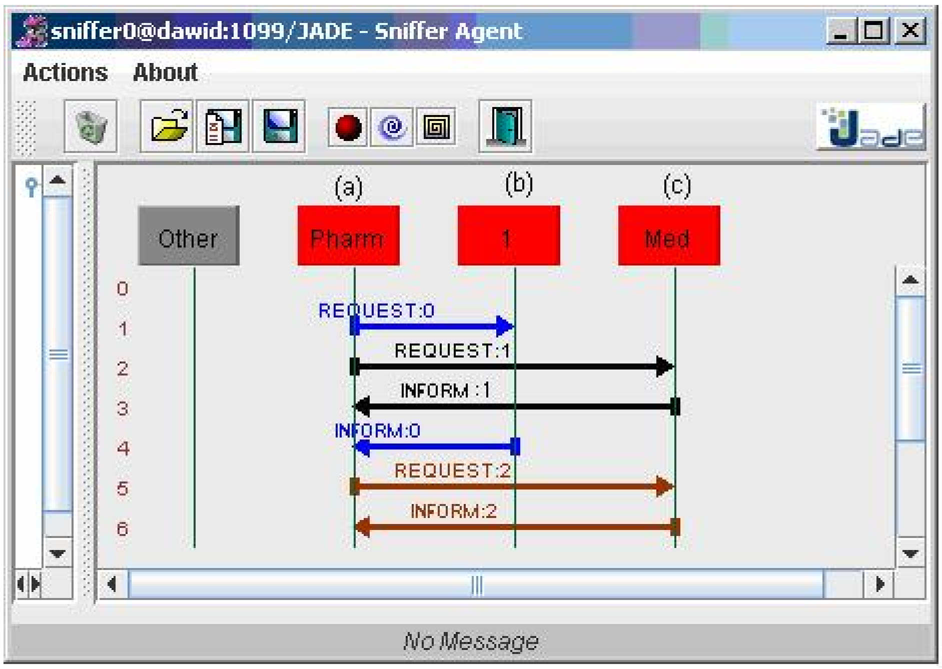


Figure 5.35: Messages sent during a drug issue

C h a p t e r 6

6 CONCLUSIONS AND FURTHER WORK

6.1 Introduction

The main goal of this research was to develop a *MAS* for administering the prescription of anti-retroviral and anti-TB drugs, assisting health care staff in carrying out the complex task of combining antiretroviral and anti-TB drugs efficiently. To accomplish this goal, the following objectives were part of the research:

- Study intelligent agents and their application in different fields
- Design a multi-agent system for the prescription of anti-retroviral drugs.
- Implement the system and show how the system is used.

The extent to which these objectives were accomplished is highlighted in Section 6.2 below.

The use of agents makes it possible to create software systems that consist of components that are autonomous software systems working independently, but also communicating with one another. The concept of Multi-Agent Systems and the characteristics of intelligent agents, as well as application of an Agent-Oriented Software Engineering methodology that supports the development of *MAS*, were all part of the study.

One way of using agents is to have them act on the behalf of a user as user agents or personal agents. The field that was of particular interest in this study was the medical and health care domain. The health care domain is a vast open environment where professionals with different skills and roles communicate and co-operate. This domain is very well suited for the use of *MASs*. Medical *MASs* include distributed patient scheduling within a hospital, organ and tissue transplant management, community care, information access, decision support systems, training and drug prescriptions.

In this study the prescription of anti-retroviral and anti-TB drugs as an area where a MAS system could be deployed was identified. A brief summary of the achievements and shortcomings of the work is given in Section 6.2 below, and directions for possible future work are outlined in Section 6.3.

6.2 Summary

Different *AOSE* methodologies that aid in the development of *MAS* were considered and the *MaSE* methodology supported by a software engineering tool called *agentTool* was chosen for the analysis and design of the system.

In the analysis phase the system requirements were used to capture the goals of the system, then use cases were applied and the different roles were defined. In the design phase agent classes were created, the conversations between agents were constructed and then the agent classes were defined in more detail. Finally a deployment diagram was drawn to show the system structure.

The MaSE methodology was very helpful, because the different stages are well defined and it leads the user through the analysis phase and the design phase. The diagrams created by agentTool are easy to interpret and to modify. The whole process is iterative and if a change is made in one stage, it is reflected in the following stages.

All stages from the original requirements until the final implementation can be visualized, including the agents, their roles and the communication that takes place among them.

The database was designed to contain information about patients, health workers, health care centres, medications, prescriptions and issues done to patients. Relationships between the different tables were described.

The *JADE* platform was selected for the implementation of the system, because it provides a runtime environment where agents can "live", as well as a library of classes that can be used in application development. It also contains a set of graphical tools that supports the debugging and deployment phases. The software framework is fully implemented in the *Java* programming language and the agents are also implemented in *Java*.

All communication and interaction between agents have been done through the exchange of *JADE* messages, and all actions of agents have been implemented as *JADE* behaviours. The implementation of the database has been done in *MySQL*.

The database design having a distributed database located on a number of computers, was not implemented in the current system, but it was simplified and implemented as a central database. This was done because of time constraints.

6.3 Strengths and Weaknesses of the System

The system that was implemented is a good demonstration of the use of agents in *MAS*. All agents that were identified in the design process were implemented, except the counsellor agent. The behaviours of the different agents to execute their actions and accomplish their own personal goals, as well as the communication between agents via messages, have been implemented.

The system can accomplish the necessary actions that take place during a patient's visit to a nurse or doctor. A new patient can be added to the system, and the personal and medical information of the patient can be captured by the nurse or the physician to be stored by the *PatientAgent*. For an existing patient, the *PatientAgent* can access the information of that patient and make it available to the other agents. These agents may be on the same site or they may be located at another hospital or clinic. This is a great improvement on the current system, which is mainly paper-based and where all information has to be gathered again when a patient moves to another location.

Collaboration takes place between the different agents in the system to determine whether a prescription for ARVs has to be compiled or changed. The patient agent provides the personal information, whereas the medication database agent provides information about the medication that is suitable for that patient. The medication agent combines the information and compiles the prescription, which can be changed and has to be confirmed by the doctor before it is finalised. Although the decision-making process of the system can still be refined, in many cases the prescription that is compiled by the medication agent can be used unchanged. A hard copy of the prescription can be given to the patient, and the pharmacist also has access to the information on the system.

The decision-making process used by the medication agent in the compilation of prescriptions is not very sophisticated at this stage. It only distinguishes between three different predefined treatment regimes for ARVs: one for males, one for females and a third for persons for whom the previous treatment was not successful. The doctor can afterwards add or remove components from the prescription to adapt it for patients with special needs. The prescription of anti-TB drugs is not part of the current system. These three ARV regimes are the only prescriptions used in most public hospitals in the country, while TB treatment is done separately by special clinics. To improve the process of compiling more personalised prescriptions automatically and to include the prescription of anti-TB drugs, more data about drugs and symptoms has to be incorporated into the system. To do this, the involvement of specialists in the field of ARVs and anti-TB drugs will be needed, who have more insight regarding the pharmacological effects of the different drugs, the interaction between drugs and the side effects caused by these drugs.

At the moment the system is activated by means of a command line instruction that initialises the *JADE* platform and creates a number of agents. The agents reside on the system all the time and cannot be deleted from the system or added to the system, except via the *JADE* graphical user interface. The only exception is the patient agent, where new instances of the agent can be created by the nurse agent or physician agent when a patient visits the clinic or hospital for the first time.

6.4 Future Work

The system has not been tested in a working environment and some actions will be changed or improved during the testing phase. Security and privacy issues also have to be addressed. An authentication process must be included before the system can be set into operation.

Future work will include the implementation of the system distributed over different computers. The database implementation will also be be changed from a single database to distributed databases located on a number of hosts.

In the current system the information that is gathered for patients includes TB symptoms and a history of TB treatment, but in the compilation of prescriptions it only includes the prescription of ARVs, and not TB medication. This is to conform with the treatment of Aids patients in public hospitals in South Africa, where these two conditions are treated separately with little coordination. The system could be improved to include the prescription of TB drugs and to coordinate the treatment of the two diseases, which will be a vast improvement on the status quo.

The decision-making of the system in the compilation of prescriptions can also be improved. More information about pharmacological effects, drug interaction and side effects can be combined with symptoms experienced by patients to create better and more personalised prescriptions. A further possible improvement to the system will be to make it more adaptive, so that it can incorporate new information as research in this field is done and more drugs become available. It is also possible to add a learning ability to the system to make it more adaptive and intelligent.

7 REFERENCES

Alsinet, T., Ansótegui, C., Béjar, R., Fernández, C. and Manyà, F. (2003). Automated monitoring of medical protocols: a secure and distributed architecture. Artificial Intelligence in Medicine 27 (2003): 367-92.

Alsinet, T., Béjar, R., Ansótegui, C., Fernàndez, C. and Manyà, F. (1998). JAFDIS, a Java framework for dialogical institution specification. Technical Report DIEI-98-RT-2, Universitat de Lleida. Also available at http://fermat.eup.udl.es/~cesar/recerca/jafdis_ev.ps.gz, Accessed on 12 August 2005.

AOL Shopping Main - Online Shopping Made Easy, http://shopping.aol.com/ (2007), Accessed on 10 July 2007

Barro, S., Presedo, J., Castro, D., Fernandez Delgado, M., Fraga, S., Lama, M. and Vila, J. (1999). Intelligent telemonitoring of critical-care patients. IEEE Engineering in Medicine and Biology Magazine, Jul-Aug; 18 (4), 80-88.

Bauer, B., Müller, J.P., Odell, J. 2001. Agent UML: A Formalism for Specifying Multiagent Interaction, *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag, Berlin, pp. 91-103, 2001.

Baujard, O., Baujard, V., Aurel, S., Boyer, C. and Appel, R.D. (1998). MARVIN, a multi-agent softbot to retrieve multilingual medical information on the Web. *Medical Informatics* 23 (3), Taylor & Francis, London, 187-191.

Beer, M.D., Huang, W. and Sixsmith, A. (2002). Using agents to build a practical implementation of the INCA-Intelligent Community Alarm- system. In: Jain, L.C., Chen, Z. and Ichalkaranje, N. (eds.): *Intelligent Agents and their applications. Studies in Fuzzines and Soft Computing,* Physica-Verlag, Berlin, 320-345.

Bond, A.H. and Gasser, L. (1988) An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, Readings in Distributed Artificial Intelligence, pages 3–36. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

BookFinder.com: Search for New & Used Books, Textbooks, Out-of-Print and Rare Books, http://www.bookfinder.com/ (2007), Accessed on 10 July 2007

Bordini R.H., Dastani, M., Dix, J., and El Fallah Seghrouchni, A., editors. Multi-Agent Programming: Languages, Platforms and Applications. Number 15 in *Multiagent Systems, Artificial Societies, and Simulated Organizations.* Springer, 2005.

Boyer, C., Baujard, O., Baujard, V., Aurel, S., Selby, M. and Appel, R.D., (1997). Health on the Net automated database of health and medical information. International Journal of Medical Informatics, Volume 47, Number 1, November 1997 , pp. 27-29(3)

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. 2004. TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers 8(3), pp. 203 – 236.

Camarinha-Matos, L.M. and Vieira, W. (1999). Intelligent mobile agents in elderly care, *Journal of Robotics and Autonomous Systems (Elsevier)*, Vol. 27, N. 1-2, April 1999, ISBN 0921-8890, pp. 59-75

Chavez, A. and Maes, P. (1996). Kasbah: An Agent Marketplace for Buying and Selling Goods, Proceedings of the First Intern. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology. London, UK, April 1996.

Cossentino, M. (2005). From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B. and Giorgini, P., editors (2005). Agent-Oriented Methodologies. Idea Group Publishing., Chapter IV, pp. 79—106.

Decker, K. and Li, J. (1998). Coordinated hospital patient scheduling. In *Proceedings of the Third International Conference on Multi-Agent Systems*, ICMAS-98. p. 104-111., Paris, France.

DeLoach, S.A. (2001). Analysis and Design using *MaSE* and agentTool. In: Proceedings of the *12th Midwest Artificial Intelligence and Cognitive Science Conference*, March 2001. Miami University, Oxford, Ohio.

DeLoach, S.A. and Wood, M. (2000) Developing multiagent systems with agentTool. In Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop (ATAL 2000), C. Castelfranchi, Y. Lesperance (Eds.). *Lecture Notes in Computer Science*. Vol. 1986, Springer Verlag, Berlin, 2001. 1429–1436, 2001.

DeLoach, SA., Wood, M.F., Sparkman, C.H. (2001) Multiagent systems engineering. In *The International Journal of Software Engineering and Knowledge Engineering,* Vol. 11, No. 3, 231-258, 2001.

Durfee, E.H. and Lesser, V. (1989). Negotiating task decomposition and allocation using partial global planning. In L. Gasser & M. Huhns, Distributed artificial intelligence (Volume II) (pp. 229 – 244). London/San Mateo, CA: Pitman Publishing/Morgan Kaufmann, 229-224.

Faltings, B. (Ed.), (2000) Intelligent Agents: Software Technology for the new Millennium, INFORMATIK 1/2000

Ferguson I.A., and Wooldridge, M.J. (1997). Paying Their Way: Commercial Digital Libraries for the 21st Century in D-lib magazine, June 1997, ISSN 1082-9873

FIPA, http://www.fipa.org (2005), Accessed on 12 August 2005.

Godo L., Puyol-Gruart, J., Sabater, J., Torra, V., Barrufet, P. and Fàbregas, X. (2003). A multi-agent system approach for monitoring the prescription of restricted use antibiotics. *Artificial Intelligence in Medicine* Volume 27, Issue 3, Pages 259-282.

Hayes-Roth B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., and Selves', A. (1992). Guardian: A Prototype Intelligent Agent for Intensive-Care Monitoring. *Artificial Intelligence in Medicine*, Vol. 4, No. 2, Mar. 1992, pp. 165-185.

Hospers, M., Kroezen, E., Nijholt, A., Op den Akker, H.J.A., Heylen, D.K.J. (2003) *An Agent-based Intelligent Tutoring System for Nurse Education.* Technical Report TR-CTIT-03-18 Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625

Hsu, C-C. and Ho, C-S. (1999). Acquiring patient data by an intelligent interface agent with medicine-related common sense reasoning. *Expert Systems with Applications*, Volume 17, Issue 4, 257-274.

JADE, http://*JADE*.cselt.it/ (2005), Accessed on 12 August 2005.

Jennings, N.R. and Wooldridge, M. (1998). Applications of Intelligent Agents in Agent Technology: Foundations, Applications and Markets (eds. N.R. Jennings and M. Wooldridge) Queen Mary & Westfield College, University of London. 3-28.

Jennings, N.R., Faratin, P., Norman, T.J., O'Brien, P., Wiegand, M.E., Voudouris, C., Alty, J.L., Miah, T., Mamdani, E.H. (1996) ADEPT: Managing Business Processes using Intelligent Agents. Proceedings of the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems, 1996.

Jennings, N.R., Sycara, K., Wooldridge, M. (1998) A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems,* 1(1): 7-38, 1998.

Juan, T., Pearce, A., and Sterling, L. (2002). ROADMAP: Extending the *Gaia* Methodology for Complex Open Systems. (International Conference on Autonomous Agents, Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), Bologna, Italy, Pages: 3 – 10, July 2002

Kostkova, P., Mani-Saada, J. and Weinberg, J. (2002). Agent-based up-to-date data management in the National Electronic Library for Communicable Disease. In *Proceedings of the Workshop on Agent Applications in Health Care, at the 15th European Conference on Artificial Intelligence*, ECAI 2002. Eds: U. Cortés, J. Fox, A. Moreno. Lyon, France, 59-63.

Kumar, A.D., Kumar, A.R., Kekre, S., Prietula, M.J. and Ow, P.S. (1989). Multi-agent systems and organizational structure: the support of hospital patient scheduling. In: Proceedings of the 3rd International Conference on Expert Systems and the Leading Edge in Production and Operations Management, South Carolina, USA, (1989) 551-566.

Lanzola, G., Gatti, L., Falasconi, S. and Stefanelli, M. (1999) A framework for building cooperative software agents in medical applications. *Artificial Intelligence in Medicine*, vol. 16, no. 3, pp. 223 – 249.

Lieberman, H. and Mason, C. (2002) Intelligent Agent Software for Medicine in *Future of Health Technology*, Renata Bushko, ed., IOS Press, Amsterdam, 2002.

Lieberman, H., (1995), Letizia: An Agent That Assists Web Browsing, International Joint Conference on Artificial Intelligence IJCAI-95, Montréal, August 1995.

Lieberman, H., Rosenzweig, E. and Singh, P. (2001) Aria: An Agent For Annotating And Retrieving Images, IEEE Computer, July 2001, pp. 57-61.

List of User-Agents (Spiders, Robots, Crawler, Browser), http://www.user-agents.org/ (2007), Accessed on 20 January 2007.

Ljungberg, M. and Lucas, A. (1992) The OASIS Air Traffic Management System. Proc. of the Second Pacific Rim International Conference on Artificial Intelligence (1992).

Mabry, S. L., Schneringer, T., Etters, T. and Edwards, N. (2003). Intelligent Agents for Patient Monitoring and Diagnostics, ACM International Symposium on Applied Computing (SAC 2003), Melbourne, FL.

Maes, P. (1994). Agents that Reduce Work and Information Overload, Communications of the ACM, 37(7).

Marinagi, C., Spyropoulos, C.D., Papatheodorou, C., Kokkotos, S. (2001) Continual Planning and scheduling for managing patient tests in hospital laboratories, *Artificial Intelligence in Medicine* vol. 20, no. 2, pp. 139-154.

Miksch, S., Cheng, K. and Hayes-Roth, B. (1996). The patient advocate: a cooperative agent to support patient-centered needs and demands., *Proc AMIA Annu Fall Symp. 1996*, 244-8.

Moreno, A. and Isern, D. (2002). Accessing distributed health-care services through smart agents. Proceedings of the 4th IEEE International Workshop on Enterprise Networking and Computing in the Health Care Industry (HealthCom 2002), Nancy, France, 34-41.

Moreno, A., Medical Applications of Multi-Agent Systems, (2003) AIME, http://cyber.felk.cvut.cz/EUNITE03-BIO/pdf/Moreno.pdf , Accessed on 21 March 2006.

mySimon – Price Comparison shopping (2007), http://www.mysimon.com/, Accessed on 10 July 2007.

Nealon, J.L. and Moreno, A. (2004). Agent-Based Applications in Health Care. In: e-Health: Applications of Computing Science in Medicine and Health Care. Research in Computing Science, 5. Instituto Politecnico Nacional Centro de Investigacion en Computacion.

Nwana, H.S. and Ndumu, D.T., (1999) A Perspective on Software Agents Research. The Knowledge Engineering Review Volume 14, Issue 2, September 1999, Pages: 125 – 142.

Object Management Group - Unified Modeling Language (2007), http://www.uml.org/, Accessed on 30 January 2007.

OpenClinical: knowledge management technologies for healthcare: Software Agents (2007), http://openclinical.org/agents.html, Accessed on 20 January 2007.

Parunak, H.V.D. (1987). Manufacturing experience with the contract net, in: M. N. Huhns, ed., Distributed Artificial Intelligence, Morgan Kaufmann Publishers, 285-310.

Rialle, V., Lamy, J.-B., Noury, N. and Bajolle, L. Telemonitoring of patients at home: A Software Agent approach. Computer methods and programs in Biomedicine, pages 257-268, 72 (3) 2003.

Shoham, Y., (1997). An Overview of Agent-oriented Programming, ed. J.M. Bradshaw, *Software Agents*, AAAI Press, Menlo Park, California.

Shopping Bots (2007), http://ecommerce.hostip.info/pages/938/Shopping-Bots.html, Accessed on 10 July 2007.

Sycara, K., Paolucci, M., Van Velsen, M., Giampapa, J.A. (2003). The RETSINA MAS Infrastructure. Autonomous Agents and Multi-Agent Systems, Vol. 7, No. 1/2, July, 2003, pp. 29 – 48.

The Sims – Official Site (2006), http://thesims.ea.com, Accessed on 10 July 2007.

UN Chronicle: WHO Report 2005: TB linked to HIV at alarming Levels in Africa (2005), http://www.un.org/Pubs/chronicle/2005/issue2/0205p17.html,. Accessed on 18 August 2005.

Vázquez-Salceda, J., Padget, J.A., Cortés, U., López-Navidad, A. and Caballero, F. (2003). Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, Volume 27, Issue 3, 233-258.

Vicari, R.M., Flores, C.D., Silvestre, A.M., Seixas, L.J., Ladeira, M. and Coelho, H. (2003). A multi-agent intelligent environment for medical knowledge. *Artificial Intelligence in Medicine*, Volume 27, Issue 3, March 2003, 335-366.

Weiss G. (2002). Agent orientation in software engineering. *Knowledge Engineering Review*, January 2002.

Wood, M. (2000). Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems, M.Sc. Thesis.

Wooldridge, M. and Decker, K., 2000. Agents on the Net. IEEE Internet Computing, http://computer.org/internet/

Wooldridge, M.J. (2002). *Introduction to Multiagent Systems,* John Wiley & Sons Inc February 2002. ISBN 0 47149691X.

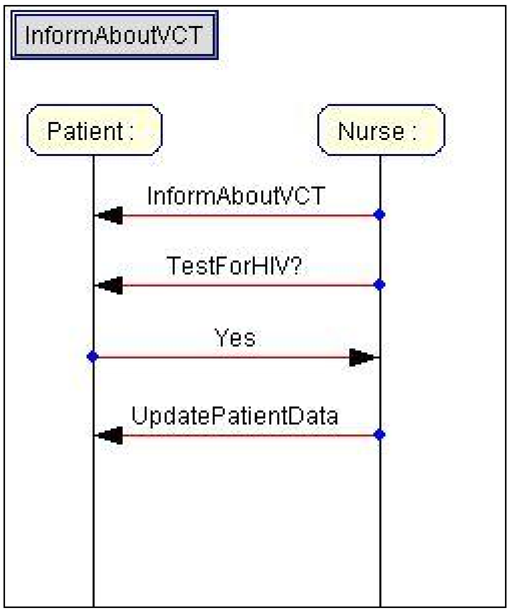
Wooldridge. M. and Jennings, N.R. (1995). "Agent Theories, Architectures, and Languages: a Survey", in Wooldridge and Jennings Eds., *Intelligent Agents*, pp. 1-22, Berlin: Springer-Verlag.

World Health Organization: HIV/AIDS topical information (2007), http://www.who.int/hiv/topics/en/, Accessed on 18 August 2007.

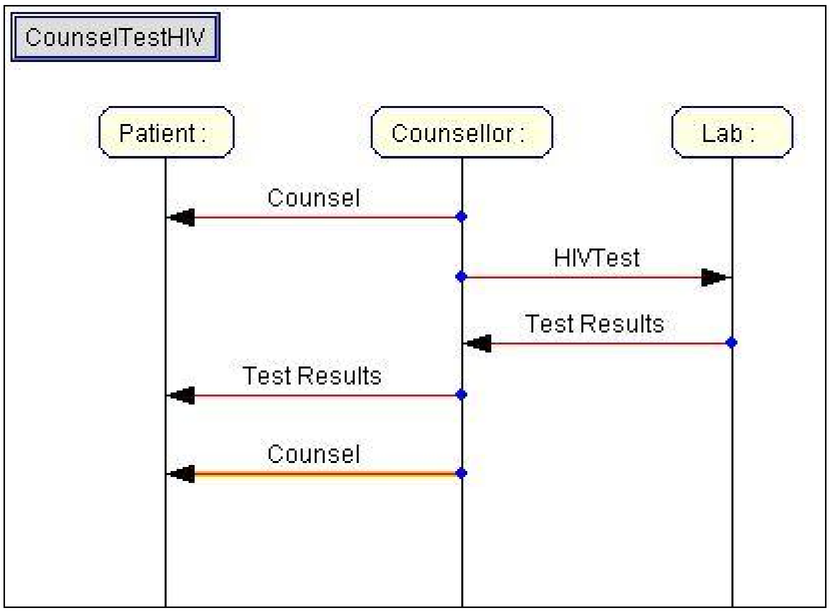
Yahoo! Shopping, http://shopping.yahoo.com/ (2007), Accessed on 10 July 2007

Zambonelli, F., Jennings, N.R., Omicini, A. and Wooldridge, M. (2003). Developing multiagent systems: The *Gaia* methodology. *ACM Transactions on Software Engineering and Methodologies (TOSEM),* Vol. 12, No. 3, July 2003, Pages 317 – 370.

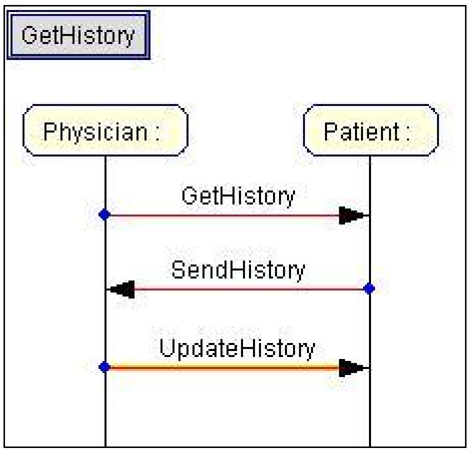
8 Appendix A: Sequence Diagrams



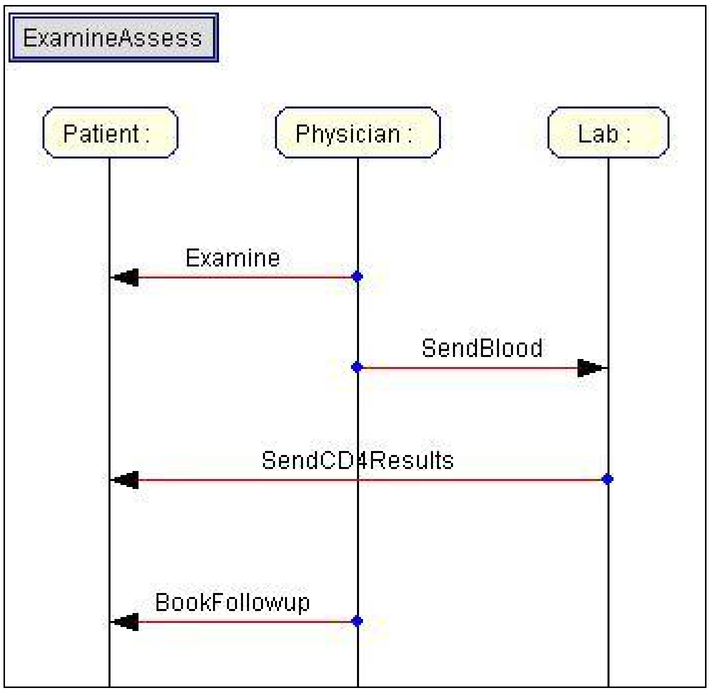
Inform About VCT



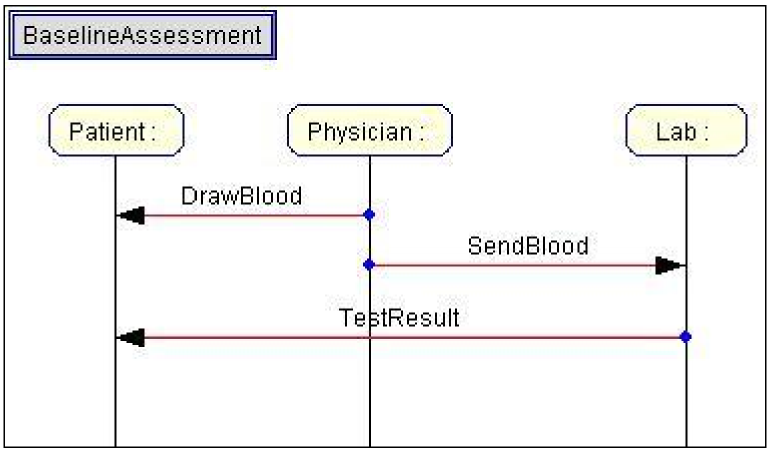
Counsel and Test for HIV



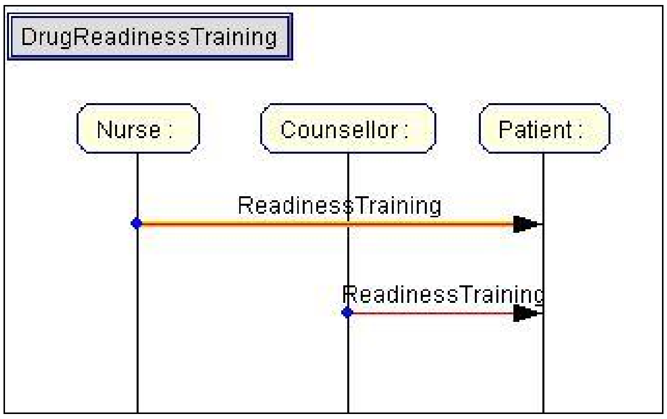
Get History



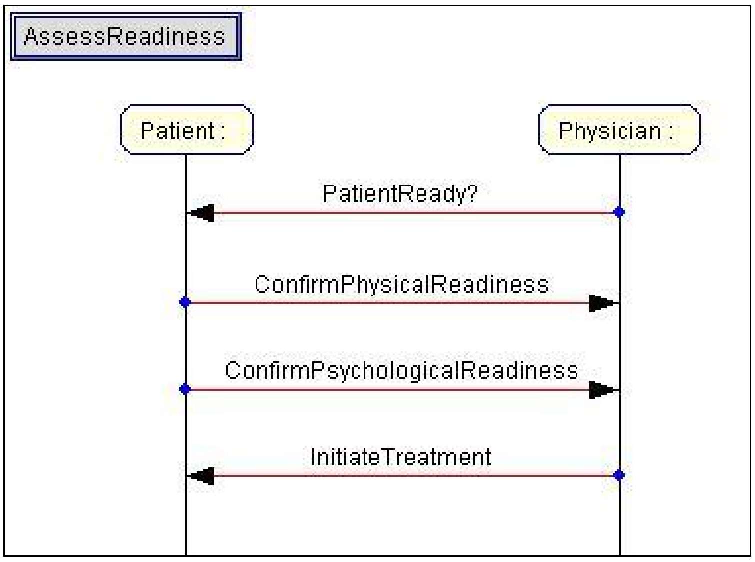
Examine and Assess



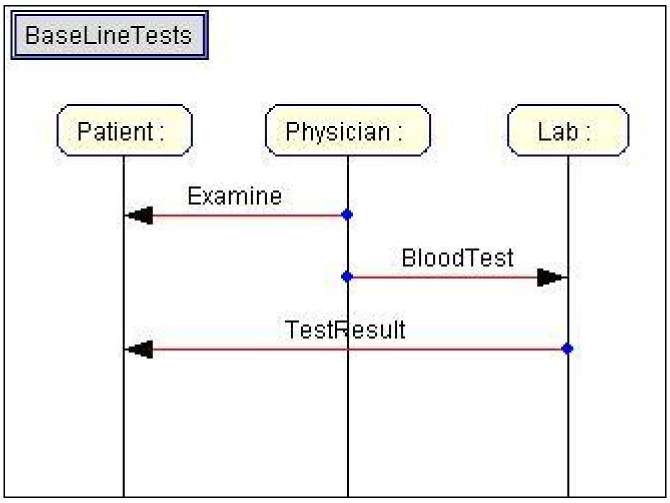
Baseline Assessment



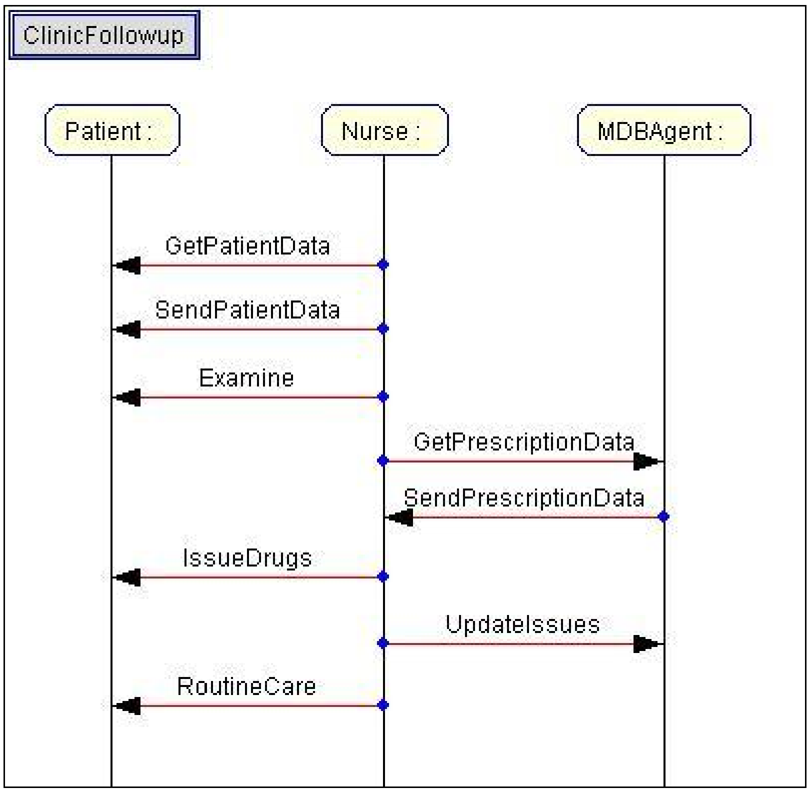
Drug Readiness Training



Assess Readiness

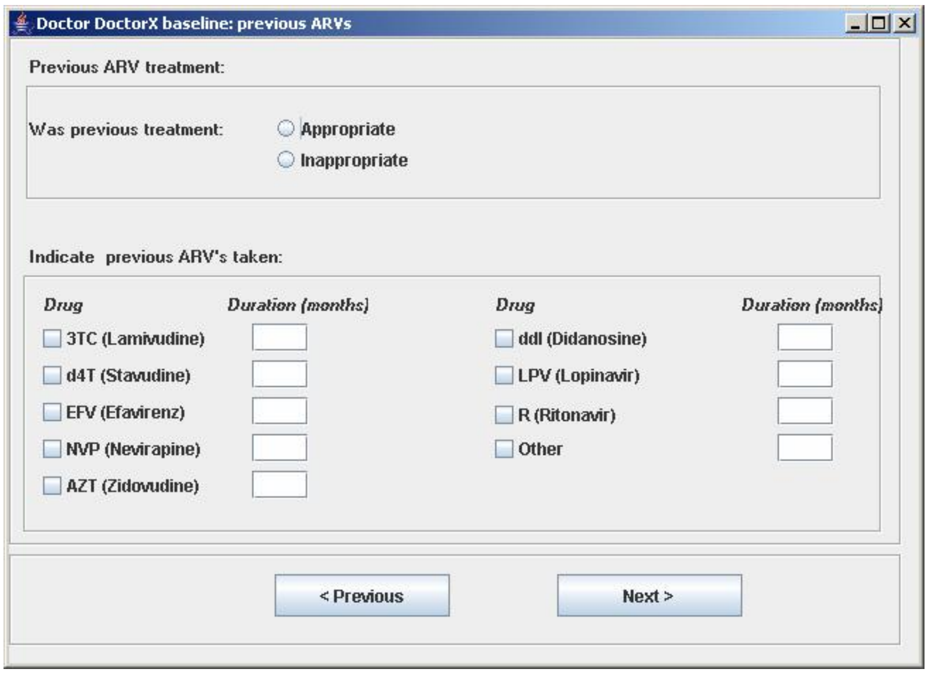


Baseline Viral Loads



Clinic Follow-Up

9 Appendix B: Graphical User Interfaces



Doctor DoctorX baseline: previous ARVs

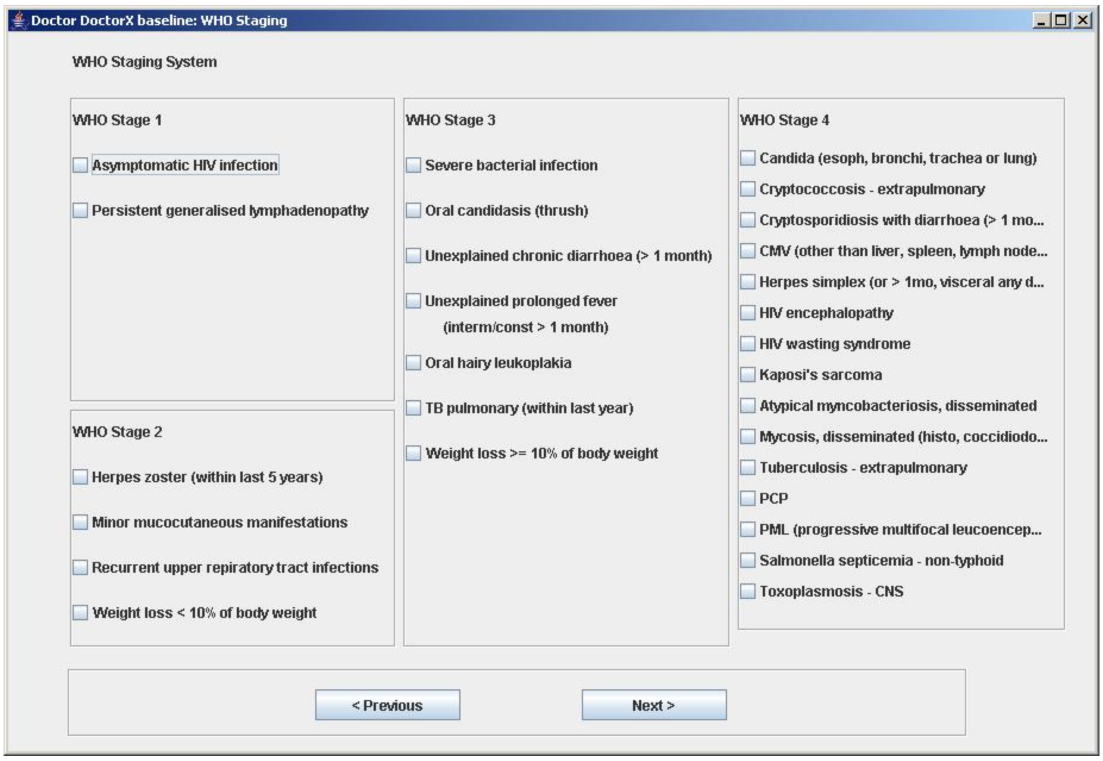
Previous ARV treatment:

Was previous treatment: ◯ Appropriate ◯ Inappropriate

Indicate previous ARV's taken:

Drug	Duration (months)	Drug	Duration (months)
☐ 3TC (Lamivudine)		☐ ddI (Didanosine)	
☐ d4T (Stavudine)		☐ LPV (Lopinavir)	
☐ EFV (Efavirenz)		☐ R (Ritonavir)	
☐ NVP (Nevirapine)		☐ Other	
☐ AZT (Zidovudine)			

Doctor Baseline Test Screen 1



Doctor DoctorX baseline: WHO Staging

WHO Staging System

WHO Stage 1
- ☐ Asymptomatic HIV infection
- ☐ Persistent generalised lymphadenopathy

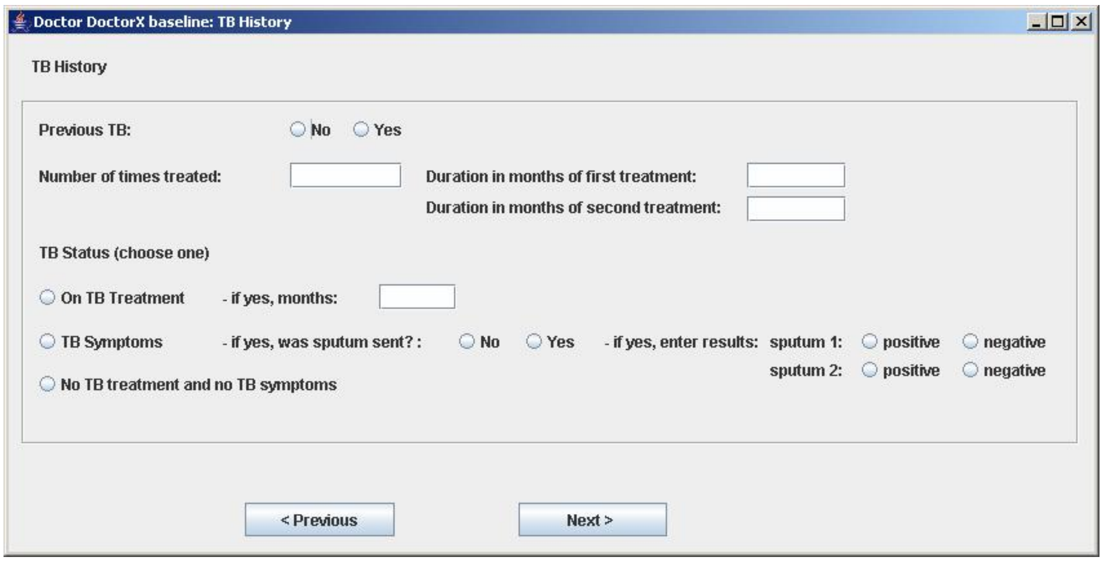
WHO Stage 2
- ☐ Herpes zoster (within last 5 years)
- ☐ Minor mucocutaneous manifestations
- ☐ Recurrent upper repiratory tract infections
- ☐ Weight loss < 10% of body weight

WHO Stage 3
- ☐ Severe bacterial infection
- ☐ Oral candidasis (thrush)
- ☐ Unexplained chronic diarrhoea (> 1 month)
- ☐ Unexplained prolonged fever (interm/const > 1 month)
- ☐ Oral hairy leukoplakia
- ☐ TB pulmonary (within last year)
- ☐ Weight loss >= 10% of body weight

WHO Stage 4
- ☐ Candida (esoph, bronchi, trachea or lung)
- ☐ Cryptococcosis - extrapulmonary
- ☐ Cryptosporidiosis with diarrhoea (> 1 mo...
- ☐ CMV (other than liver, spleen, lymph node...
- ☐ Herpes simplex (or > 1mo, visceral any d...
- ☐ HIV encephalopathy
- ☐ HIV wasting syndrome
- ☐ Kaposi's sarcoma
- ☐ Atypical myncobacteriosis, disseminated
- ☐ Mycosis, disseminated (histo, coccidiodo...
- ☐ Tuberculosis - extrapulmonary
- ☐ PCP
- ☐ PML (progressive multifocal leucoencep...
- ☐ Salmonella septicemia - non-typhoid
- ☐ Toxoplasmosis - CNS

Doctor Baseline Test Screen 2

Appendix B


Doctor DoctorX baseline: TB History

TB History

Previous TB: No Yes

Number of times treated: ______ Duration in months of first treatment: ______

Duration in months of second treatment: ______

TB Status (choose one)

On TB Treatment - if yes, months: ______

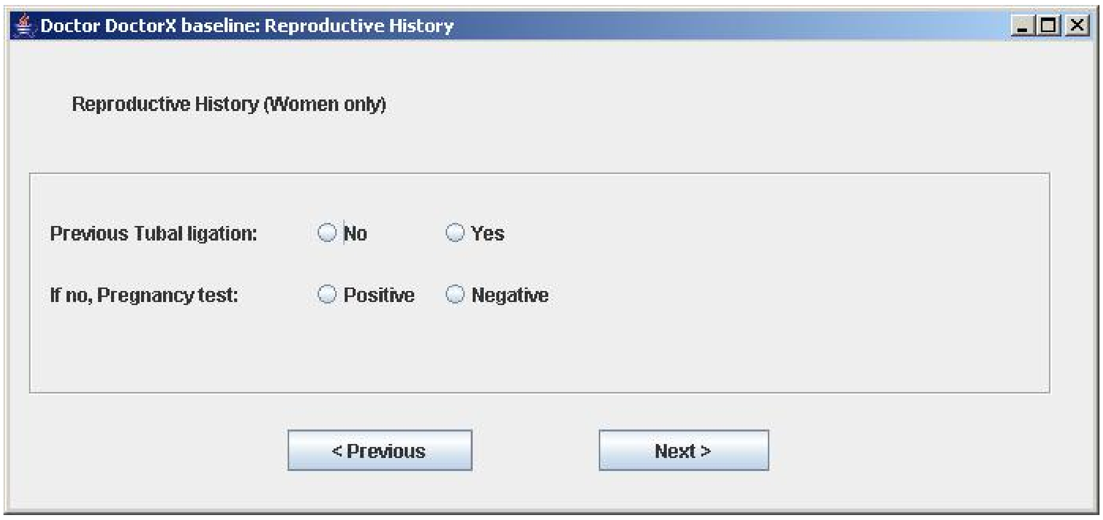
TB Symptoms - if yes, was sputum sent? : No Yes - if yes, enter results: sputum 1: positive negative

sputum 2: positive negative

No TB treatment and no TB symptoms

< Previous Next >

Doctor Baseline Test Screen 3


Doctor DoctorX baseline: Reproductive History

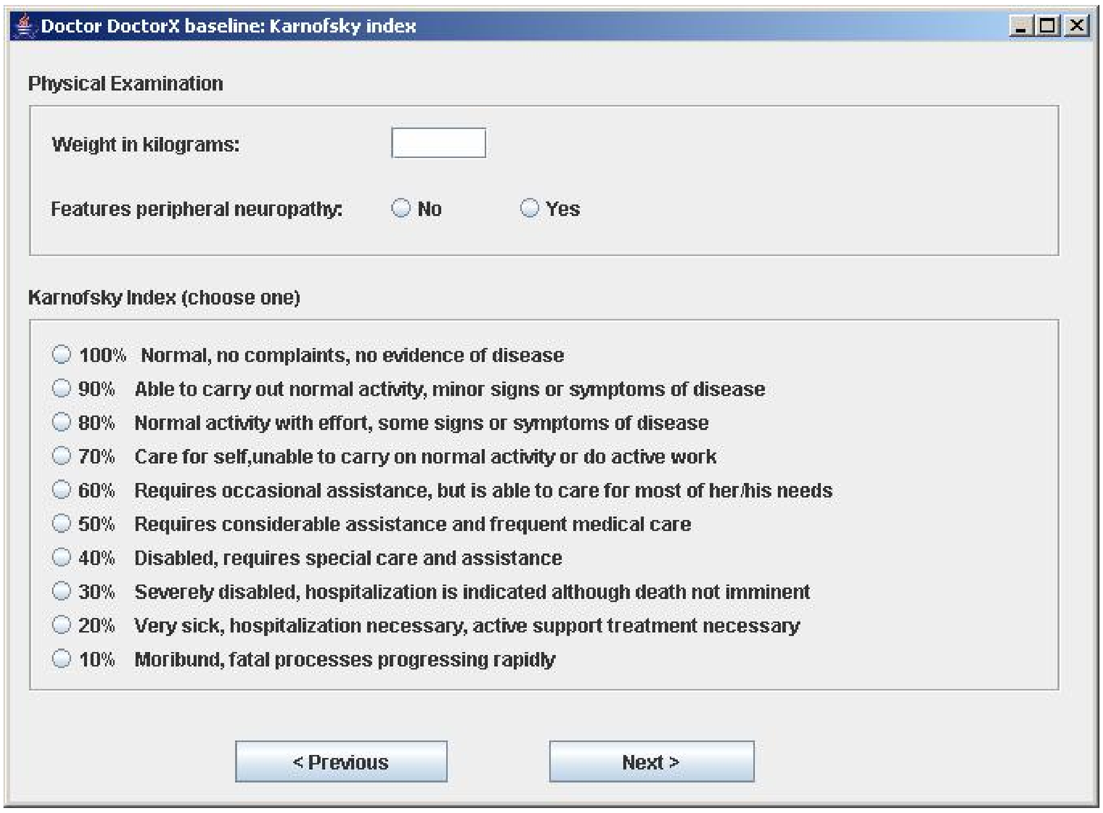
Reproductive History (Women only)

Previous Tubal ligation: No Yes

If no, Pregnancy test: Positive Negative

< Previous Next >

Doctor Baseline Test Screen 4



Doctor DoctorX baseline: Karnofsky index

Physical Examination

Weight in kilograms:

Features peripheral neuropathy: ○ No ○ Yes

Karnofsky Index (choose one)

○ 100% Normal, no complaints, no evidence of disease
○ 90% Able to carry out normal activity, minor signs or symptoms of disease
○ 80% Normal activity with effort, some signs or symptoms of disease
○ 70% Care for self,unable to carry on normal activity or do active work
○ 60% Requires occasional assistance, but is able to care for most of her/his needs
○ 50% Requires considerable assistance and frequent medical care
○ 40% Disabled, requires special care and assistance
○ 30% Severely disabled, hospitalization is indicated although death not imminent
○ 20% Very sick, hospitalization necessary, active support treatment necessary
○ 10% Moribund, fatal processes progressing rapidly

< Previous   Next >

Doctor Baseline Test Screen

Appendix C: Code Sample

```
/*****************************************************************

Pharmacist Agent:


Gets prescription data from MDBAgent.


Issues medicine and informs MDBAgent that medicine was issued.


*****************************************************************/


import jade.core.Agent;

import jade.core.AID;

import jade.core.behaviours.*;

import jade.lang.acl.ACLMessage;

import jade.lang.acl.MessageTemplate;

import jade.domain.DFService;

import jade.domain.AMSService;

import jade.domain.FIPAAgentManagement.*;

import jade.domain.FIPAException;

import jade.wrapper.AgentController;

import jade.wrapper.PlatformController;


import java.util.*;

import java.text.SimpleDateFormat;

import java.util.*;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.DriverManager;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;

import java.sql.SQLException;

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;


public class PharmacistAgent extends Agent {

       // User interfaces


    public PharmacistGui myGui;

    public PharmacistPatientPersonalInfo myPersonalInfoForm;

    public PharmacistIssueARVs myPharmacistIssueARVs;


//********************************************

// Agent initializations

//********************************************


    protected void setup() {

              // Add behaviours


        addBehaviour(new PatientDataReceived());

        addBehaviour(new PrescriptionReceived());


              // Initialize GUI

        newConsultation();


    }

//********************************************

// Agent clean-up operations

//********************************************


    protected void takeDown() {
            //  Close the GUI
        myGui.dispose();


    }


//***********************************************
//   Behaviours
//***********************************************


/***********************************************/
    public void getData(){
    /* Oneshot behaviour activated when patient number is entered by pharmacist
       Send request to PatientAgent to send personal data of patient
       and send request to MDBAgent to send prescription of patient  */

        addBehaviour(new OneShotBehaviour() {
            public void action() {
            // Create Request for Data to send to Patient Agent
                String ID = myGui.textFileNumber.getText();
                ACLMessage getMessage = new ACLMessage(ACLMessage.REQUEST);
                getMessage.addReceiver(new AID(ID,AID.ISLOCALNAME));
                getMessage.setConversationId("clinicVisit");
                getMessage.setContent(ID);
                getMessage.setReplyWith("clinicVisit"+System.currentTimeMillis());
                myAgent.send(getMessage);

            // Create Request for Data to send to MDBAgent
                ACLMessage          getPrescriptionMessage          =          new
                                                    ACLMessage(ACLMessage.REQUEST);
                getPrescriptionMessage.addReceiver(new AID("Med",AID.ISLOCALNAME));
                getPrescriptionMessage.setOntology("pharmacy-get-request");
                getPrescriptionMessage.setConversationId("getPrescription");
                getPrescriptionMessage.setContent(ID);

                getPrescriptionMessage.setReplyWith("getPrescription"+
                                                                System.currentTimeMillis());

                myAgent.send(getPrescriptionMessage);


            // Open Personal Info GUI

                myPersonalInfoForm.setVisible(true);


            }

        } );

    }       // End of getData()

/************************************************/
```

```
/************************************************/

    public void UpdateIssues() {

    /* Oneshot behaviour activated by Next button in PharmacistIssueForm

     * Sends request to PatientAgent to update visit data */

        addBehaviour(new OneShotBehaviour() {

            public void action(){

                ACLMessage updateIssueMessage = new ACLMessage(ACLMessage.REQUEST);

                updateIssueMessage.addReceiver(new AID("Med",AID.ISLOCALNAME));

                updateIssueMessage.setConversationId("issueUpdate");

                // Create Content of Message (String)

                Properties issueData = new Properties();

                issueData.setProperty("patientID", patientID);

                issueData.setProperty("prescriptionID",prescriptionID.toString());

                // Update all ARVs that were issued

                if (myPharmacistIssueARVs.jCheckBoxARV1.isSelected()) {

                    issueData.setProperty("ARV1",

                    myPharmacistIssueARVs.jCheckBoxARV1.getText());

                }

                else {issueData.setProperty("ARV1","");}

                if (myPharmacistIssueARVs.jCheckBoxARV2.isSelected()) {

                    issueData.setProperty("ARV2",

                    myPharmacistIssueARVs.jCheckBoxARV2.getText() );

                }

                else {issueData.setProperty("ARV2","");}

                if (myPharmacistIssueARVs.jCheckBoxARV3.isSelected()) {

                    issueData.setProperty("ARV3",

                    myPharmacistIssueARVs.jCheckBoxARV3.getText() );

                }

                else {issueData.setProperty("ARV3","");}

                if (myPharmacistIssueARVs.jCheckBoxARV4.isSelected()) {

                    issueData.setProperty("ARV4",

                    myPharmacistIssueARVs.jCheckBoxARV4.getText() );
                }
                else {issueData.setProperty("ARV4","");}

                updateIssueMessage.setContent(issueData.toString());
                updateIssueMessage.setReplyWith("issueUpdate"+
                                            System.currentTimeMillis());
                // Send message
                myAgent.send(updateIssueMessage);

            }
        } );
    }  // End of UpdateIssues()
/************************************************/

/************************************************/
    private class PatientDataReceived extends CyclicBehaviour {
    // Cyclic Behaviour to receive patient data from PatientAgent
        public void action() {
            // Receive message
            MessageTemplate receiveTemplate =
                    MessageTemplate.MatchConversationId("clinicVisit");
            ACLMessage receiveMessage =
                    myAgent.receive(receiveTemplate);

            if (receiveMessage != null) {
                // Break up message content into string values
                Properties values = (new
                     MyProperties(receiveMessage.getContent())).GetMyProperties();
                firstName = values.getProperty("firstName","");
                surname = values.getProperty("surname","");
                dateOfBirth = values.getProperty("dateOfBirth");
                IDNo = values.getProperty("IDNo","0");
                weight = values.getProperty("weight","0");
                gender = values.getProperty("gender","0");
                patientID = receiveMessage.getSender().getLocalName();

                // Put received values into GUI
                myPersonalInfoForm.jTextFieldFirstName.setText(firstName);
                myPersonalInfoForm.jTextFieldSurname.setText(surname);
                myPersonalInfoForm.jTextFieldDateOfBirth.setText(dateOfBirth);
                myPersonalInfoForm.jTextFieldIDNo.setText(IDNo);
                if (gender.equals("M")) {
                    myPersonalInfoForm.jTextFieldGender.setText("Male");
                }
                else {myPersonalInfoForm.jTextFieldGender.setText("Female");}
                myPersonalInfoForm.jTextFieldWeight.setText(weight);

            }
            else {   // No message received.  Block execution
                block();
            }
        } //  End of action
    }  // End of Behaviour PatientDataReceived
/************************************************/

/************************************************/
    private class PrescriptionReceived extends CyclicBehaviour {
    // Cyclic Behaviour to receive prescription data from MDBAagent
        public void action() {
            MessageTemplate receiveTemplate =
                MessageTemplate.MatchConversationId("getPrescription");
            // Receive message
            ACLMessage receiveMessage = myAgent.receive(receiveTemplate);
            if (receiveMessage != null) {
```

```
// Break up message content into string values

Properties values = (new

 MyProperties(receiveMessage.getContent())).GetMyProperties();

 String prescriptionstr =

               values.getProperty("prescriptionID","");

 if (prescriptionstr != "") {

     prescriptionID = Integer.parseInt(prescriptionstr);}

String ARVID1= values.getProperty("ARVID1","");

String ARVName1 = values.getProperty("ARVName1","");

String ARVDose1 = values.getProperty("ARVDose1","");

Integer ARV1Issuesleft =

  Integer.parseInt(values.getProperty("ARV1Issuesleft","0"));

String ARVID2= values.getProperty("ARVID2","");

String ARVName2 = values.getProperty("ARVName2","");

String ARVDose2 = values.getProperty("ARVDose2","");

Integer ARV2Issuesleft =

  Integer.parseInt(values.getProperty("ARV2Issuesleft","0"));

String ARVID3= values.getProperty("ARVID3","");

String ARVName3 = values.getProperty("ARVName3","");

String ARVDose3 = values.getProperty("ARVDose3","");

Integer ARV3Issuesleft =

  Integer.parseInt(values.getProperty("ARV3Issuesleft","0"));

String ARVID4= values.getProperty("ARVID4","");

String ARVName4 = values.getProperty("ARVName4","");

String ARVDose4 = values.getProperty("ARVDose4","");

Integer ARV4Issuesleft =

  Integer.parseInt(values.getProperty("ARV4Issuesleft","0"));


// Place values in GUI


myPharmacistIssueARVs.jCheckBoxARV1.setText(ARVID1);

myPharmacistIssueARVs.jCheckBoxARV2.setText(ARVID2);
myPharmacistIssueARVs.jCheckBoxARV3.setText(ARVID3);

myPharmacistIssueARVs.jCheckBoxARV4.setText(ARVID4);

myPharmacistIssueARVs.jLabelARV1Name.setText(ARVName1);

myPharmacistIssueARVs.jLabelARV2Name.setText(ARVName2);

myPharmacistIssueARVs.jLabelARV3Name.setText(ARVName3);

myPharmacistIssueARVs.jLabelARV4Name.setText(ARVName4);

myPharmacistIssueARVs.jLabelARV1Dose.setText(ARVDose1);

myPharmacistIssueARVs.jLabelARV2Dose.setText(ARVDose2);

myPharmacistIssueARVs.jLabelARV3Dose.setText(ARVDose3);

myPharmacistIssueARVs.jLabelARV4Dose.setText(ARVDose4);


if (ARVID1=="") {

    myPharmacistIssueARVs.jCheckBoxARV1.setVisible(false);

 }

if ((ARVID1!="") && (ARV1Issuesleft < 1)) {

    myPharmacistIssueARVs.jLabelARV1Dose.setText(

                                        "All issues done");

    myPharmacistIssueARVs.jCheckBoxARV1.setVisible(false);

}

if (ARVID2=="") {

    myPharmacistIssueARVs.jCheckBoxARV2.setVisible(false);}

if ((ARVID2!="") && (ARV2Issuesleft < 1)) {

    myPharmacistIssueARVs.jLabelARV2Dose.setText(

                                        "All issues done");

    myPharmacistIssueARVs.jCheckBoxARV2.setVisible(false);

}

if (ARVID3=="") {

     myPharmacistIssueARVs.jCheckBoxARV3.setVisible(false);}

if ((ARVID3!="") && (ARV3Issuesleft < 1)) {

    myPharmacistIssueARVs.jLabelARV3Dose.setText(

                                        "All issues done");

    myPharmacistIssueARVs.jCheckBoxARV3.setVisible(false);
```

```
                }
                if (ARVID4=="") {
                    myPharmacistIssueARVs.jCheckBoxARV4.setVisible(false);}
                if ((ARVID4!="") && (ARV4Issuesleft < 1)) {
                    myPharmacistIssueARVs.jLabelARV4Dose.setText(
                                                    "All issues done");
                    myPharmacistIssueARVs.jCheckBoxARV4.setVisible(false);
                }
            }
            else {  // No message received.  Block execution
                block();
            }
        } //  End of action
    }  // End of PrescriptionReceived
/************************************************/
```

```
/************************************************/

    protected void closeGUIs() {

    // Close GUIs for Pharmacist

        myGui.setVisible(false);

        myGui.dispose();

        myPersonalInfoForm.setVisible(false);

        myPersonalInfoForm.dispose();

    }

/************************************************/


/************************************************/

    protected void newConsultation(){


    // Open GUIs for Pharmacist


        myGui = new PharmacistGui(this);

        myPersonalInfoForm = new PharmacistPatientPersonalInfo(this);

        myPharmacistIssueARVs = new PharmacistIssueARVs(this);


        myPersonalInfoForm.setVisible(false);

        myPharmacistIssueARVs.setVisible(false);

        myGui.setVisible(true);

    }

/************************************************/


/************************************************/

// Variables declaration


    String dateDiagnosed, dateBaseline, dateOfBirth, dateToday, dateARVStart;

    String CD4Date, hospitalisation1, hospitalisation2, hospitalisation3;

    String pretreatmentCD4Date, timesHospitalised, firstName;

    String hospital, patientID, IDNo, referringClinic;
```

Appendix C

```
String surname, weight, gender;

String datePreviousVisit, prevARV1;

String prevARV2, prevARV3, Cotrimoxazole, Fluconazole, INH;

String TBTreatment, CD4, VL, ALT, Triglycerides;

String Hb, FastingGlucose, FastingCholesterol, ARV1, ARV2, ARV3, ARV4;

int  treatmentStage;

int year,month,day;

String ARVID1,ARVName1,ARVDose1;

String ARVID2,ARVName2,ARVDose2;

String ARVID3,ARVName3,ARVDose3;

String ARVID4,ARVName4,ARVDose4;

boolean ARV1Issued=false,ARV2Issued=false,ARV3Issued=false,ARV4Issued=false;

Integer prescriptionID = 0;
```