TALLIN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics
Chair of Software Engineering

THE AGENT-ORIENTED APPROACH TO APPLICATION DEVELOPMENT

Bachelor's Thesis

Student: Aleksandr Jermakov
Student code: 123631IAPB
Supervisors: Jekaterina Ivask
Kristina Murtazin

Tallinn
2015

Author's Declaration

I declare that this thesis is based on my own work and has not been submitted for any degree of examination in other universities. All ideas, major views and results of researches by other authors are used only with a reference to the source.

---------- ----------

(*date*) (*signature*)

Abstract

The aim of this bachelor thesis is to investigate the domain of agent-oriented programming and the facilities that this approach provides for the development of applications.

The development of distributed application is one of the problems for which the agent-oriented approach can provide a proper solution. The results of the research of this problem and its solution are provided in this thesis as well.

The result of this thesis is the agent-based application, which has been developed during the research process of the agent-oriented approach. The application is supposed to gather student's data from Moodle and uses several software agents, which can run on different hosts and compose a distributed platform.

The thesis is in English and contains 39 pages of text, 4 chapters, 9 figures and 3 tables.

Annotatsioon

Töö eesmärgiks on uurida agentorienteeritud programmeerimist ja võimalusi, mida selline lähenemisviis pakub rakenduste arendamise jaoks.

Jagatud rakenduste arendamine on problem mille jaoks agentorieteeritud lähenemisviis annab mõistlikku lahendust. Selle probleemi uurimise tulemusi ja probleemi lahendus on pakutud selles töös.

Töö tulemuseks on rakendus, mis oli arendatud agentorienteeritud lähenemisviisi abil agentide ala uurimise protsessi jooksul. Rakenduse eesmärgiks on koguda tudengi andmeid Moodle'st kasutades programmagente, mis võivad töötada erinevatel arvutitel ja kujundavad jagatud platvormi.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 39 leheküljel, 4 peatükki, 9 joonist ja 3 tabelit.

Abbreviations

AOP	***Agent-Oriented Programming***
OOP	***Object-Oriented Programming***
JADE	***Java Agent Development Framework***
FIPA	***The Foundation for Intelligent Physical Agents***

List of Figures

List of Tables

Table of Contents

1. Introduction

The agent-oriented programming approach is one of the programming paradigms where the implementation of the software bases on the concept of software agents and their interaction with users, an environment and each other.

The agent-oriented programming can be viewed as a specialization of object-oriented programming (Shoham, 1990) which extends its potential to provide software developers with a new powerful approach to the programming and brings a new view on the software development in general.

1.1 Research Problem

The domain of agent-oriented programming is being researched by many scientists around the world since the definition of this concept by Yoav Shoham in 1990. Despite the fact that the object-oriented and other programming paradigms provide developers with adoptable solutions for almost every existing problem in the software engineering area, several objectives can be achieved with a much easier and more effective way using the agent-oriented approach. One of those problems is the development of an application, which can be distributed across the several machines in order to use distributed computation to increase the performance of the application.

1.2 Research Objectives

The main objective of this thesis is to research the agent-oriented programming approach, its potential, advantages and disadvantages comparing to the pure object-oriented programming, and to implement a software product using the approach described above.

1.3 Methodology

The objectives are going to be achieved by implementing a distributed application, based on software agents, and using the agent-oriented approach. The target environment to interact with is Moodle – learning management system, and, more specifically, HITSA Moodle,

which is currently being used in educational purposes by many universities in Estonia, including Tallinn University of Technology. The target programming language for the application is Java (version 8). The agents in this application represent user (personal) agents, which main goal is to collect, sort and present information for user. Any other agents, including the data-mining, predictive and agents from the artificial intelligence agents, are out of the scope of research and not mentioned in this thesis.

1.4 Thesis Structure

The thesis is divided into three chapters.

The first chapter is an overview of agent-oriented programming, which serves as an introduction to the domain of agents. This chapter briefly describes the most primary aspects and principles of agent-oriented programming, agents themselves, agent-based systems and applications. The commonly used for agent programming technologies are reviewed in this chapter as well.

The second chapter is focuses on an agent-based application, which has been developed for this thesis during the research of agent-oriented programming. The detailed overview of the workflow of the application as well as its architecture is provided in this chapter. The chapter also contents a description of technologies that were used during the development of the application.

The third chapter is a conclusion. The objective of this chapter is to give a brief overview of what was done during the research and development processes and to summarize the results of the thesis.

2. Agent-Oriented Programming

2.1 Agent-Oriented Programming Overview

The term “agent-oriented programming” (AOP) was first proposed by Yoav Shoham during his research of artificial intelligence in 1990. According to the author, this new programming paradigm can be viewed as a specialization of object-oriented programming (OOP), since it takes the core of OOP and transfers it into a new approach to programming, currently known as agent-oriented programming.

The main idea of AOP is to centralize software development around the concept of software agents – a particular implementation of intelligent agents as objects in the programming. The entire process of calculation in the program, which was built using the AOP approach, executes by interaction of software agents with an environment, which provides the agents with an essential data, with users, which are using the program, and by communication between the agents.

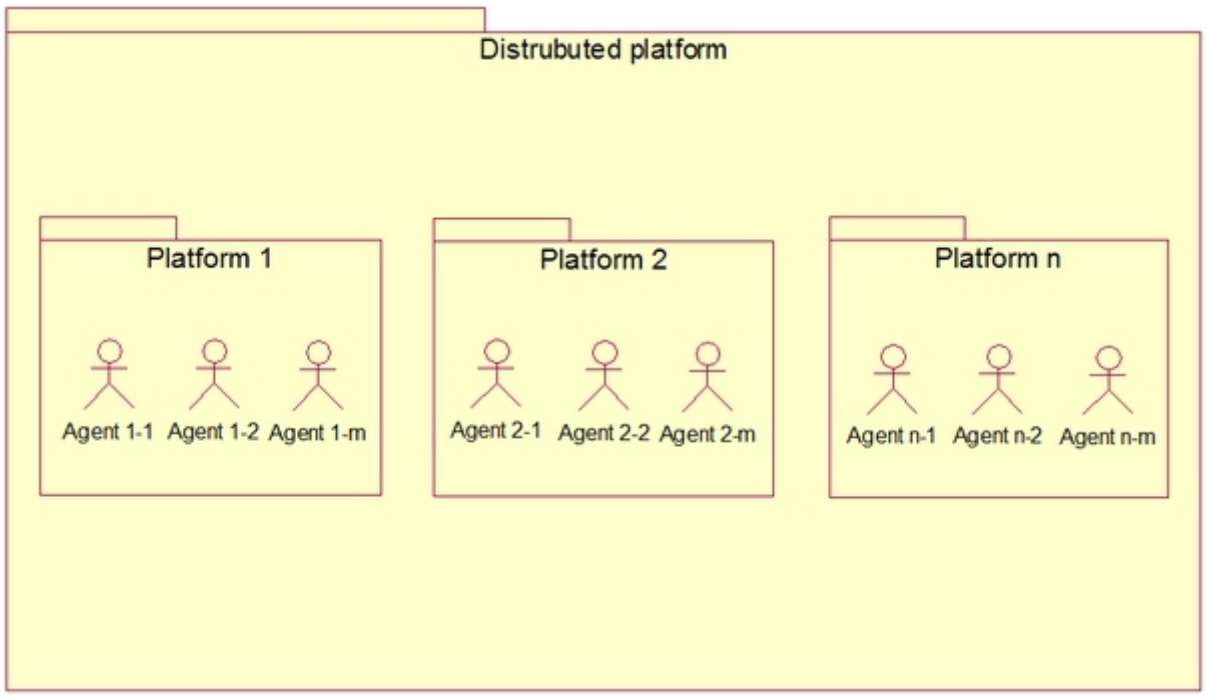

Figure 1. The high-level representation of an application as a distributed platform based on the AOP concept.

The significant advantage of AOP is that this approach, as long as agents can work separately, supposes the building of an application, which can be easily distributed across the several platforms (computers) to use their computing capabilities together to increase the speed of computation. In other words, the program, created using AOP, is an application, consisting of several agents, which are distributed across the platforms (**Figure 1**).

2.2 Agent and Object Oriented Programming Comparison

Since AOP can be viewed as a special case or a computational programming framework of OOP, it is frequently being compared with its parent. Whereas OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and that have individual ways of handling incoming messages, AOP specializes the framework by fixing the state (now called mental state) of modules (now called agents) to consist of precisely defined component called beliefs (including beliefs about the world, about themselves, and about one another, capabilities, choices and possibly other similar notions. (Shoham, 1997). All computations in AOP consist of messaging (inform, offer, accept, reject and other, more specified, request- and response-messages) between agents using predetermined protocol, the same way as the messaging between objects in OOP, just in a more constrained way. The **Table 1** represents the main difference between AOP and OOP paradigms:

Paradigm	OOP	AOP
Basic unit	Object	Agent
Parameters defining state of basic unit	Unconstrained	Beliefs, commitments, capabilities, choices…
Process of computation	Message passing and response methods	Message passing and response methods
Types of message	Unconstrained	Inform, request, offer, promise, decline…
Constraints on methods	None	Honesty, consistency…

Table 1. OOP and AOP comparasion.

More briefly, the main difference between AOP and OOP is the using of agents in the first one, which act by sending and receiving messages according to their pre-known set of rules, beliefs and commitments. Not to be confused, not all objects in AOP are agents, and the

existence of agents do not eliminate the utilization of objects (in the sense in which objects are represented in OOP).

2.3 Agents

2.3.1 Agents Overview

As already was mentioned in the section above, the agents in AOP can be viewed as an implementation of the objects from OOP with defined behavior according to their beliefs. The term “agents” in AOP implies the “software agents”, an autonomous computer program, which can be viewed as an entity that can act on behalf of someone in the environment and perceive events and reasons (Kuldar Taveter, Alex Norta, 2015).

There are a lot of properties that can describe the software agents and that most researches consider to include in their definition of software agent. Agents typically possess several (or all) of the following characteristics (Todd Sundsted, 1998):

- Autonomy – an ability to act separately from other components in application.
- Adaptability – a change own behavior according to the analyzed data, set of facts and own beliefs.
- Mobility – a possibility to move from platform to platform which are adorable for agent to work in.
- Persistence – code is not executed on demand but runs continuously and decides for itself when it should perform some action.
- Goal oriented – the main purpose of every agent is to be able to complete received task using own abilities and resources.
- Collaboration – an ability to work with other agents together to achieved the assigned task.

Agents also supposed to be small, since a single agent do not represent a complete application, but the application is being formed by agents and their cooperation with its components and each other.

2.3.2 Agents Architecture

Despite the fact that every developer should decide by himself the main functional of his agent, what the agent should do and how should react on the action from the outside, there is a set of basic requirement that should be implemented in order to transform a simple project into an agent.

First of all, there must be a possibility to distinguish one agent from another and consequently an agent must have its own identity – a unique object that can identify the agent.

Secondly, an agent must belong to an agent host – a platform, which provides agents with resources they need for working (the agent hosts are in detail described in the next section)
Thirdly, an agent should be able to communicate with other agents and therefore agents need an implementation of communication mechanism, which would allow agents to send and receive messages. Usually such mechanism works with the agent host's components to provide the ability to communicate for the agent.

To summarize, the high-level architecture of any agent can be represented in the next way (**Figure 2**): an agent has a unique instance, a reference to its agent host and a communication mechanism.

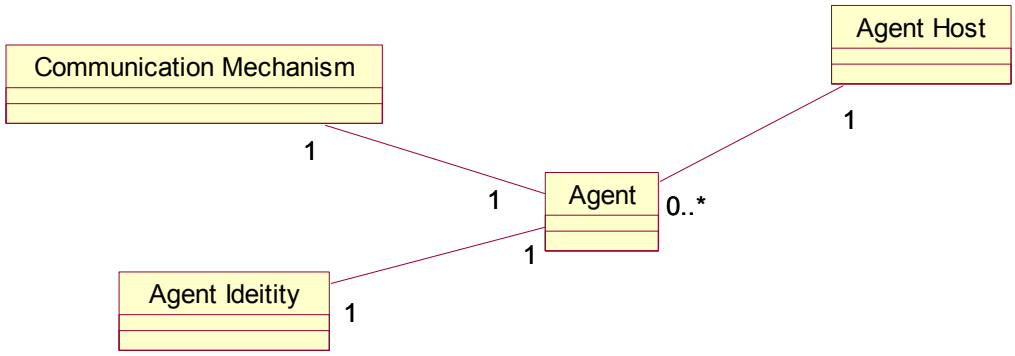

Figure 2. The architecture of an agent.

The lifecycle of an agent is also well-defined and is very similar to the applet's lifecycle (Todd Sundsted, 1998).

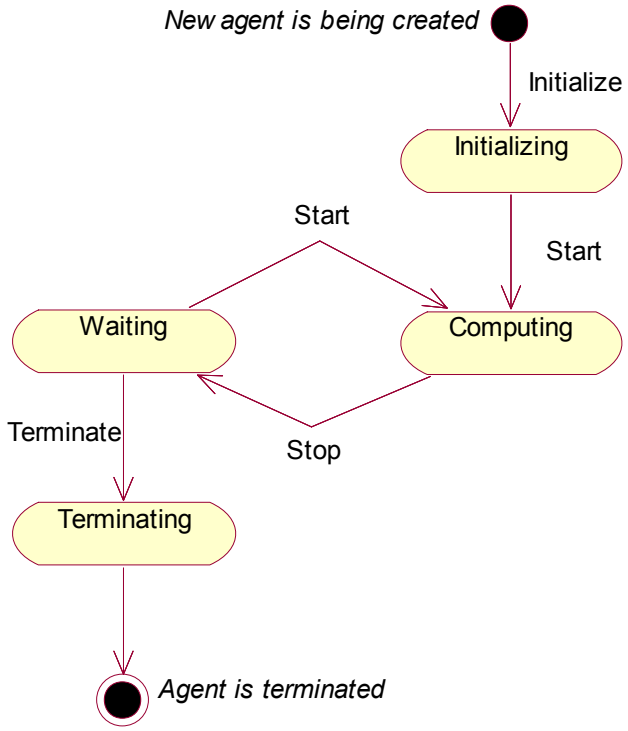

Figure 3. The lifecycle of an agent.

As **Figure 3** illustrates, it begins with one-time initialization, which setups all of initial data structures of the agent that the agents need to start working. When initialization is finished, an agent starts its calculations. After the calculations is finished, an agent stops and waits until he will be invoked to begin some new activity or to be terminated. Eventually, the agent's lifecycle finishes with the one-time termination of the agent.

2.3.3 Agent hosts

Considering the fact that agents should have a place where they are being executed, a new component, the agent platform, also known as agent host, must be implemented in order to provide the agents with resources needed to work. Just as well as the agents, the implementation of an agent platform depends on a developer and has the list of mandatory attributes (Todd Sundsted, 1998):

- An agent platform must allow multiple agents to co-exist and execute simultaneously.
- An agent platform must be able to initialize and terminate agents.

- An agent platform must allow agents to communicate with each other and via itself.
- An agent platform must be able to negotiate the exchange of agents.
- An agent platform must be able to freeze an executing agent and transfer it to another host.
- An agent platform must be able to thaw an agent transferred from another and allow it to resume execution.
- The agent platform must prevent agents from directly interfering with each other.

In other words, an agent platform can be viewed as an environment that manages agents, provides them with a place and resources to work and helps them to work with each other. In general, a distributed application, which is based on agent's work, consists of several agent platforms, which, in the same time, contains several agents.

2.4 Agent Frameworks Overview

Since the first mention of AOP, a lot of researches has been done in this area. Even thought the agent-oriented approach is not using as much as its parent, OOP, there is a plenty of frameworks, which were developed to simplify the implementation of agents and agent applications.

Usually, the agent frameworks provide developers with a pre-written classes that implements the agents and agent platforms, what allows developers to abstract from the creating of inner architecture and coding such basic things as messaging between the agents, the search engine for agents and other common components. In addition, some of these frameworks also provide useful tools to simplify the development and testing processes, such as graphical user interface to monitor platforms, manage agents and send fake messages to control the behavior of agents during the receiving of messages.

The frameworks for agent development exist for a majority of programming languages, including C, C+, Java, Visual Basic and many other languages. Considering that the one of the special conditions of this thesis is to implement an agent-based application on Java, only the frameworks for Java are going to be reviewed as a possibility to build the application with.

There are several frameworks for implementation of agents and agent-based systems and application. As long as different frameworks suggest different functionality and tools for development, the selection of framework depends on the developer's final goal. Since the objective of this thesis is to build a distributed agent-based application, the following frameworks might be examined:

- JADE (Java Agent Development Framework)
- JABM (Java Agent-Based Modeling toolkit)
- MASON (Multi-Agent Simulator Of Neighborhoods)

2.5 Comparison of Frameworks

The brief description of the chosen frameworks and differences between them are listed in the **Table 2** below:

Framework	Stable release	User Support	GUI	Description
JADE	November 12, 2014	Tutorials, examples, API, documentation, books.	Yes	Framework for distributed applications based on autonomous agents.
JABM	January 7, 2015	Tutorials, API, documentation.	No	Framework for building agent-based models.
MASON	December 17, 2010	Tutorials, examples, API, documentation.	Yes	Framework for agent systems, social complexity and abstract modeling.

Table 2. The table of comparasion of frameworks for agent development.

According to the functionality that provide listed frameworks, one of the best solutions to start the research and development of agent-based application is JADE. JADE is still being supported by its developers, provides a lot of documentation and tutorials which allows developers to get familiar to the framework pretty quickly and, as one of the most important advantages, it contents built-in GUI which simplifies the development and testing processes of application.

JABM belongs to one of the few modern frameworks for agent-oriented programming and uses an object-oriented design pattern called dependency-injection (Phelps, 2012). Some of the capabilities provided by JABM might be a little bit better that their older analogues from JADE, but this frameworks still lacks the documentation and GUI to provide its developers more easier and flexible way to setup their applications.

And the last one, MASON, is one of the oldest agent-based frameworks. It can be very useful in case of building very huge agent systems which implements social activity, but is too complicated for the project in this thesis.

Considering all the advantages of different frameworks listed above, the best solution for this thesis will be JADE, because it all based around the distributed application, contents a lot of auxiliary materials and provides GUI for developing and testing the agents.

2.6 JADE Framework

As was already mentioned, JADE is a framework that simplifies the development of agent-based systems and applications.

JADE implements the core of AOP, which means that any application based on JADE is made of a set of components called Agents each one having a unique name (David Grimshaw, 2010). Agents in JADE belongs to the agent-platforms, which provides agents with the resources and services such as message delivery, and are grouped into containers. Platforms and containers can be executed on different host to form a single distributed platform (**Figure 4**). Each platform contains at least one main container, which includes two already implemented agents by default:

- The AMS agent that provides the platform management and is the only one agent what is able to start and shut down other agents, containers and platforms (but other agents still can request to perform such actions).

- The DF agent that provide the Yellow Page service – a place where agents can publish the services they provide and find other agents with services they need.

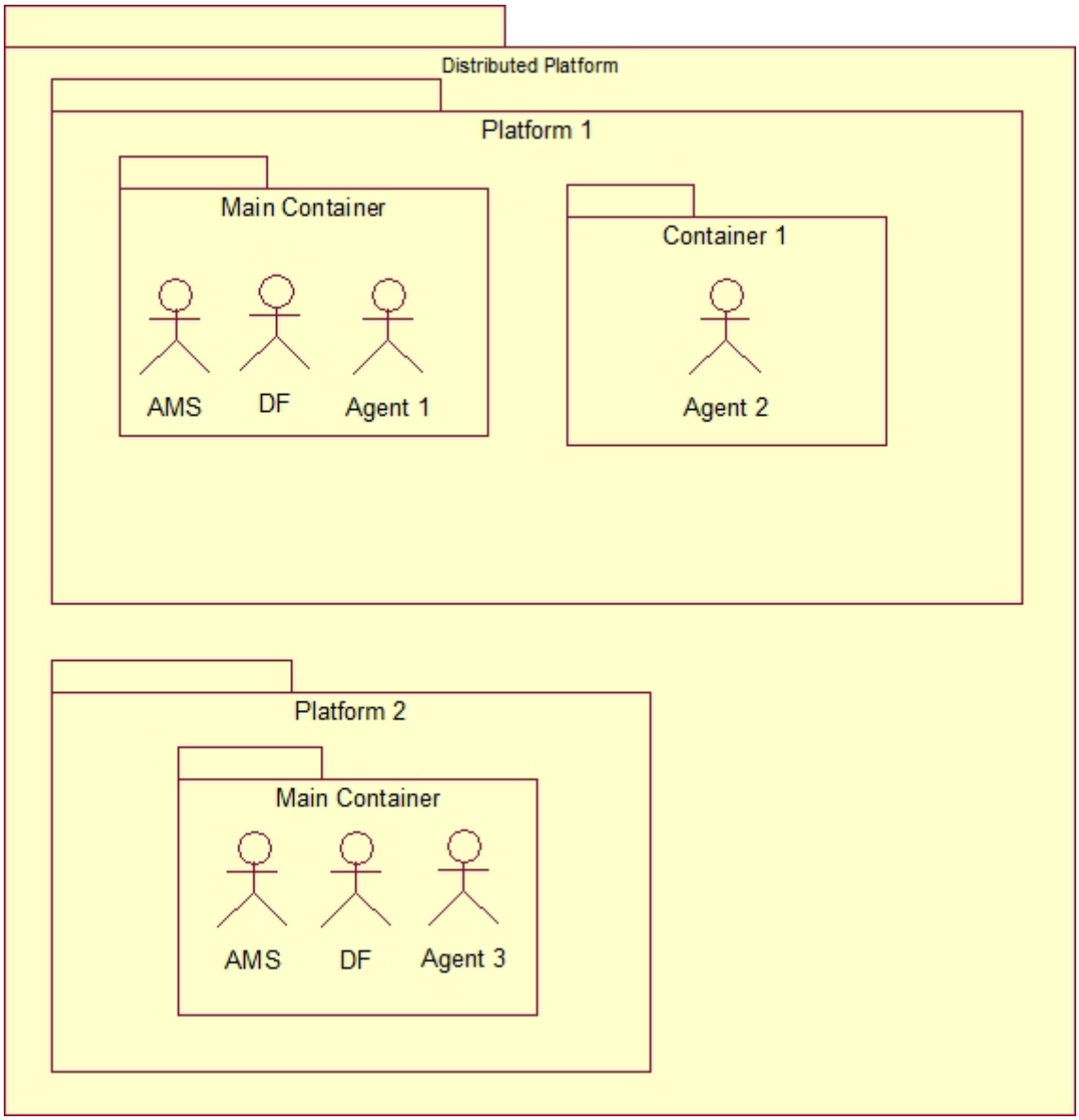

Figure 4. **An architecture of a JADE application.**

The communication between agents in JADE occures according to standart defined by FIPA - an IEEE Computer Society standards organization that provides the ACL language for messaging between the agents. ACL format is very simple, but, in the same time, provides all information that agents may need to communicate with each other, namely:

- The sender.
- The reciever(s).
- The communicative act – the field that allows reciever to understand, what sender wants to recieve as an answer.

The content of the message.While working with JADE, developers does not need to build the structure of the ACL messages by themself since JADE already provides the buisness classes for writing and reading those messages.

One of the powerfull tools provided by JADE is Remote Agent Managment GUI (**Figure 5**). The GUI simplifies the processes of development, testing and configuration of agent-based application. The main functions provided by the GUI are:

- Agent managment: starting, killing, migrating and others actions which can pe applied to the agents.
- Invoking the dummy agents to send ACL messages to test how the agents reacts on the incomming messages.
- Platform managment: creating and shutting down the platforms. Provides the possibility to join remote platforms in order to form the distributed platform as well.

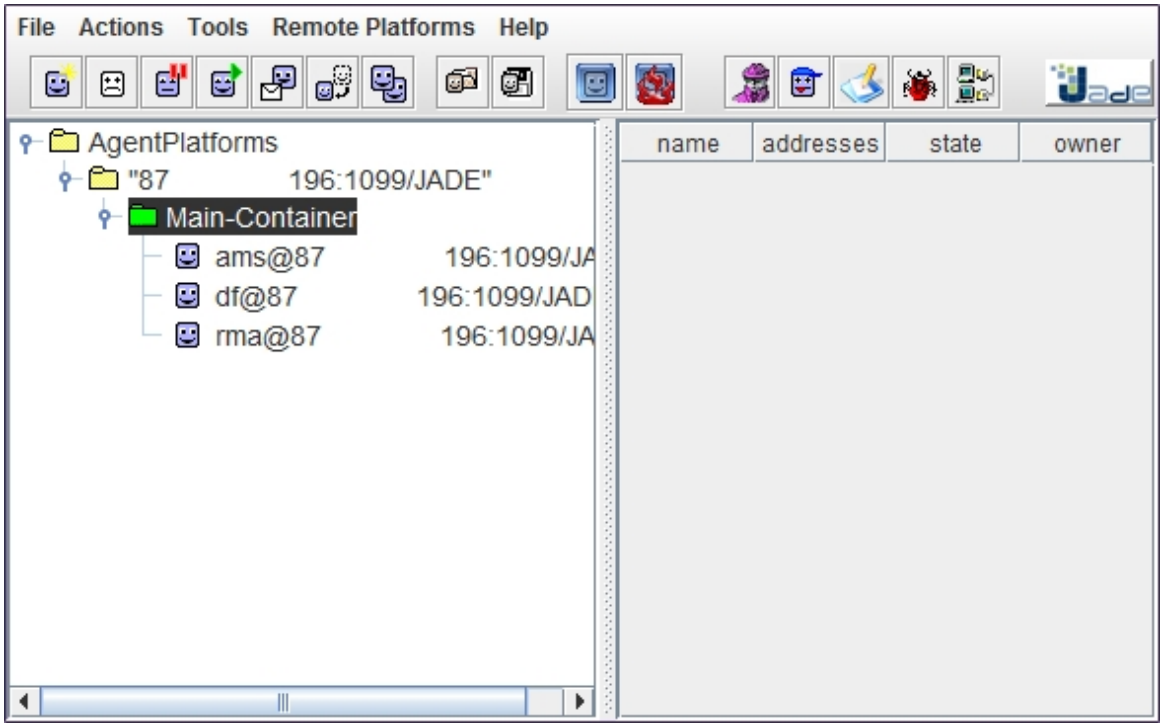

Figure 5. JADE's GUI.

To summarize, JADE provides with both the pre-written architecture of software agents and agent-platforms in order to allow developers to abstract from the long process of design of common components needed to develope an agent-based application and straight way begin the implementation of agent's logic, as well as with set of graphical tools to simplify the development and testing processes. Considering all features provided by JADE, it can be certainly choosen as a framework for development the application mentioned in this thesis.

3. The Case Study

This section is dedicated to give a detailed view on the project, which has been developed during the research process to show the potential of AOP in the domain of application development. The project represents an agent-based application, called Moodle Helper Application, which works with Moodle – the learning management system, which is being actively used by many universities in Estonia, including Tallinn University of Technology. The application is supposed to use several agents to gather student's data from the Moodle's webpage and save it. In other words, the application represents the set of gathering data agents, which work together to achieve their goal. Other functional like analyzing and using gathered data to assist students to organize the studying process is out of the scope of this thesis, but the design of the application does not eliminates the possibility to implement this function using the developed application as a gathering data module.

3.1 High-Level Design

The application consists of 3 main parts, which represent Model-View-Controller architectural pattern:

- The main agent that can be viewed as a controller. This is the core of the application what coordinates all actions and processes.
- The model – classes that are used to store gathered information.
- The view – simple GUI that is used in order to represent information to user. In fact, this part of the application is not essential, as agent can be launched from the console and be controlled by the messages, but GUI simplifies the testing process. In other words, with GUI it is easier to understand that application works correctly.

When controller is told that some student's data is required (via GUI by user or by message from any other agent), it begins to gather data, saves it into the model and represents to user using the view (or sends it back in the message, if data has been required by other agent).

3.2 Low-Level Design

3.2.1 Agents

3.2.1.1 Moodle Helper Agent

As was mentioned in the section above, the core of the application is an agent, called Moodle Helper Agent, which is responsible for collecting data and saving it into the model. As any other agent implemented with JADE, this agent extends class `Agent` from JADE.

The lifecycle of an agent (**Figure 6**) in JADE begins with `setup()` method, which is being launched automatically when the agent is being added into the platform. In this method, Moodle Helper Agent registers itself in the DF Service and, if the target student, whose data is going to be collected, has been defined in the arguments, loads student's profile, shows it and waits for another task. Otherwise, it offers to user to define the student's login and password to be able to start working.

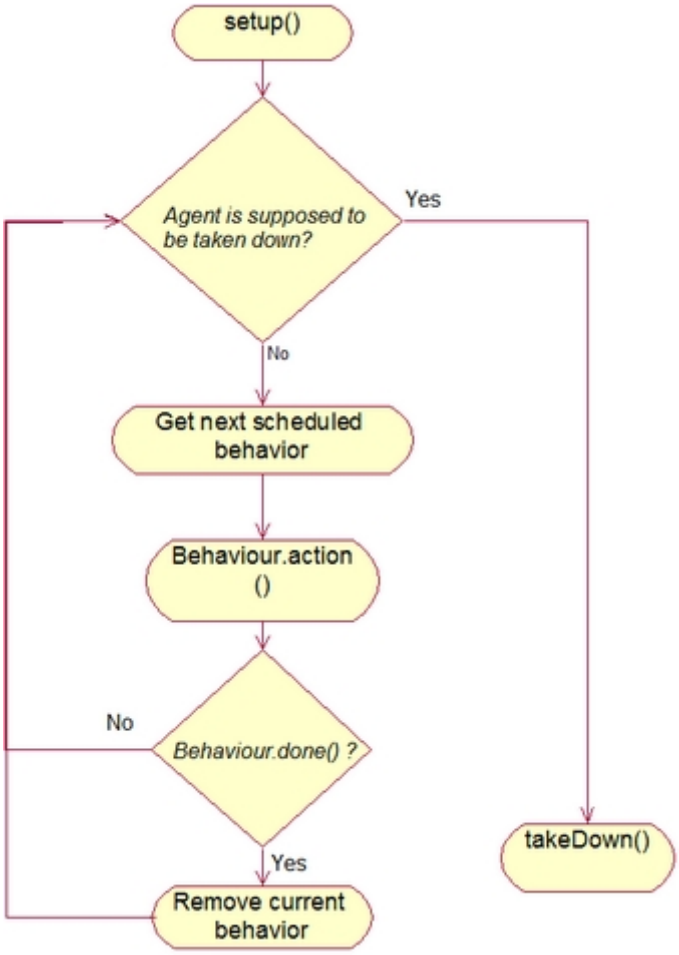

Figure 6. The lifecycle of an agent in JADE.

An action that should be performed by the agent does not located in the agent's class and does not executed directly by calling its method. There is a superclass in JADE named `Behaviour`, which is responsible for agent's activity. In order to perform an action, a behavior must be scheduled to the agent. An action that should be executed by an agent is places in the `action()` method in the `Behaviour` class. This method is called automatically when behavior is being executed. In addition, `Behaviour` class provides `done()` method. This method is called when `action()` is performed and controls if action should be performed once more (by returning `true`) or not (by returning `false`). This approach allows to create different types of behaviors, such as one-shoot behavior, which action is performed only once, cyclic behavior, which repeats again and again, and behaviors that should be performed step by step.

The agent's lifecycle ends when method `takeDown()` is called. If it happens, agents deregisters itself from the DF Service and deletes from the platform. Since Moodle-Helper Agent must work permanently and wait for tasks to perform, this method is called only if whole application with JADE platform is supposed to be closed.

3.2.1.2 Data Gathering Agent

The main goal of Moodle Helper Agent is to collect information about the student and to save it into the model. However, the mentioned agent does not do that directly by itself – it uses other agents called Data Gathering Agents to perform such action.

Data Gathering Action is an another extension of class `Agent` in JADE. Just as Moodle Helper Agent, it has the same life cycle and the same manner of performing actions by scheduling behaviours. The main aim of this agent is to directly connect to Moodle to get student's data. The behavior which is responsible for data gathering is scheduled in `setup()` method works indefinitely and never ends. The agent with this behavior always checks his mailbox for incoming messages and, if mailbox contains one, starts the gathering process. When information is collected, agent puts all gathered information into the message and sends the response.

3.2.2 Behaviours

3.2.2.1 Agent Finding Behaviour

In order to distribute the gathering process across several agents, Moodle Helper Agent should be able to find agents which provide this service. The simplest approach to this

problem is to directly put IDs of agents which provide data gathering service into the inner code of Moodle Helper Agent. This solution may be good and useful if developer knows beforehand how many target agents will be used in the application and their IDs. But if the application is supposed to contain an undefined amount of agent, and, moreover, when those agents can be launched on different platforms separately, it is impossible to predict which agents will be used to gather the information and which will be not. Fortunately, JADE provides DF service, which was already mentioned in this thesis.

DF Service provides a place where agents can register themselves and services which they provide. Every time when one agent needs to know how many agents can provide some service, it should ask it from the DF Service:

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(service); //service that should be provided by target agent
template.addServices(service);

DFAgentDescription[] result = DFService.search(myAgent, template);
```

This process has been implemented in the `Agent Finding Behavior`. Every time when an agent needs to find other agents, this behavior should be scheduled for execution. The behavior searches in DF Service for services that were required by agents, saves IDs of found agents and schedules a behavior that should be performed with found agents.

3.2.2.2 Profile Request Behaviour

The basic data that can be received from Moodle is the information about the student itself: student's name, country, city, and list of studied courses. Every time when Moodle Helper Agents needs to get this information, `Profile Request Behaviour` should be scheduled right after Agent Finding Behaviour.

Using this behavior, agent starts with forming an ACL Message. Fortunately, to simplify this process, JADE contents `ACLMessage` class and the message is forming automatically according to FIPA specification. Everything that left is to put the list of recievers' IDs into the message (in this case, there is not too much information to collect and only one target agent is needed) and specify the student whose information will be gathered. Also, agent should somehow distinguish the response to this message from other ones, which can be received in

the same time. In order to do that, the message must contain a unique ID that will be used by receiver to form an answer. Thereafter the message can be sent.

When message is sent, agent begins to wait for incoming messages in order to receive the response. When response is received, the agent unpacks the student's data from the message and saves it into the model class `Student.` Now this class can be used in order to get information about the specific course and student's achievements on this course.

3.2.2.3 Course Request Behaviour

Knowing the list of courses where the student participates in, Moodle Helper Agent can get detailed information about each course. Since any course contains several section with a lot of materials, tests and tasks for students, the gathering of such information is a process that would be reasonable to distribute across the several agents. `Course Request Behaviour` has been implemented precisely for this purpose.

Just as Profile Request Behaviour, which was described in the section above, Course Request Behaviour should be implemented after the target agents are found. It also begins with a forming of a message with request to course's data from the Data Gathering Agent. This request is formed in the same way as a request to student's information and needs only one agent to perform it. The reply on this message is course's data: sections, which divide the course into the pieces to group educational material, tests, tasks and documents, which belong to this course. Agents saves this data into the model classes `Section`, `Course`, `Test`, `Task and` Material respectively.

For now, the agent knows which sections contents given course, and which materials, tasks and tests belong to each section. If documents with material in Moodle usually are represented as an individual file, uploaded to server, tests and tasks content additional information like the date when the assignment have to be completed, the grade and other useful data, which can be used in order to analyze student's results and help to organize the studying process. In order to expand the knowledge about the tasks and tests, agent should request this information from the Data Gathering Agents, separately for each task and test. Considering that each request would take some time, this process can take too much time if all requests will be directed to one single agent. This objective should be distributed across the several agents in order to increase the performance of the application.

The process of sending requests for tasks and tests is very similar to the one for course's data. This time, the agent prepares messages for every task and test included in course and sends each message to individual agent (if the amount of tests and tasks is greater than amount of available agents, just some agents will be asked to provide information about several tests or tasks). After sending the messages, agent begins waiting until all replies will be gotten. When all required data is received, agent updates modal classes `Test` and `Task` and finished the behavior.

3.2.2.4 Data Gathering Behaviour

As far as Data Gathering Agent is an agent class implemented with JADE, it also needs a behaviour. In JADE any agent what is supposed to wait for incoming messages, what tells which action should be performed, needs a behaviour which will automatically check its mailbox for new messages. Data Request Performer is a behaviour which is scheduled in Data Gathering Agent's `setup()` method and tells the agents to be ready for incoming messages. However, the agent supposed to response only to those messages which contain a request to gather some information. `MessageTemple` class from JADE allows to quickly filter the messages and check only those with request to data gathering service which is being provided by the agent. If is known that such message should content CFP (call for purpose) performative, the following template in method `receive()`, what takes the latest message from agent's mailbox, will automatically reject all other incoming messages:

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
```

In addition to filtering messages, should be considered the fact that if continious behaviour has been scheduled, the agent will start continious loop that is extremely exhausting for system resources (Caire, 2009). The agent should suspend this action when there is no more messages left and resume when the new one is received. Luckily, there is a break() method implemented for agents in JADE. When this method is called, agent stops executing current behavior and will continue only if message will appear in the mailbox.

```
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
	//perform an action
} else {
	block();
}
```

When Data Gathering Agent gets the message that matches the implied template, it extracts information about target student (a username and a password, which will be used in order to request data from Moodle), information about data that should be collected (for example, a specific course or task). Thereafter the agent uses `HttpService` class, which was developed especially for this thesis, to send request to Moodle. As long as Moodle does not provide any open API in order to get student's data directly from its database, the only potential way to gather the data is to get student's page from Moodle and parse its inner HTML code. As a result, `HttpService` will return necessary HTML page from the Moodle directly using HTTP protocol. When page is received, another class, `HttpParser`, which is also implemented for this thesis, takes the page's inner code, parses it and returns needed information as a model object.

When required data is parsed and packed, the agent puts an object with this data into the message, sends the message back to the agent that required the data and starts waiting for another request.

3.2.3 Utilities

3.2.3.1 HTTP Service

The `HttpService` class provides a utility to access to information in Moodle. The service is based on Jsoup – a library for java which grants a set of tools to work with HTML documents, which also includes implemented network service to get HTML documents from the web servers.

In Moodle, there is a security police that doesn't allow unauthorized users to browse student profile, courses that they are learning and their result. As a consequence, the first thing that should do HTTP Service is to log in into Moodle. Firstly, it sends GET request to Moodle's main page, using `Connection` class provided in Jsoup, to get session cookies. Those cookies will be needed in order to be able to save the session during the authentication process. Secondly, POST request with student's username and password is being sent to Moodle and Moodle creates a session which can be used for future requests. When authentication is completed, HTTP Service can be used in order to get any HTML pages from Moodle. An instance of HTTP Service remembers the last session, and, as long as session is not expired, it can be used to request student's pages from Moodle without necessity to log in each time when some page is required, that highly increases the data gathering process.

3.2.3.2 HTML Parser

The `HtmlParser` is another class based on Jsoup library. As well as sending http requests, Jsoup provides a set of useful classes that make the parsing process of any HTML page much faster and easier. This functionality has been utilized in order to implement a parser for Moodle pages.

Every method of `HtmlParser` class implements the parsing of different pages from Moodle. It takes a HTML inner code of page as string, browses it and finds all information that has been required. When the page is parsed, HTML Parser saves all found information into an instance of model class (according to the object that was required) and returns it as a result of a method.

3.2.4 Model

All information that can be gathered from Moodle in this application is stored in model classes. Model classes are used in order to structure data which will be used in communication between agents in the application and represented to user.

The following classes are provided in the model in order to save data from Moodle:

- `Student`. This class can be perceived as a main class of the model, since it is related with every other class in the model. Student class stores the main information about a student – student' name, country, city and courses that currently are being studied.

- `Course`. This call represents information about a single course. It stores course's name and URL. This class does not directly knows about all materials, tests and tasks that are assigned to the course, but holds information about sections – parts of courses where materials, tests and tasks are grouped by their topic.

- `Section`. As was mentioned before, any course in Moodle consists of smaller pieces – sections. Every section contents different materials, test and tasks, which are usually related to the same topic.

- `Material`. The information about different documents, such as presentations or studying materials, is saved in this class.

- `StudentTest`. This class contains information about tests which student should complete during the studying the course. It stores test's name, a date until which the test should be completed and a grade that student gets when test is resolved.

- Task. The tasks are very similar to tests – they also have a name, a date and a grade, but since the Moodle has the different page structure for tests and tasks, which should be parsed differently, an individual class has been implemented to store information about tasks that student completes on the course.

As **Figure 7** illustrates, Moodle has the following simple structure which also has been implemented in this thesis: there are students, students learn courses which consist of sections with materials, tasks and tests related to this course.

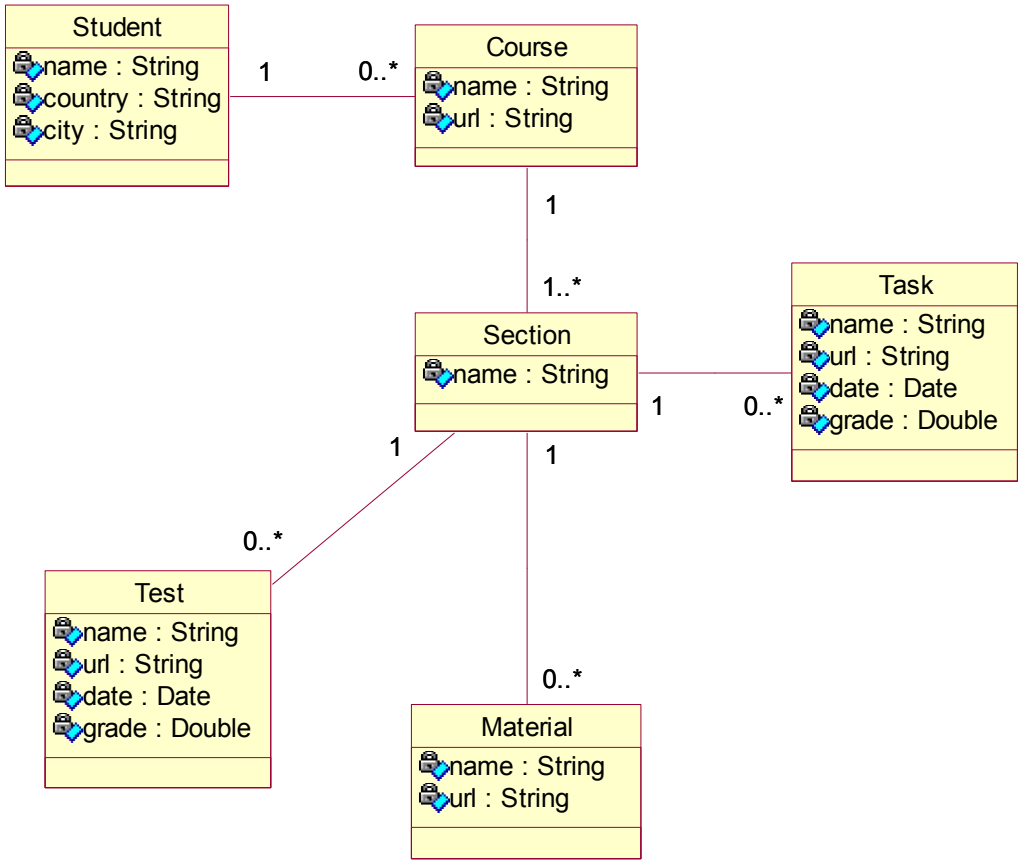

Figure 7. Class diagramm for model in Moodle Helper Application.

3.2.5 Graphical User Interface

The graphical user interface (GUI) is also has been developed for this application. Even though very often software agents does not provide any GUI at all, in this application GUI has been implemented in order to represent information that is being gathered in a more visual form to a user. GUI also simplifies the testing process of the application and allows to clearly understand that application work as it is supposed to.

The detailed description of development and design of GUI is an individual domain of research and is out of the scope of this thesis, but, considering that the application has a GUI that allows to user to interact with its agents, a brief review of GUI is provided in this section.

GUI for this application has been implemented using Swing graphical library, which is included in the Java core components. All GUI can be divided into two parts: the login frame and the agent frame.

The login frame appears immediately after the launching of Moodle Helper Agent in case where the agent is launched without specified username and password. This frame allows to specify a student whose data is supposed to be collected during the work of agents.

The agents frame is a view which is used by Moodle Helper Agent to represent collected information to the user. This frame can be divided into three components:

- The navigation panel, which is used to switch between user's profile and courses;
- The profile panel, where is the main information about the student is shown.
- The courses panel, the place where is the information about courses, sections, materials, tests and tasks is presented.

When users logs in or selects specific course to display, Moodle Helper Agent starts the process of data gathering. When all required data is collected, the agent forwards it as an instance of a model class to GUI and updates the needed panel.

3.2.6 Launcher

The launcher class is the first class that is being called during the launching of the application. This class is provided in order to be able to configure the application before the launching

from the command line. The following arguments can be passed to the class in order setup the application's configuration:

-student <username> <password>	Initializes the target student of the application.
-dag <amount>	Setups the initial amount of available Data Gathering Agents in the application.
-jadegui	Loads JADE GUI along with the application.

Table 3. Available parameters for configuring the application.

If no arguments has been specified before the launching, the application will be initialized with no target student (the student's data will be required in the login frame), with 10 Data Gathering agents by default and without JADE GUI.

3.3 Workflow of the Application

The workflow of Moodle Helper Application can be presented as a process consisting of two parts:

- The initialization process, which includes the specification of target student and loading the student's profile (**Figure 8**);
- The process of course's data request, where the application gets information about a specific course chosen by the user (**Figure 9**).

The work of the application starts with the launch of all components it needs to work: JADE platform, Moodle Helper Agent and at least one Data Gathering Agent. If the username and a password of target student has been specified beforehand in parameters, the Moodle Helper Agent immediately starts it works. Otherwise, the agent shows login form to the user and waits until the user specify student's username and password for Moodle.

When the agent receives username and password, it starts the loading of student's profile. This process begins with checking the DF Service in order to find at least one Data Gathering Agent what is currently available and can provide a gathering service. When such agent is found, Moodle Helper Agent puts the login data of student whose profile is supposed to be loaded and sends the message to the Data Gathering Agent.

Gathering Data Agents receives the message and starts to load student's profile directly from Moodle using HTTP protocol. When loading is finished, the agent forms the message for the

response, places the gathered information into the message and sends it back to the agent that was requiring this data.

Moodle Helper Agents receives the response, unpacks the information about the student, saves into the model and updates his GUI. Now the information is shown to the user and the application can be used in order to get the detailed description of each course.

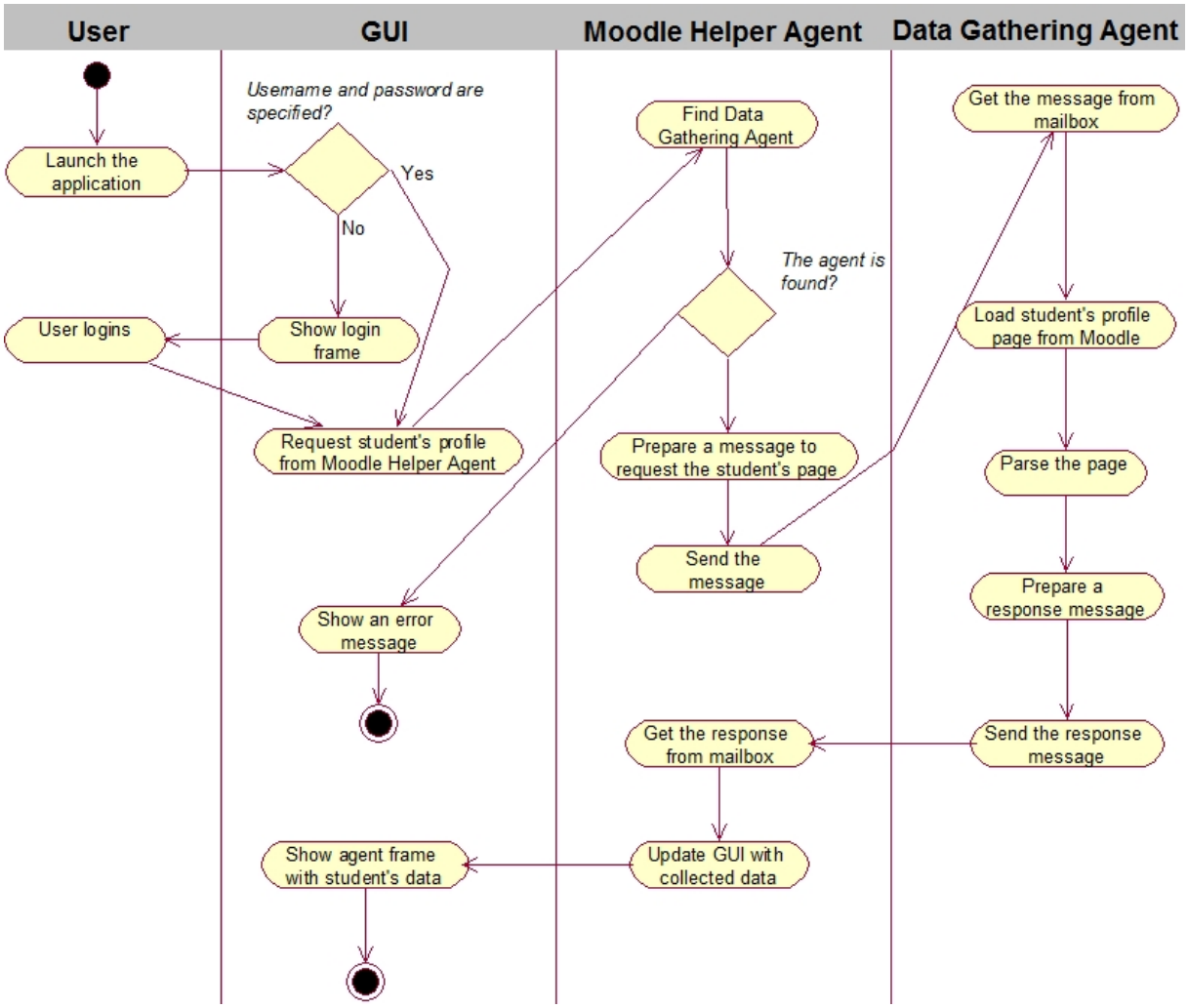

Figure 8. The initialization process.

The next process of gathering information from Moodle begins when the user selects a specific student's course. The list of all student's courses is represented on the respective panel.

The first part of this process is very similar to the request of student's profile. The Moodle Helper Agents sends the request to Data Gathering Agent, which starts collecting the

information and sends it back in the response. For now, Moodle Helper Agent knows the requested course and what elements (tests, tasks and materials) the course contents, but the detailed description of each element is still missing.

So far as there can be a lot of elements and each requires a separate request, this is the place where the advantage of agent-oriented approach is shown. Moodle Helper Agent can request information about each element asynchronously from several Data Gathering Agents. Since the DF service provides information about all available agents, Moodle Helper Agent does not need to know the addresses of all target agents. It also solves the problem with dynamic launching and shutting down of agent during the work of application. Moodle Helper Agents messages for every course's element and sends to the several receivers.

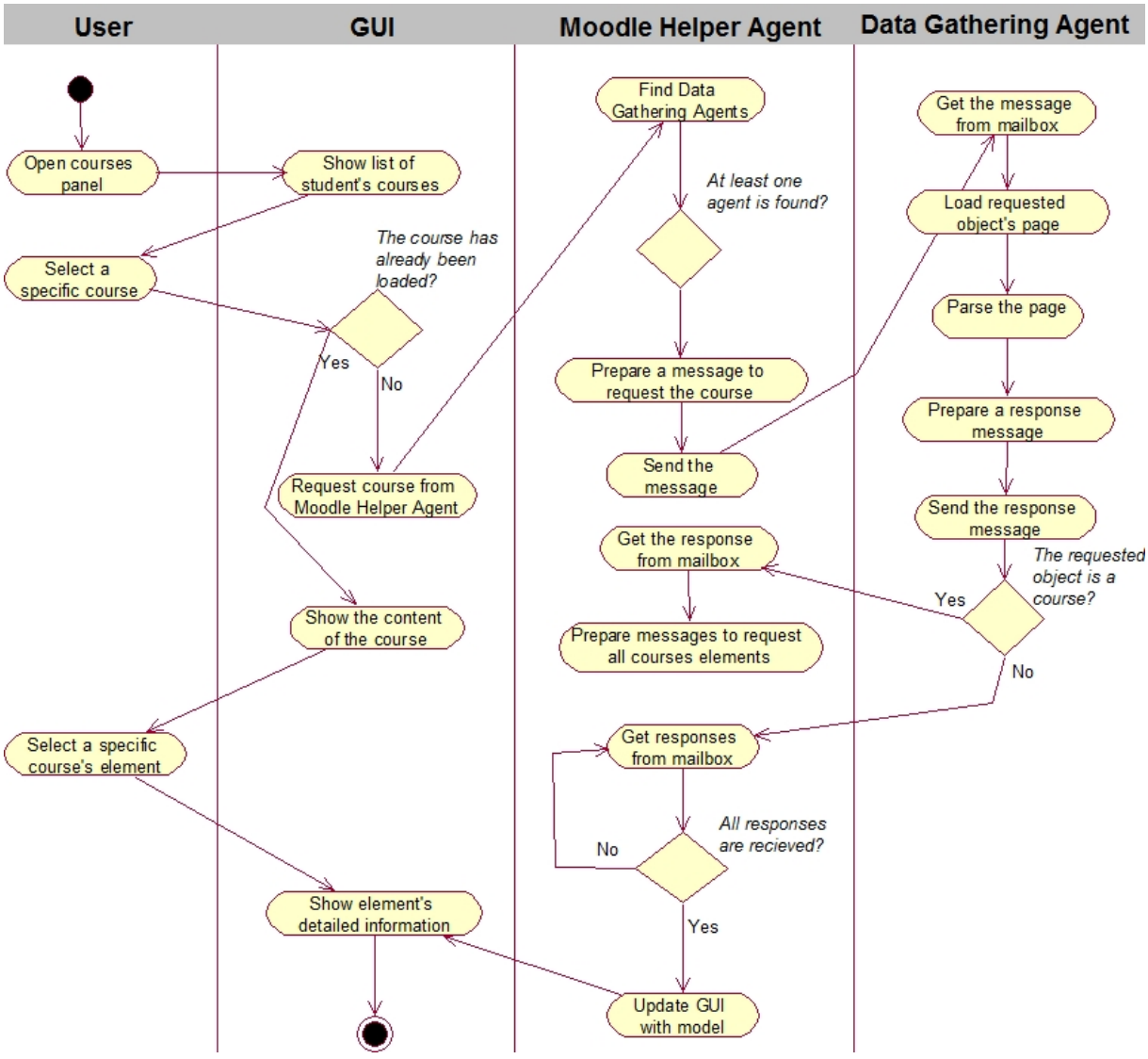

Figure 9. The process of course's data request.

Once one of the responses has arrived, the agent unpacks the information and saves it into the model. When all responses are received, GUI can be updated and the panel with the information about the course is shown to the user. If the user requests the same course once again, the agent does not need to gather information one more time, since it has been stored in the model and can be used immediately without additional requests.

3.4 Results and Analysis

The ultimate result of the work done in this chapter is an agent-based application which is supposed to collect student's data from Moodle. The application uses distributed computations in order to achieve specified task: several agents for gathering student's data are being used in order to get the data from Moodle. The agents can be distributed across the several hosts, but still work together forming the single distributed platform.

As a result, the application shows how the agent-oriented approach can be used in development of application. The using of such paradigm, especially with pre-written frameworks for agent modelling like JADE, makes the development process very simple and the development application flexible. One of the significant benefits of the application is that, as long as it based on the agents, it can be easily integrated into the multi-agent system as a module that provides other agents with required data (for example, in the system that analyzes student's activity and prepares individual curriculums for every student according to received data). The only requirement for such implementation is to be able to communicate by FIPA protocol.

To summarize, AOP can be considered as one of the strongest paradigms in case of implementing the huge distributed systems, which activity is based on continuous messaging between each other. While AOP can be still too complicated for the development of applications which do not require any distributed computation and intercommunication between components, this paradigm should have been considered by developers in case of implementing enormous distributed system.

4. Summary

The aim of this work was the research of the domain of agent-oriented programming and the facilities that this approach provides for application development. The characteristics of AOP has been investigated and provided in the first part of the thesis as well as one of the problems that can be solved using this approach – the development of distributed application.

An agent-based application has been developed during the research process in this thesis to show how can be a simple software product developed using AOP. The application's objective was to collect student's data from Moodle using several agents which can run on different machines and which form a single distributed platform. The application can be used as a part of multi-agents system that needs student's data in order to analyze it and prepare a learning program for the student. The detailed description of the application has been provided in the second part of the thesis.

According to information that has been investigated during the research process of AOP, as well as during the development of the application, it can be concluded that the concept of AOP provides a very tidy solution for the applications which activity can be distributed across the several machines. Although for applications that are not based on continuous messaging between its components and do not need any distribution AOP can be too complicated and resource exhaustive, if the application is based on distributed computing, then the using of AOP can be considered as one of the best solution for this kind of applications.

Kokkuvõte

Töö eesmärgiks oli uurida agentorienteeritud programmeerimise valdkonda ja võimalusi, mida selline lähenemisviis pakkub rakenduste arendamise jaoks. AOP omadused olid uuritud ja antud töö esimeses osas koos ühe probleemiga, mida võib olla lahendatud agentorienteeritud lähenemisviisi abil – jagatud rakenduste arendamine.

Agentide põhinev rakendus oli arendatud selles töös selleks, et näidata, kuidas võib arendada tarkvara kasutades AOP. Rakenduse eesmärgiks oli koguda tudengi andmeid Moodle'st kasutades agente, mis võivad olla käivitatud erinevates masinates, et kujundada jagatud platvormi. See rakendus võib olla kasutatud nagu komponent multi-agentide süsteemis, mis vajab tudengi andmeid, et analüüsida neid ja andmete põhjal valmistada õppekava tudengi jaoks. Rakendus oli detailemalt kirjeldatud töö teises osas.

Uuringud, mis oli tehtud selle töö ja rakenduse arendamise jooksul, näitavad, et võib sedastama, et AOP mõiste pakub väga toredat lahendust sel juhul, kui on vajadus arendada rakendust, mille tegevus tuleb jagada mitmetele masinatele. Kuna tarkvara arendamine, mille jaoks pidev suhtlemine komponentide vahel ja jagatud arvutused ei ole vaja, võib olla arendatud teiste programmerimise metoodikatega lihtsam ja kiirem, kuid kui rakendus on baseerub jagatud komponentidel, siis AOP kasutamist võib nimetada parimaks lagenduseks.

References

1. Y. Shoham. Agent-Oriented Programming (Technical Report STAN-CS-90-1335, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, 1990).

2. Y. Shoham. Software Agents (MIT Press, Cambridge, MA, 1997), pp. 271-290.

3. K. Taveter, A. Norta. Introduction to the Course on Agent-Oriented Modelling and Multi-Agent Systems. Retrieved from: http://maurus.ttu.ee/sts/wp-content/uploads/2015/02/Lecture-1-04-Feb-2015.pdf (15.04.2015)

4. T. Sundsted. An Introduction to Agents. Retrieved from: http://www.javaworld.com/article/2076676/learn-java/an-introduction-to-agents.html (16.04.2015)

5. T. Sundsted. Agent on the Move. Retrieved from: http://www.javaworld.com/article/2076703/learn-java/agents-on-the-move.html (18.04.2015)

6. D. Grimshaw. JADE Administation Tutorial. Retrieved from: http://jade.tilab.com/documentation/tutorials-guides/jade-administration-tutorial/architecture-overview/ (04.05.2015)

7. Steve Phelps. Applying Dependency Injection to Agent-Based Modeling: the JABM Toolkit. Retrieved from: http://www.bracil.net/ccfea/WorkingPapers/2012/CCFEA-WP056-12-r2.pdf (08.05.2015)

8. G. Caire. JADE Programming for Beginners. Retrieved from: http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf (12.05.2015)