Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Communication platform for agents in heterogeneous environment

Diplomová práca

Študijný program: Umelá inteligencia
Študijný odbor: 2503 Kognitívna veda
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: RNDr. Jozef Šiška PhD.

Bratislava 2012

Bc. Michal Vince

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Michal Vince
Študijný program: kognitívna veda (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.11. kognitívna veda
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Communication platform for agents in heterogeneous environment

Cieľ: Dizajn a implementácia komunikačnej platformy pre multi-kontextové systémy v C++, Jave a Pythone. Komunikácia bude založená na FIPA štandarde a bude implementovaná ako RESTFUL protokol cez TCP/IP.

Anotácia: Multi-kontextové systémy sú formalizmom simultánneho usudzovania vo viacerých kontextoch. Rôzne kontexty sú prepojené tzv. mostovými pravidlami, ktoré umožňujú čiastočné mapovanie medzi formulami/konceptami/ informáciami v rôznych kontextoch. FIPA je súčasťou štandardizačnej organizácie IEEE, ktorá podporuje technológie založené na agentoch a interoperabilitu ich štandardov s inými technológiami. Špecifikácia FIPA agentovej komunikácie používa správy napisané v Agentovom komunikačnom jazyku, interakčné protokoly pre výmenu správ, komunikačné akty založené na Teorií rečových aktov a reprezentačný jazyk. Implementácie tejto špecifikácie sú zvyčajne naročné na systémove prostriedky, tým padom neumožňujú vytvárať malých nenáročných agentov. Myslíme si, že je potrebná platforma, ktorá by dokázala spojiť agentov z rôznych platforiem, naprogramovaných v rôznych jazykoch a prepojené iba pomocou komunikačného protokolu. Táto platforma by mala byť schopná bežať aj na malých zariadeniach, napr. PDA.

Vedúci: RNDr. Jozef Šiška, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Dátum zadania: 21.10.2010

Dátum schválenia: 29.11.2010

prof. RNDr. Pavol Zlatoš, PhD.
garant študijného programu

....
študent

....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Michal Vince
Study programme: Cognitive Science (Single degree study, master II. deg., full time form)
Field of Study: 9.2.11. Cognitive Science
Type of Thesis: Diploma Thesis
Language of Thesis: English
Secondary language: Slovak

Title: Communication platform for agents in heterogeneous environment

Aim: Design and implement a communication platform for multi-context systems in C++, Java and Python. Communication will be based on FIPA standard and implemented as a RESTFUL protocol over TCP/IP.

Annotation: Multi-context systems (MCS) are a formalization of simultaneous reasoning in multiple contexts. Different contexts are inter-linked by so called bridge rules which allow for a partial mapping between formulas/concepts/information in different contexts. FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. FIPA Agent Communication specifications deal with Agent Communication Language (ACL) messages, message exchange interaction protocols, speech act theory-based communicative acts and content language representations. Implementations of this specification are often too heavyweight and do not allow to create small lightweight agents There is a need for a platform that can connect agents from different platforms, programmed in different languages, connected only by communication protocol. A platform that is portable and lightweight enough to run even on small devices, e.g. PDA.

Supervisor: RNDr. Jozef Šiška, PhD.
Department: FMFI.KAI - Department of Applied Informatics

Assigned: 21.10.2010

Approved: 29.11.2010 prof. RNDr. Pavol Zlatoš, PhD.
Guarantor of Study Programme

....

Student Supervisor

Poďakovanie

Moja vďaka patrí vedúcemu práce RNDr. Jozefovi Šiškovi, PhD. za cenné rady a poskytnutú podporu. Vďaka patrí aj RNDr. Andrejovi Lúčnemu, PhD. za jeho neoceniteľné rady z oblasti multi-agentových systémov, môjmu priateľovi a kolegovi Jánovi Šilarovi za jeho pomoc v oblasti filozofie jazyka. V neposlednej rade by som sa chcel poďakovať RNDr. Kristíne Rebrovej, Veronike Tučekovej, Mgr. Matejovi Vincemu ako aj celej mojej rodine za podporu a pomoc, ktorú mi poskytovali počas vzniku tejto práce a počas celého môjho štúdia.

Abstrakt

Cieľom tejto práce je implementácia komunikačnej platformy pre agentov, ktorí sa nachádzajú v rôznych prostrediach. V tejto práci ponúkame implementáciu multi-agentového middleware-u, ktorý je založený na FIPA implementáciách. Aby sme zabezpečili beh agentov v rôznych prostrediach, náš middleware sme implementovali v troch rôznych jazykoch, konkrétne C++, Jave a Pythone. Architektúra celej platformy je inšpirovaná FIPA špecifikáciami, tak ako aj jednotlivé komponenty. Pre posielanie správ používame FIPA HTTP protokol pre prenos správ, ktorý sme rozšírili pre potreby vyzdvihovania správ. Použitím tohto protokolu, ktorý sám o sebe spĺňa REST požiadavky, sme zabezpečili schopnosť agentov komunikovať aj s agentami z iných platforiem. Samotné správy sú založené na jazyku FIPA-ACL, ktorý je inšpirovaný Teóriou rečových aktov. Platforma ma nízke systémové požiadavky a je možné ju používať na rôznych mobilných zariadeniach. Práve kvôli využitiu jazyka FIPA-ACL a nízkym systémovým požiadavkam platformy sa tento middleware stáva vhodným nástrojom pre výskum v oblasti kognitívnej vedy.

Klúčové slová: multi-agentový systém, Teória rečových aktov, Agent Communication Language (ACL), FIPA, REST, Python, Java, C++.

Abstract

The aim of this thesis is to implement a communication platform for agents in heterogeneous environment. In this thesis we propose an implementation of a multi-agent middleware, that is implemented according to FIPA specifications. To ensure agents are able to run in heterogeneous environment, we implemented this middleware in three popular languages, C++, Java and Python. The architecture of the platform is inspired by the FIPA Abstract Architecture and so are other components of the platform. We are using FIPA HTTP protocol for sending messages. We extended this protocol for the purpose of message polling. By using this HTTP protocol, which is REST-ful, we ensure that agents from our platform are able to communicate with agents from other FIPA compliant platforms. Messages are using language that is inspired the Speech Act Theory, FIPA ACL language. The platform has low system requirements and is able to run on different mobile devices. Because of the low system requirements and the use of FIPA ACL language this middleware is a suitable tool for cognitive science surveys.

Key words: multi-agent system, Speech Act Theory, Agent Communication Language (ACL), FIPA, REST, Python, Java, C++.

Contents

Introduction **x**

1 Motivation **1**

2 Theoretical foundations **2**

2.1 Intelligent agent 2
2.2 Multi-agent systems 4
2.2.1 Short history 4
2.2.2 Types of Multi-agent systems 5
2.2.3 Practical use of MAS 9
2.2.4 MAS in Slovakia 9
2.3 Communication in MAS 10
2.4 Speech Act Theory 12
2.4.1 Austin's speech act theory 12
2.4.2 Searle's Speech acts 14
2.4.3 Searle's Expression and Meaning 15
2.5 Agent Communication Language 18
2.5.1 Generalized ACL framework 18
2.5.2 KQML 20
2.5.3 Arcol 21
2.5.4 FIPA-ACL 23

3 Technical foundations **25**

3.1 REST 25
3.2 FIPA 27
3.2.1 Agent Management Specification 27
3.2.2 Message transport service 29
3.2.3 Agent Communication Language Message 30
3.2.4 Content language 31
3.3 Event-driven programming 32
3.4 Singleton design pattern 33

3.5 Marshalling/Demarshalling . . . 34

4 Problem description **35**

4.1 Platform dependency . . . 35

4.2 Centralization . . . 37

4.3 Direct communication . . . 37

4.4 ACL in MAS . . . 38

4.5 System requirements . . . 38

5 Design and Analysis of the Solution **40**

5.1 Abstract Architecture . . . 40

5.1.1 FIPA inspired . . . 42

5.1.2 Heterogeneity . . . 43

5.2 Agent . . . 43

5.3 Message Transport Service . . . 44

5.3.1 Message polling . . . 44

5.4 Discovery Service . . . 47

5.5 Platform . . . 48

6 Implementation **49**

6.1 Python implementation . . . 49

6.1.1 Agent . . . 49

6.1.2 Discovery Service . . . 50

6.1.3 IOServer . . . 50

6.2 Java implementation . . . 50

6.2.1 jAgent . . . 51

6.2.2 IOServer . . . 51

6.2.3 Discovery Service . . . 51

6.3 C++ implementation . . . 52

6.3.1 Threading . . . 52

6.3.2 IOServer . . . 52

6.3.3 Discovery Service . . . 53

6.3.4 Platform . . . 53

7 Conclusion **54**

7.1 Summary . . . 54

7.2 Future work . . . 54

Bibliography **56**

A Extended FIPA ACL Message Representation in XML Specification **62**

A Homepage of the project **66**

Introduction

In the past, many different multi-agent platforms have been created, so another one may seem redundant. However, new multi-agent platforms are still important. Older ones are usually designed for computers, but every year many new devices with different specification are created and many new technologies invented. For these platforms it becomes difficult to deal with specific devices with low system specification.

New multi-agent platforms can be created with these limitations in mind and for their implementation newest technology can be used. Only a few of these platforms will eventually become popular and widely used, but any of them might eventually turn out to be important for future research.

Goals

The main goal of this thesis is to design and implement new multi-agent middleware. This platform should be able to connect agents from different environments. Because of this, it should be implemented in several different programming languages. The communication between the agents should be REST-ful and implemented over the TCP/IP protocol.

Structure of the thesis

This thesis is divided into seven chapters. The **Motivation** chapter describes our motivation for creating such a middleware. The next two chapters, **Theoretical** and **Technical foundations**, are designed to be independent. They provide the foundation for remaining chapters by describing the current state of respective fields and can thus be skipped by people who are familiar with them.

Following chapters summarize problems of current multi-agent middlewares (**Problem description**), describe the architecture of our solution (**Design and analysis of the solution**) and the process of implementation (**Implementation**).

Finally, the **Conclusion** chapter briefly recapitulates the results of this thesis and our future plans with this middleware.

Chapter 1

Motivation

Small and smart devices are becoming part of man's everyday life. It is not so long time ago when people did not know what Internet is. Nowadays, most of them can not imagine their lives without it. People are creating and adapting different devices so they can keep them close and online. These devices help them to solve everyday tasks or just keep them entertained during their spare time.

These devices possess invaluable information about their owners, they know their habits, what kind of music they like, their favorite places to go, who are they friends with and what they have in common, and many others. Possessing such information about somebody gives the opportunity to deduce his personality, what his mood is or even predict his future actions.

Big Internet companies, such as Google, recognized this trend at the very beginning and started to use these information in their own advantage. On the other hand, it looks like the cognitive science audience have failed to catch this early train. In 2012 G. Miller in his article [48] summarized the use of smart phones in different surveys and predicts the future development of these surveys. Although first surveys using smart devices date back to year 2001, they are just becoming more widespread in year 2010. Until this day, the scientific community did not offer any solution to connect these devices, nor grant them the ability to make agreements.

We think, that the age of modern smart devices will develop into the age of ambient intelligence, where devices work in concert to support people in carrying out their everyday life activities, tasks and rituals in easy, natural way using information and intelligence that is hidden in the network connecting these devices [73]. One of the possibilities how to connect these devices is a multi-agent middleware. In this work we offer such a solution.

Chapter 2

Theoretical foundations

In order to help the reader to fully understand what we are doing and how we are trying to achieve it, first, we need to discuss some theoretical background of this thesis. In this chapter we provide the definition of an intelligent agent, short history of multi-agent systems and their classifications. Later, we summarize the use of multi-agent systems both in the world and in Slovakia. In the second part of this chapter we pay special attention to the communication between agents in multi-agent systems. We will provide brief description of the Speech Act Theory founded by J. L. Austin and later adapted by J. Searle. In the last section we summarize few agent communication languages used in multi-agent systems based on these theories.

2.1 Intelligent agent

Although the concept of an intelligent agent is vaguely used in many fields, Russell and Norvig offer the following definition:

Definition An intelligent agent is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals (i.e. it is rational).[58]

A simple agent program can be defined mathematically as an agent function. Agent function is an abstract concept as it could incorporate various principles of decision making like calculation of utility of individual options, deduction over logic rules, fuzzy logic, etc.[58] Intelligent agents can be very simple (e.g. thermostat), but also very complex (e.g. model of a human being). Russell and Norvig divide agents into five classes according to their degree of perceived intelligence and capability:

- **Simple reflex agents**.
  This is the simplest kind of an agent, selecting actions based on the current percept, ignoring the rest of the percept history. The agent function is based on the

condition-action rule: if condition then action. This agent function only succeeds in fully observable environment. Therefore, infinite loops are often unavoidable in partially observable environments (Figure 2.1).

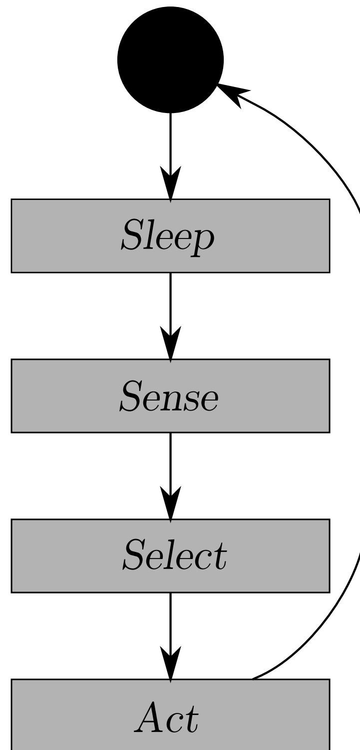


Figure 2.1: State chart diagram of the simple reflex agent's life cycle in partially observable environment.

- **Model-based reflex agents**.
  This class can handle a partially observable environment, by creating internal **model of the world**, which depends on the percept history and thereby reflects at least some of the unobservable aspects of the current state. The action is then chosen in the same way as in simple reflex agent.

- **Goal-based agents**.
  They expand the capabilities of model-based agents by introducing the goal information. This information describes desired situation, thus agents are able to select actions that lead to this desired state. The selection of proper action sequence is possible through planning and search (sub-fields of AI).

- **Utility-based agents**.
  Goal-based agents can only distinguish between two types of states, goal states and non-goal states. Utility-based agents can measure how desirable the state is according to the utility function. This allows agents to compare states (i.e. how

much would they gain/lose by getting to the particular state), therefore help the agents to select better action sequences, as rational agents are trying to maximize the expected utility.

- **Learning agents**.
  Learning agents are able to operate in unknown environments and by learning become more competent. A learning agent is composed of the learning element, responsible for making improvements; and the performance element, responsible for selecting actions; and the problem generator, responsible for suggesting actions that will lead to new experiences. The learning element uses the feedback from the environment to improve the performance element, ensuring that the performance element will select better actions in the future.

2.2 Multi-agent systems

Multi-agent system is a system composed of multiple interacting intelligent agents and their environment. They can be used to solve problems that are difficult or impossible for an individual agent. According to Wooldridge[71], and Huhns and Stephens[32], agents in a multi-agent system have several important characteristics:

- **Decentralization:** There is no designated controlling agent, which means, that agents have to cooperate in order to achieve the goal.
- **Local views:** Either the multi-agent system is too complex for an agent to make use of the information of a global state or no agent has a global view of the system.
- **Autonomy:** Agents are at least partially autonomous.
- **Interaction:** Agents are able to interact with other agents in the system.

2.2.1 Short history

Multi-agent systems are a relatively new sub-field of computer science, the study about them began in the field of distributed artificial intelligence about 20 years ago and the field gained widespread recognition probably with the 37th issue of the Communication of the ACM in July 1994. In this issue different streams of multi-agent designs and architectures were presented, out of which Michael Genesereth represented the current main stream. All of this resulted in FIPA 97 (The Foundation for Intelligent Physical Agents), the initiative to produce software standards specifications for heterogeneous and interacting agents and agent based systems.

Since then, international interest in the field has grown enormously, because of the belief that agents are an appropriate software paradigm through which to exploit possibilities presented by massive open distributed systems - such as the Internet. Nevertheless, multiagent systems seem to be a natural metaphor for understanding and building a wide range of what we might crudely call artificial social system[71].

The field of multi-agent systems is highly interdisciplinary, as it takes inspiration from different areas (philosophy, logic, social science, etc.), but despite all of this, multi-agent systems did not take the lead in the artificial intelligence. Furthermore, some of the people in the computer science community are rather skeptical, whether multi-agent system is not just a distributed system/concurrent system, economics/game theory, social science etc. Wooldridge is trying to satisfy these skeptical voices in his book An Introduction to Multi-Agent systems. We let the reader to make his own opinion, but for the purpose of this work we are going to consider it an independent field.

2.2.2 Types of Multi-agent systems

There are lots of different classifications of multi-agent systems: classifications based on the type of agents, on the type of environments where agents are based, the complexity of agents, etc.

Classification of environments

Russel and Norvig suggest following classification of environment properties[58]:

- **Fully observable vs. partially observable**
  A fully observable environment is one in which the agent can obtain complete, accurate, up-to date information about the environment's state. Most of the real-world environments are not fully observable.

- **Deterministic vs. stochastic**
  Deterministic environment is one in which any action has a single guaranteed effect, on the other hand, in stochastic environment there is uncertainty about the state that will result from performing an action.

- **Static vs. dynamic**
  A static environment is one that is changing by the performance of an action by the agent, otherwise it remains unchanged. Dynamic environment, however, changes in ways beyond the agent's control (e.g. the Internet).

- **Discrete vs. continuous**
  If there is a finite number of distinct states, actions, percepts, etc. we are talking about discrete environment.

Classification according to the type of interaction

One of the most important classification of multi-agent system is based on the type of interaction between the agents[50]:

- **direct interaction between agents** - agents are communicating directly with each other, and
- **indirect interaction between agents** where agents are communicating through environment (e.g. ants with their pheromones).

Classification of agents

Researchers were trying to classify the agents throughout these two decades, we can summarize their attempts and knowledge from [72, 71, 26, 50, 32] into these classes:

- **Deliberate** - logic-based agents. Often implementing AI techniques such as planning, neural networks, etc.
- **Simple reactive agents** - with no AI techniques, simply reacting to the inputs.
- **Hybrid agents** - a mixture of the two classes mentioned above.

Types of implementation of MAS

When implementing a multi-agent system, a designer needs to choose what kind of environment he will situate it in. There are three types of environments:

- **distributed environment**,
- **multi-threaded environment** and
- **multi-processor environment**,

For us, the most important from these is the distributed environment. An implementation of multi-agent system in distributed environment means that agents run on different computers, which are connected to some **local area network (LAN)** or even to a **wide area network (WAN)**. Since agents need to communicate with each other across a computer network they need to be programmed in such a way, this is so called **network programming**. The functions of communication system are grouped into logical layers in the **Open System Interconnection (OSI) model**, which was produced by the Open System Interconnection effort at the **International Organization for Standardization (ISO)**. There are 7 layers arranged on top of each other (Figure 2.2), each layer using the layer below current, and serving the layer above current. Protocols on the same layer can interact with each other (Figure 2.3).

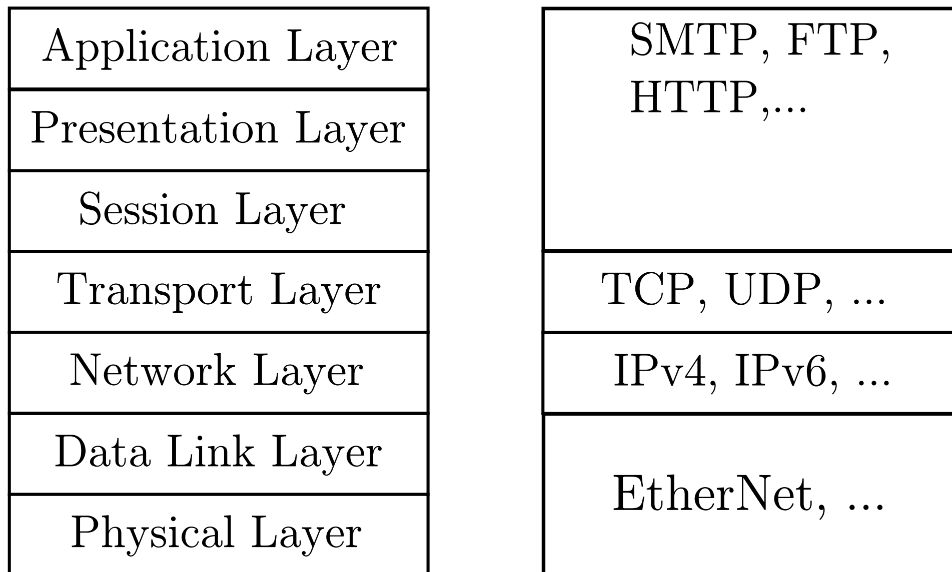


Figure 2.2: Open System Interconnection model

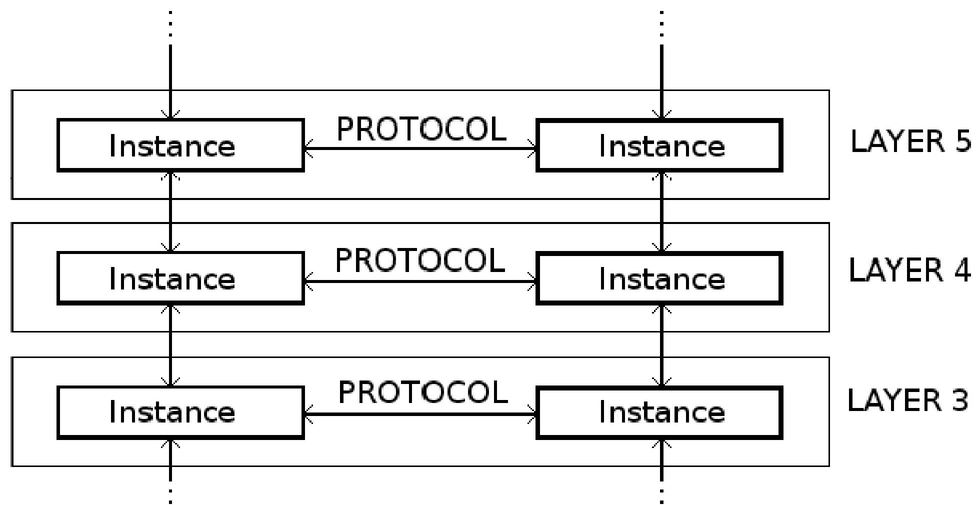


Figure 2.3: Communication in the OSI-Model

Layer 1: physical layer is the lowest layer and it defines electrical and physical specification for devices, relationship between a device and a transmission medium, such as a copper and fiber optical cable (including voltages, cable specifications, hubs, repeaters, etc.). Its main function is establishment and termination of a connection to a communications medium and media, signal and binary transmission. The data unit of this layer is **Bit**.

Layer 2: data link layer is responsible for physical addressing (e.g. MAC, LCC, etc.) and provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the physical layer. The data unit used in this layer is **Frame**.

Layer 3: network layer provides transferring variable length data sequences from source host on the network to a destination host on a different network, while maintaining the quality of service requested by the transport layer. The main function of the network layer is path determination and logical addressing. The main data unit is **Packet** or **Diagram** and protocols providing such services as this layer are for example IPv4, IPv6, etc.
Layer 4: transport layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The transport layer also provides the acknowledgment of the successful data transmission and sends the next data if no error occurred. The data unit in this layer is **Segment** and protocols are TC, USP, etc.
Layer 5: session layer controls the connections between computers, establishes, manages and terminates the connections between the local and remote application. It provides for full-duplex, half-duplex, or simplex operation, establishes check-pointing, adjournment, termination, and restart procedures. It is commonly implemented applications that use **remote procedure calls** (CORBA, RMI, SOAP, etc.) and data unit is **Data**.
Layer 6: presentation layer establishes context between application-layer entities, in which the higher-layer entities may use different syntax and semantics if the presentation service provides a mapping between them. It is also responsible for encrypting and decrypting data and conversion from machine dependent data to machine independent data (e.g. serialization of objects and other data structures from and to XML).
Layer 7: application layer is the layer closest to the end user, both the user and this layer interact directly with the software application. Functions of application layer typically include identifying communication partners, determining resources availability, and synchronizing communication. Examples of application layer implementations are Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), etc.
The implementation of a multi-agent system needs to implement all seven layers, however, there are multi-agent platforms already implementing communication tier (layers 1,2,3 and 4) and partially implementing middle tier (layers 5 and 6), therefore the name **middleware**. Middle tier in a middleware can be implemented in various ways [45]:

- **as distributed objects** - sometimes implemented by Remote Method Invocation (RMI), Portable Object Adapter (POA) in CORBA, etc,
- **as services** - using CORBA, Remote Procedure Call (RPC), etc,
- **as transactions** - using SQL,

- **as message passing** - implemented by Simple Object Access Protocol (SOAP) and others.

2.2.3 Practical use of MAS

Over decades, researches found many different ways of application of multi-agent systems in order to solve various problems in various fields of science:

- **The SEWASIE** (SEmantic Webs and AgentS in Integrated Economies) project is a European research project that aims at designing and implementing an advanced search engine enabling intelligent access to heterogeneous data sources on the web, in a rich semantic (ontological) framework.[63, 14]
- Studying the behavior of agent-based methods over varying levels of uncertainty in comparison to traditional optimization methods[46].
- A Multi-agents based fault diagnosis reference model for MSW incineration process[70].
- **MAS/LUCC** - Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change[51].
- **Solving constraint satisfaction problems[54].**
- **The STAFF** real-time simulator uses an adaptive model for flood forecasting, which is composed of two levels of self-organizing multi-agent systems [27].

2.2.4 MAS in Slovakia

Multi-agent systems were not and are not vastly used neither in business nor in the scientific field in Slovakia. Perhaps the first book related to agents was written by J. Kelemen. In his book, Strojovia a agenty [37], he tries to explain differences between the terms **living machines** and **nonliving agents** and their common meaning in the ordinary world. In this book, we can find topics from several fields, like philosophy, cognitive or computer science and not just artificial intelligence.

To our knowledge, only two companies in Slovakia used a multi-agent system in the last few decades. Both of them for a different reason.
In year 2004 A. Lúčny presented 'Agent-Space Architecture' [43, 44], which is multi-agent architecture with indirect interaction between the agents, through another entity called **space**, agents within this system are **reactive agents**. Agents can read and write unlimited data to and from the space, furthermore, these messages can not be hidden from any agent - messages are readable and writable by every agent in the system. Sometimes this type of messages is called **stigma** and this kind of data exchange **the**

stigmergic communication[43]. Lúčny's employer, the company MicroStep-MIS[1], adapted this architecture and used it in several projects such as Integrated Meteorological System, Unified Data Collection System for Q-network - serves for collecting data from hydro meteorological stations of different kinds, etc[44].
Systems built over this architecture have several significant features[43]:

- no deadlocks,
- recovery from errors,
- scalability,
- configurability,
- modifiability.

It is up to designer to make use of these features correctly and effectively.

In 1999, few years after FIPA specification was introduced, F. Bellifemine, G. Rimassa, J. Pitt, A. Poggi started to implement **Java Agent DEvelopment Framework - JADE**, which is FIPA compliant agent framework[5, 55]. In May 2003 TILAB[2] and Motorola launched a new initiative, the JADE Governing Board, a not-for-profit organization, with the intent of promoting the evolution and the adoption of JADE by the mobile telecommunications industries as a java-based de-facto standard middleware for agent-based applications in the mobile personal communication sector. In 2004, G. Rimassa became an employer of Whitestein Technologies AG[3], company with office in Slovakia. On the other hand, Whitestein became a member of the FIPA committee and a member of the JADE Governing Board.

Whitestein used JADE for prototypes of their projects, mainly because its persistence, as they are oriented on large and robust business application.

2.3 Communication in MAS

Communication between agents is the most important feature in a multi-agent system. Without the ability to communicate, agents would not be able to coordinate, cooperate nor inform other agents, thus achieving goals will become very hard or even impossible. In other words, a multi-agent system without the communication between agents is simply not a multi-agent system at all. Perhaps, the characteristic problem in communicating concurrent systems research is that of synchronizing multiple processes, which was widely studied throughout the 1970s and 1980s [6]. Essentially, two

[1] http://www.microstep-mis.com/
[2] http://jade.tilab.com/
[3] http://www.whitestein.com/

processes (agents) need to be synchronized if there is a possibility that they can interfere with one another in a destructive way. The classic example of such interference is the *lost update scenario* [71].

However, in a multi-agent system, we don't treat communication in such low-level way, but in more *agent-oriented* way. As we mentioned before, agents within multi-agent systems are autonomous, which means agents have control over their state and behavior. **The agent can not *force* an other agent to change his state, nor to modify his behavior**(e.g. to perform some action) without agent's agreement. Because of that agents need to try to **influence** other agents to achieve intended goals. A careful reader already sees the connection between this **coordination** of agents in multi-agent systems and humans in the real-world. Agents behave much like humans in order to achieve their goals.
Huhns and Stephens [32] define coordination as the property of a system of agents performing some activity in a shared environment. They divided the coordination between the agents into:

- the coordination of agents with the **same shared goal**; and
- the coordination of agents with **different goals** or the same, but **not shared goal**.

Cooperation is the name for the coordination of the first group of agents, **competition** for the latter, in terms of the quote: 'If it's not serving my purposes, it's against them'. Agents with the shared goal usually **plan** their actions in order to achieve the goal. This planning can be done in a **centralized** or in a **distributed** way.
On the other hand, in order to achieve the goal when other agents are competitors, or

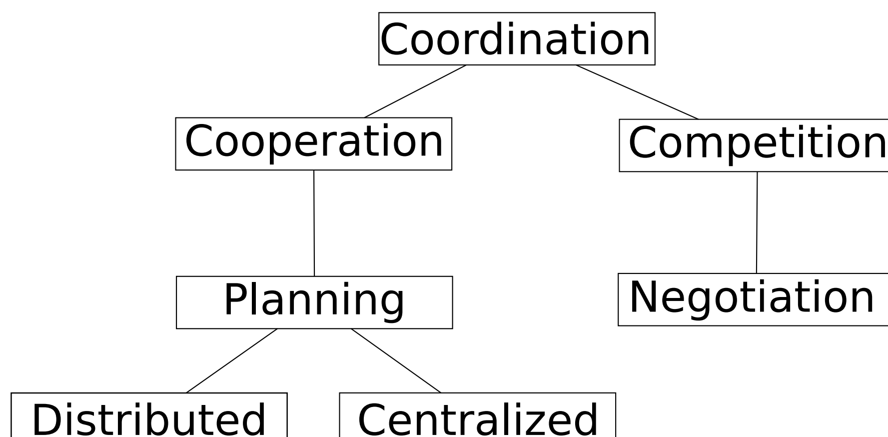


Figure 2.4: A classification of some of the different ways in which agents can coordinate their behavior and activities.

are simply not willing to help, agent needs to **negotiate**. This negotiation need not to

be successful (Figure 2.4). But to allow agents to perform such actions as negotiation, the **Speech Act Theory** needs to be implemented in a multi-agent system. Spoken human communication is then used as a model for communication among agents.

2.4 Speech Act Theory

Definition Speech act, in linguistics, is an utterance defined in terms of a speaker's intention and the effect it has on a listener.

The contemporary use of the term goes back to J. L. Austin discovery of **performative utterances** and his theory of **locutionary, illocutionary, and perlocutionary acts**.[4]

2.4.1 Austin's speech act theory

J. L. Austin formulated his Speech Act Theory in his lectures at Oxford University and issued it in 1963 in the book named 'How to Do Things with Words'. Austin investigates language from a pragmatic point of view, i.e. he analyses complete speech act, not just isolated utterances.
In his analysis of a speech act he considers statements, as well as other types of expressions and complex situations, in which the utterance was made. He presents the concept of **Performative utterances**. The name itself indicates, that performing an utterance is an act and this act, in addition, is not just an act of making a statement. On the other hand, he introduces **Constative utterances**, as the opposite of performative utterances. Making a constative utterance, with reference to the past, means, according to Austin, making of a statement. Making of a performative utterance, means e.g. to make a bet. To successfully accomplish a performative, merely utter the right words is not enough. Besides the uttering of the words of the so-called performative, a good many other things have as a general rule to be right and to go right if the action is to be said to have happily brought off. What these are, Austin is hoping to discover by looking at and classifying types of cases in which something goes wrong and the act is therefore at least to some extent a failure. The utterance is then not indeed false but in general *unhappy* [34].
For this reason he calls the doctrine of the things that can be and go wrong on the occasion of such utterances, **the doctrine of the Infelicities**. He created the scheme of things which are necessary for the smooth or *happy* functioning of a performative, consists of these assumptions:

- (A.1) There must exist an accepted conventional procedure having a certain conventional effect, that procedure to include the uttering of certain words by certain persons in certain circumstances, and further,

- (A.2) the particular persons and circumstances in a given case must be appropriate for the invocation of the particular procedure invoked.
- (B.1) The procedure must be executed by all participants both correctly and
- (B.2) completely.
- (Γ.1) Where, as often, the procedure is designed for use by persons having certain thoughts or feelings, or for the inauguration of certain consequential conduct on the part of any participant, then a person participating in and so invoking the procedure must in fact have those thoughts or feelings, and the participants must intend so to conduct themselves, and further
- (Γ.2) must actually so conduct themselves subsequently.

Breaking anyone (or more) of these six rules, the performative utterance will be in *unhappy*. Of course, there are considerable differences between these, which we can divide into **misfires**(group A and B) and **abuses**(group Γ).
By misfire, Austin means the procedure which was purported to invoke is disallowed or is botched, and the act is void or without effect. Further, he labels the A cases as **misinvocations** of a procedure, either because there is no such procedure, or because the procedure in question cannot be made to apply in the way attempted. B cases, **misexecutions**, where the purported act is *vitiated* by a **flaw** (B.1) or **hitch** in the conduct of the ceremony (B.2).
Infelicitous acts, abuses, are acts professed but hollow, and not implemented, or not consummated. Austin uses terms **insincerities** for Γ.1 and **Non-exemptions, Non-fulfilments, Disloyalties, etc** for Γ.2.

Austin considers three cases of relations between statements and performative utterance, **the locutionary act**, **the illocutionary act** and **the perlocutionary act**.
The act of 'saying something' in this full normal sense he calls the performance of a locutionary act, and the study of utterances thus far and in these respects **the study of locution**, or of the full units of speech. His interest in the locutionary act is to principally make plain, what it is, in order to distinguish it from other acts. The locutionary act is roughly equal to the making of a statement, with some meaning and reference.
To perform a locutionary act is in general, also and on its own, to perform an illocutionary act. The performance of an illocutionary act, i.e. the performance of an act *in* saying something as opposed to performance of an act *of* saying something. Acts, widely accepted as illocutionary, are for example promising, ordering someone, warning someone, etc.
Saying something will often produce certain consequential effects upon the feelings,

thoughts, or actions of the audience, or of the speaker, or of other persons. These effects may be evoked by the design, intention, or purpose of producing them. The performance of such an act is the performance of a **perlocutionary act**.

By this, Austin, created new approach which changed the perception of the language. This approach explores language from a pragmatic point of view, and by the theory of complex acts he denies the reductionist approach of understanding of individual components of a language. It is a theory of specific acts of a specific species.[69]

2.4.2 Searle's Speech acts

Searle's speech acts are inspired by the work of few different philosophers, among others, his former teacher J. L. Austin. According to Searle, speech act is every use of acoustic utterance in a common speech situation. In general, speech act is everything what one can formulate. Also, thought satisfies this, although it is not uttered, but it can be expressed in some form (written text, sign language, etc.).[36]
Searle favors an Austinian approach that emphasizes the speech act as the basic unit of meaning and communication, and which sees speaking a language as engaging in a rule-governed form of behavior.[30] He divides speech acts into four groups:

- **utterance acts** - an actual utterance of words (morphemes, sentences, ...),
- **referring and predicating** - performing propositional acts,
- **illocutionary acts** - stating, questioning, commanding, promising, etc.,
- **perlocutionary acts** - persuade someone by arguing, scare someone by warning him, etc.

Illocutionary acts are minimal units of linguistic communication. Every time we speak to someone, or communicate in some other form, we make illocutionary acts. Illocutionary acts are intentional, there is no need to promise something, if we are not to keep the promise. In the performance of an illocutionary act in the literal utterance of a sentence, the speaker intends to produce a certain effect by means of getting the hearer to recognize his intention to produce that effect; and furthermore, if he is using words literally, he intends this recognition to be achieved in virtue of the fact that the rules for using the expressions he utters associate the expression with the production of that effect. Hearers have to recognize a speaker's intention by recognizing the meaning of an utterance, bounded by semantic rules. Illocutionary acts are the units bearing the meaning in communication.
On the other hand, perlocutionary acts are the effects or the consequences of illocutionary acts, on the actions, thoughts or beliefs of hearers. Perlocutionary acts

are not necessarily intended. To summarize, **illocutionary act is an intention of the speaker and perlocutionary act are the consequences taken on hearer's side**.[61]

2.4.3 Searle's Expression and Meaning

In 1979 Searle improves his theory of speech acts in his book Expression and Meaning: Studies in the Theory of Speech Acts.[62]
Searle introduces three dimensions in order to classify speech acts:

- The **illocutionary point** (or purpose) of the (type of) speech act, i. e. what the speaker wants to achieve by performing a speech act. For example, the illocutionary point of a request of a promise is that it is an undertaking of an obligation by the speaker to do something.

- The **direction of fit** of a speech act is describing the relationship of the propositional contents and the referred world. The direction of some illocutionary points is to get the content to match the world (e.g. assertions), the direction of the others is to get the world to match the words (e.g. promises, requests). Searle is trying to illustrate this distinction by an illustration provided by E. Anscombe in her monograph Intention[2].
  Suppose a man goes to the supermarket with a shopping list given him by his wife on which are written the words 'beans, butter, bacon, and bread'. Suppose as he goes around the supermarket with his shopping cart selecting these items, he is followed by a detective who writes down everything he takes. As they emerge from the store both shopper and detective will have identical lists. However, as Searle points out, the function of the two lists will be quite different. In the case of the shopper's list, the purpose of the list is, to get the world to match the words; the shopper is supposed to make his actions fit the list. In the case of the detective, the purpose of the list is to make the words match the world; the detective is supposed to make the list fit the actions of the shopper.

- The **sincerity condition** of a speech act is the psychological state expressed in the performance of the illocutionary act. In other words, it is the psychological attitude of the speaker towards the propositional content.[11] For example, in case of a request for an action, the speaker expresses a want that the hearer performs the action.

On the basis of these three dimensions, Searle proposes new classes of speech acts.

- **Assertives**.
  The illocutionary point of these acts is to commit the hearer about the truth

of the expressed proposition. The direction of fit is words-to-world, speaker is trying to convince by his words about the truth of the state in the real world, and the sincerity condition is 'belief that p'. Using Frege's assertion sign to mark the illocutionary point common to all the members, Searle symbolizes this class as follows:

$$\vdash\downarrow B(p)$$

$$\downarrow\ - words - to - world\ direction$$

Examples of assertives are 'There is a man in the house.' or 'It is snowing'.

- **Directives**.
  The illocutionary point of members of this category consists in the fact that they are attempts by the speaker to get the hearer to do something, expressed in the propositional content. The direction of fit is world-to-words, and the sincerity condition is 'want' or 'wish' or 'desire'. Searle is uses the *shriek mark* for the illocutionary point indicating device for the members of this class generally:

  $$!\uparrow W(H\ does\ A)$$

  $$\uparrow\ - world - to - words\ direction$$

  Verbs denoting members of this class are, for example, *ask, order, command, request, beg, plead, pray, entreat, invite, etc.* According to Searle, questions are a subclass of this class, since they are attempts by the speaker to get the hearer to answer, i.e. to perform a speech act.

- **Commissives**.
  Illocutionary point of these acts is to commit the speaker to some future course of action. Using 'C' for the members of this class generally, Searle proposes following symbolism:

  $$C\uparrow I(S\ does\ A)$$

  $$\uparrow\ - world - to - words\ direction$$

  The direction of fit is world-to-words and the sincerity condition is Intention. The propositional content is always that the speaker S does some future action A.

- **Expressives**.
  The illocutionary point of this class is to express the psychological state specified in the sincerity condition about a state of affairs specified in the propositional content. Examples of expressive verbs are *apologize, congratulate, condole, deplore, etc.* In expressives there is no direction of fit, as the speaker is neither trying to get the world match the words nor the words to match the world, rather the truth of the expressed preposition is presupposed. Searle is giving an example, that when I apologize for having stepped on your toe, it is not mu purpose either to claim that your toe was stepped on nor to get it stepped on. Therefore, the truth of the proposition expressed in an expressive is presupposed. The symbolization of this class is:

  $$E\varnothing(P)(S|H\ +\ property)$$

  In this symbolization 'E' indicates the illocutionary point common to all expressives, '$\varnothing$'is the null symbol indicating no direction of fit, P is a variable ranging over the different possible psychological states expressed in the performance of the illocutionary acts in this class, and the propositional content ascribes some property to either speaker or hearer. The property specified in the propositional content of an expressive must, however, be related to the speaker or the hearer.

- **Declarations**.
  According to Searle, it is the defining characteristic of this class that the successful performance of one of its members brings about the correspondence between the propositional content and reality, successful performance guarantees that the propositional content corresponds to the world, e.g. if the act of appointing someone chairman is performed successfully, the he is a chairman, etc. He symbolizes the structure of declarations as follows:

  $$D \updownarrow \varnothing(p)$$

  'D' indicates the declarational illocutionary point, the direction of fit is both words-to-world and world-to-words because of the peculiar character of declarations. There is no sincerity condition and propositional variable p. Searle distinguished a particular subclass of declarations, which he calls assertive declarations. The speaker of an assertive declaration may logically lie because he makes a factual claim. The example of this type of declaration is 'The ball is out.' and the example of another type of declaratives is 'I nominate you for the Pulitzer prize.'.

2.5 Agent Communication Language

The coordination between agents depends on a sophisticated system of a inter-agent communication. The language used by agents for this inter-agent communication is the **Agent Communication Language** (ACL). The main purpose of this language is to model a suitable framework that allows heterogeneous agents to interact and to communicate with meaningful statements that convey information about their environment or knowledge[38].
As Kone is stating, different Agent Communication Languages evolve around the key concept of a **communicative act** from the Speech Act Theory. Speech Act Theory claims, that a communicative act is a special type of an action, in this particular case realized by sending a message.
Over the years several different Agent Communication Languages have been implemented using different approaches (KQML, ARCOL, OAA, ICL, LOGOS, etc). Due to this, agents from different environments are not able to communicate. As we already mention earlier, in 1997 the Foundation for Intelligent Physical Agents is trying to build consensus also in this field of MAS, by introducing the standard, promising to bring into a common fold future ACLs in industry and academia [16]. Kone et al. offer descriptions of several ACLs, including generalized ACL framework. In the next few subsections we will try to briefly present and summarize the most known ACLs and their pros and cons.

2.5.1 Generalized ACL framework

According to Kone et al. [38] ACLs have following several design principles:

- **The heterogeneity principle** states that agents should be able to communicate regardless of their implementation environments. The meaning of the message exchanged should be context independent and reflect a global perspective rather than the sender or receiver of private perspectives.

- **The cooperation and coordination principle** states that effective cooperation to solve a complex task requires a meaningful communication language. An ACL should give agents the means to rely on one another, to enlist the support of other agents in order to achieve goals. The actual application of this principle depends on the design of an appropriate interaction or negotiation protocol for a given task. These protocols are high-level protocols, different from message transport mechanisms, and are intimately linked to their context.

- **The separation principle** states that a message content, structure, and transport mechanism are distinct entities that should be handled separately.

- **The interoperability principle** states that an ACL should provide heterogeneous agents with the means to interoperate.

- **The transparency principle** states that multi-agent systems should be shielded from complexities of the underlying ACL specifications. Appropriate ACL API should free agents from specific details and set interactions to a higher level. With ACL transparency, the underlying transport protocol defines message handling and multi-agent systems do not need to embed any additional functionality.

- **The extensibility and scalability principle** states that ACL designers may add new communicative acts compatible with the existing ones. These new types may implement defined interaction protocols. In addition, the design of an ACL should take into account any scalability issue related to the growing number of agents in a multi-agent system.

- **The performance principle** states that an ACL implementation should use system resources efficiently (CPU, memory, and bandwidth). The primitive communicative acts supplied with the language should be compatible with the underlying network technology and exhibit unicast, multicast as well as synchronous and asynchronous connection capabilities. In addition, an ACL must support reliable, safe, and secure message exchange between agents.

Kone et al. also propose that ACL specifications should concern with the description of a message structure, its semantic model and underlying interaction protocols. These specifications define a language and supporting protocols and encompass the following:

- **The message format** defines primitive communicative acts and message parameters (sender, receiver, message ID, protocol, and language) with expressions that describe actions at the content, message, and communication layers. Other parameters could deal with the message meaning (ontology) and delivery. In addition, the ACL supplies users with a finite set of primitive communicative acts.

- **An ACL semantics model** lays down the foundation for a concise and unambiguous meaning of agent messages and depends on the interactive behavior and capabilities of these agents. When agents interact or cooperate to achieve a goal, the mutual understanding of messages exchanged depends on the semantics given to communicative actions.

- **Interaction protocols** are sets of well-defined patterns of interactions designed to facilitate inter-agent communication. Protocols are **optional**.

- **Shared ontologies and content language** are prerequisites to knowledge sharing because the ability to exchange messages does not assume the understanding of their content. ACL designers share the same concerns in ontology development with researchers in the fields of knowledge-based systems and natural language processing (ontological engineering).

- Generally, **other services** (e.g. security) are usually provided.

2.5.2 KQML

The **Knowledge Query and Manipulation Language** (KQML) is a versatile, universal language that supports communication between several agents with a set of reserved primitives called **performatives**. KQML was build in the project sponsored by the American Government's Defense Advanced Research Projects Agency (DARPA) [52]. It is the result of the research done by the Knowledge Sharing Effort (KSE) [15], an initiative that aims at developing a foundation for software systems interaction and interoperability. This consortium was composed of three working groups: the **interlingua group**, the **shared and reusable knowledge base group**, and the **external interface group**.

The interlingua group was responsible for designing the **Knowledge Interchange Format** (KIF), mostly M. Genesereth and R. Fikes [24], a common language for describing a message content. This format is very similar to the first order logic.

The shared and reusable knowledge base group examined the problem of sharing the content of shareable knowledge base. Every knowledge-based system relies on some conceptualization of the world that is embodied in concepts, distinctions, etc. using a formal representation. This group worked on the construction of ontologies for various domains. Each ontology, written in KIF, defines a set of classes, functions, and objects for some domain of discourse, and includes an axiomatization to constrain the interpretation [7].

The **external interface group** produced the KQML language and looked at interactions of system components at the run time [38].

KQML consists of a set of performatives aiming to support interaction between agents. In KQML, agent's mental attitudes (belief, intention, commitment, choice) are expressed in the message that represents a communicative act. A KQML message is conceptually divided into three layers[39]:

- the communicative layer which specifies the sender and receiver agents, message ID;

- the message level which mainly specifies the type of performatives, language and ontology;

- the content layer, specifying message content.

```
(tell
:sender Agent-A
:receiver Agent-B
:in-reply-to message_1234
:ontology Software
:language Prolog
:content (Price Name_of_the_Software 100)
)
```

Figure 2.5: An example of KQML message

The goal of the KQML performative is to convey to a receiver that a particular proposition is true (Figure 2.5). According to Kone et al., the main advantage of KQML is its ability to support a wide range of agent architectures with extensible set of performatives. KQML first defined the agent communication language composed of distinct and independent layers. Because of this KQML became widespread in agent communication across several areas.
Although KQML was based upon the Speech Act Theory, its set of performatives does not cover all categories (e.g. commissive, permissive, expressive) and was proposed without defined semantics. Early version was criticized because of this [10] and later Labrou and Finin [40, 41] propose a significant improvement. However, the new semantics for KQML, defined with preconditions, postconditions, and completion conditions for the state of agents, provides no semantic model for mental attitudes [7, 38].
Because KQML was so popular, a number of applications in variety of fields have been proposed, ranging from the engineering of hardware and software systems to database systems and technology integration experiments. Also, M. Genesereth, in early mentioned July 1994 issue of The Communication of the ACM, proposed one of the most promising applications in the Agent-Based Software Engineering (ABSE) approach [25], where integration and interpolation between software components is achieved with facilitator agents.

2.5.3 Arcol

The **ARTIMIS** agent technology was developed by France Telecom and described in Sadek et al. [59]. It is a generic framework for instantiating communication-enabled agents. In ARTIMIS, an agent can cooperatively interact with humans as well as with other agents (Figure 2.6). Agent's communicative acts are modeled as normal 'rational actions'. By this, agents are enabled to reason about knowledge and plan actions pertaining to their communicative acts. **ARCOL** (ARtimis COmmunication Language) is the agent communication language used in ARTIMIS. An ARCOL expression relies on the language called **Semantic Language** (SL) for the definition of

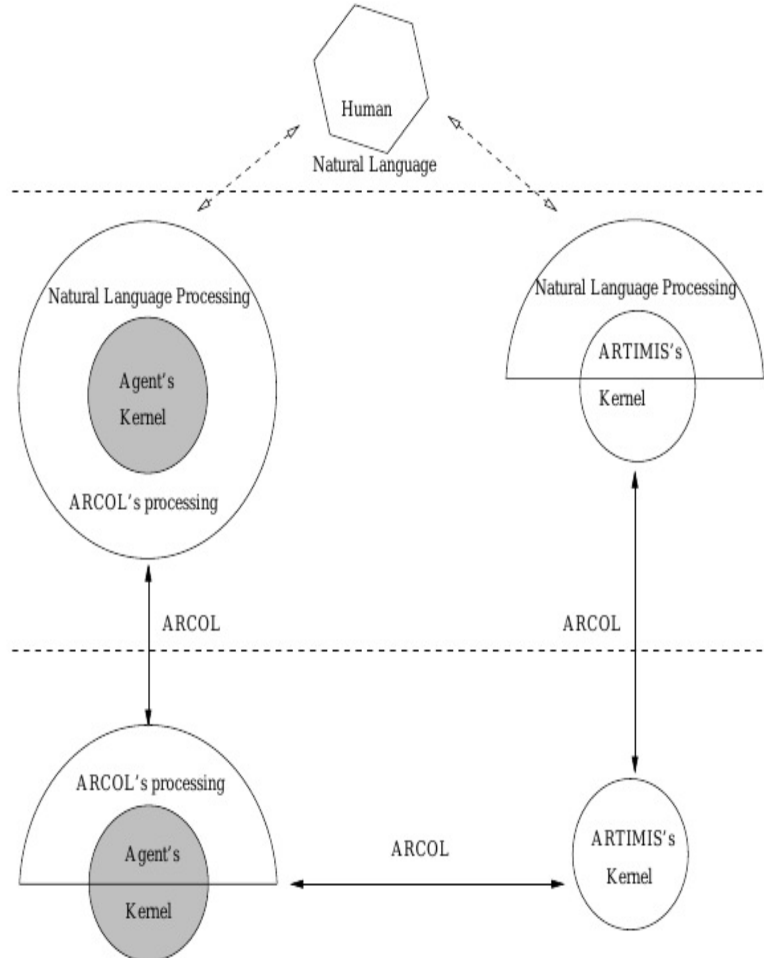


Figure 2.6: Structure of an ARTIMIS system by D. Sadek [59]

its semantics. Moreover, the Semantic Language uses the language Semantic Content Language (SCL) to describe the semantics content of a communicative act. In ARCOL we can find following set of mutually exclusive primitives:

- **Inform:** An agent uses the assertive act Inform to convey a message to another agent provided that it believes the content of this message. It needs to be stated that in ARCOL sincerity is the necessary condition for all communicative acts.
- **Request:** This directive enables an agent to demand an action from another agent provided that it has the capabilities to perform that action.
- **Confirmation:** When the sender believes that the receiver is uncertain about the property being transmitted, the communicative act becomes context-relevant.
- **Inform referent:** This communicative act enables an agent to inform another agent of the value of a referent with a given description.

The goal or intention of an agent in performing a communicative act in ARCOL is called the **Rational Effect** (RE), while prerequisites are called **Feasibility Preconditions** (FP). Feasibility Preconditions are composed of *ability preconditions* and *context relevance preconditions* [38].
The most important feature of the ARCOL, unlike KQML, is its formal semantics as a reliable support for the interoperability. On the other hand, according to Singh [64], ARCOL's fixed context with the sender agent required to be *sincere* is a constraint for the heterogeneity.

2.5.4 FIPA-ACL

```
(request-when
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content
    "((action (agent-identifier :name j)
      (inform
        :sender (agent-identifier :name j)
        :receiver (set (agent-identifier :name i))
        :content
          \"((alarm \"something alarming!\"))\"))
    (Done( alarm )))"
  ...)
```

Figure 2.7: An example of FIPA-ACL message: Agent i tells agent j to notify it as soon as an alarm occurs.

In 1996 when FIPA organization was founded, it was composed of seven sub-specifications: agents management, agents communication, agents interaction, personal travel assistance, personal assistant, audio-visual entertainment, and broadcasting network provisioning and management. In 1997 FIPA adopted the proposal by France Télécom for ARCOL ACL and its associated semantic definition and developed FIPA-ACL. This decision was made because ARCOL had defined formal semantics to underpin it, whereas KQML in those times did not. On the other hand, KQML inspired the structure of the FIPA-ACL (see Figure 2.7 for an example of a FIPA-ACL message).
Conceptually, FIPA-ACL distinguishes two levels in communication messages. At the inner level, the content of messages can be expressed in any logical language. The outer level describes locutions that agents can use in their communication. The content of messages is wrapped in these locutions. FIPA-ACL specifies 22 locutions [22], among others:

- **Accept proposal:** The action of accepting a previously submitted proposal to perform an action.
- **Agree:** The action of agreeing to perform some action, possibly in the future.

- **Confirm:** The sender informs the receiver that a given proposition is true, where the receiver is known to be uncertain about the proposition.
- **Failure:** The action of telling another agent that an action was attempted but the attempt failed.
- **Inform:** The sender informs the receiver that a given proposition is true.
- **Request:** The sender requests the receiver to perform some action. One important class of use of the request act is to request the receiver to perform another communicative act.
- **Subscribe:** The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.

The advantage of FIPA-ACL is that it lays out clearly practical components of agent communication and cooperation and a well-founded formal semantics.
However, some practical applications pointed out several limitations of the FIPA standard, for example, according to Jennings et al. [35] the FIPA standard provides no support for real-time and performance requirements of telecommunication applications. Messages should include time-related properties or temporal reasoning. In particular, interaction protocols defined in the FIPA standard do not provide fault tolerance or quality of service support that could be of interest to real-time systems.

Chapter 3

Technical foundations

Throughout this work we often use and refer to different design patterns, architectural styles, programming styles, specifications and others. In this chapter we will discuss these topics. In the first section of this chapter we describe the REST architectural style introduced by R. T. Fielding. The next section deals with FIPA specifications as our starting point in order to build a multi-agent system. In the third section we describe one of the programming paradigm - the event-driven programming and in the next section we introduce the singleton design pattern. The last section of this chapter deals with marshalling and demarshalling, an important process in transmitting information.

3.1 REST

As predicted by Perry and Wolf[53], software architecture had been a focal point for software engineering research in the 1990s. The complexity of modern software systems had necessitated a greater emphasis on componentized systems, where the **implementation** is partitioned into **independent components** that communicate to perform a desired task. Software architecture research investigates methods for determining how to best partition a system, how components identify and communicate with each other, how the information is communicated, how elements of a system can evolve independently, and how all of the above can be described using formal and informal notations[13].

In year 2000, R. T. Fielding in his dissertation introduced the **Representational State Transfer** (REST) architectural style, derived from several other network-based architectural styles. He described how to use REST to guide the design and development of the architecture of the modern Web[13].

In his work he claims, that REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.

The REST architectural style describes six constraints, which need to be conformed in order to application be REST-ful:

- **Client-Server constraint,** is based on the principle of separate concerns. By separating the user interface concerns from the data storage concern, Fielding is trying to improve **portability** of the user interface across multiple platforms (because clients are not concerned with the data storage) and improve **scalability** by simplifying server components (because servers are not concerned with the user interface). This allows to replace and/or develop clients and servers as long as the **interface** between them **is not changed**.

- **Stateless constraint,** is dealing with a client-server interaction. Fielding is convinced, that **communication** between a client and a server **must be stateless** in nature. **Each request** from a client to a server **must contain all** of the information **necessary to understand the request** and can not take the advantage of any stored context on the server. Session state is kept entirely on the client side. This improves visibility of the server, as the monitoring system does not have to look beyond a single request to determine the nature of the request. Reliability is also improved, as the task to recover from a partial failure becomes easier. Also scalability of the server is improved because not having to store the data between requests allows the server component to quickly free resources and simplifies implementation of the server. The drawback is, that it may decrease network performance by increasing the repetitive sent in a series of request, since the data cannot be stored on the server.

- **Caching constraint** requires that the data within a response for a request be implicitly or explicitly labeled as cacheable or non-cacheable. Caching can improve network efficiency in some situations by reducing the average latency of a series of interactions. The trade-off, however, is that the cache can decrease reliability if stale data within the cache differs significantly from the data that would have been obtained, if the request had been sent directly to the server.

- **Uniform interface** between components is the central feature that distinguishes between the REST and other network based architectural styles. Fielding claims, that by applying the software engineering principle of the generality of the component interface, the overall system architecture is simplified and the visibility of interactions is improved. The trade-off, though, is that a uniform interface degrades efficiency, since information is transferred in a standardized form instead of the one which is specific to an application's needs.

- **Layered system** style allows an architecture to be composed of hierarchical layers by constraining component behavior (e.g. component cannot see beyond

immediate layer). Layers can be used to **encapsulate** legacy services and to **protect new services from legacy clients**, simplifying components by **moving** infrequently used **functionality to a shared intermediary**. The primary disadvantage of layered systems is that they add overhead and latency to the processing of the data, reducing user-perceived performance.

- **Code-On-Demand** - REST allows client functionality **to be extended by downloading and executing code** in the form of applets or scripts. This simplifies clients by reducing the number of features required to be implemented. Allowing features to be downloaded after deployment improves system extensibility. However, it also reduces visibility, and thus is only an **optional constraint** within REST.

3.2 FIPA

FIPA was established in 1996 as an international non-profit association of companies that agreed to share efforts to produce standard specifications of generic agent technologies. Since then FIPA has generated a set of specifications that went through three cycles of review: FIPA97, FIPA98 and FIPA2000. The last standardized review is called FIPA2002. The aim of these specifications is to make agent technologies usable across a large number of applications so that a high level of interoperability across applications is achieved. These specifications are focusing more on explicitly specifying how agents communicate and connect and less on specifying components such as agents, humans, data and services that transform, generate, process and store messages that are communicated[56]. The overall abstract architecture is defined in SC00001 [17], other specifications can be regarded as concrete parts that realize this abstract architecture (see Figure 3.1). We will briefly describe few of the specifications.

3.2.1 Agent Management Specification

The specification SC00023 [1] provides the normative framework within FIPA agents exist and operate. It establishes the logical reference model (Figure 3.2) for the creation, registration, location, communication, migration and retirement of agents. The agent management reference model consists of the following logical capability sets (services) [19]:

- An **Agent** is a computational process that implements the autonomous, communicating functionality of an application. Agents communicate using an Agent

[1] `http://www.fipa.org/specs/fipa00023/index.html`

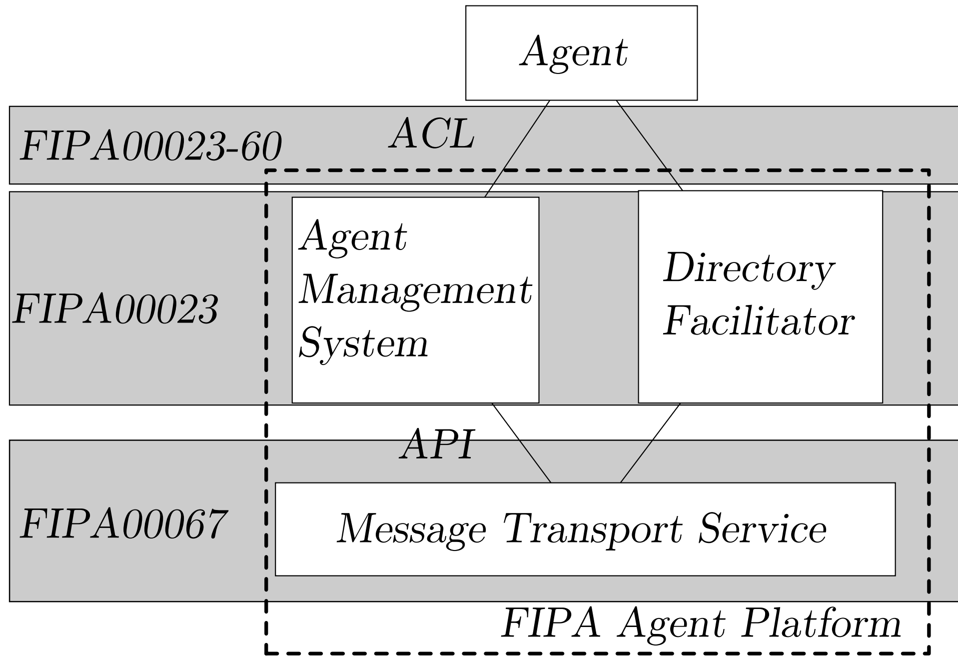


Figure 3.1: Parts of FIPA architecture specified in different specifications

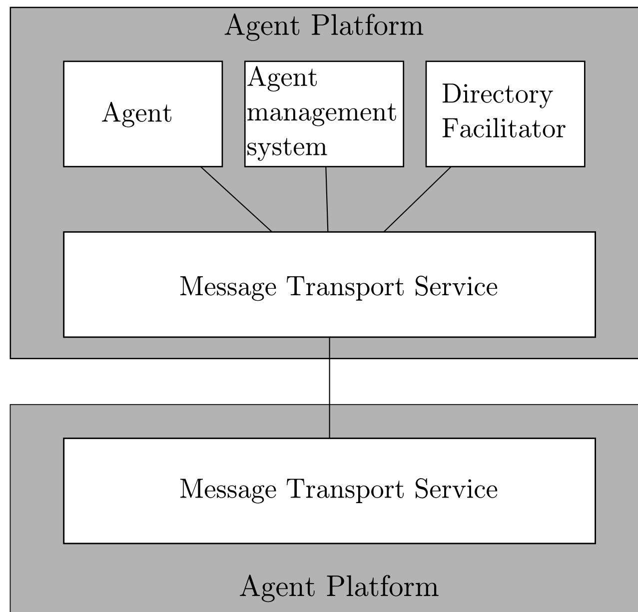


Figure 3.2: Agent Management Reference Model

Communication Language (ACL). An agent is the fundamental actor on an Agent Platform(AP) which combines one or more service capabilities, as published in a service description, into a unified and integrated execution model. An agent must have at least one owner, for example, based on organizational affiliation or human user ownership, and an agent must support at least one notion of identity. This notion of identity is the Agent Identifier (AID) that labels an agent

so that it may be distinguished unambiguously within the Agent Universe. An agent may be registered at a number of transport addresses at which it can be contacted.

- A **Directory Facilitator (DF)** is an optional component of the AP. The DF provides yellow pages services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Multiple DFs may exist within an AP and may be federated.

- An **Agent Management System (AMS)** is a mandatory component of the AP. The AMS has supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of AIDs which contain transport addresses (among other things) for agents registered with the AP. The AMS offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID.

- An **Message Transport Service (MTS)** is the default communication method between agents on different APs, it is specified in the specification SC00067[2].

- An **Agent Platform (AP)** provides the physical infrastructure in which agents can be deployed. According to the specification, the AP consists of the machine(s), operating system, agent support software, FIPA agent management components (DF, AMS and MTS) and agents.

FIPA leaves internal design of an AP for agent system developers, thus it is not a subject of standardization. FIPA is concerned only with how communication is carried out **between agents** who are **native to the AP** and agents outside the AP. **Agents are free to exchange messages directly by any means that they can support.**

3.2.2 Message transport service

FIPA SC00067 specification deals with message transportation between inter-operating agents. The reference model (see Figure 3.3) for agent message transport comprises of three levels [21]:

- The **Message Transport Protocol (MTP)** is used to carry out the physical transfer of messages between two Agent Communication Channels.

- The **Message Transport Service (MTS)** is a service provided by the AP to which an agent is attached. The MTS supports the transportation of FIPA ACL messages between agents on any given AP and between agents on different APs.

[2] `http://www.fipa.org/specs/fipa00067/index.html`

- The ACL represents the payload of the messages carried by both the MTS and MTP.

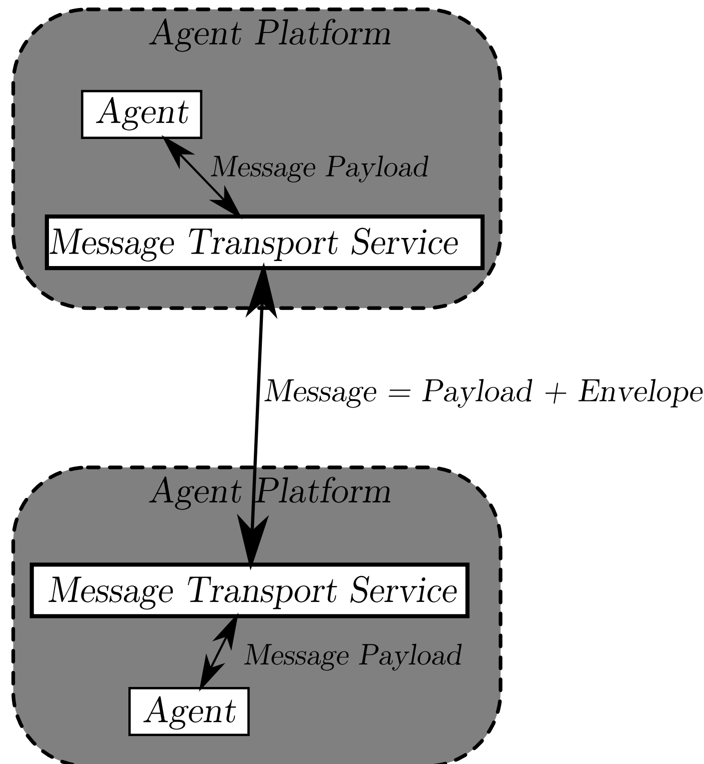


Figure 3.3: Message Transport Reference Model

The message that is being exchanged between the ACCs is, in its abstract form, made up two parts: **a message envelope** expressing transport information and a **message payload** comprising the ACL message of the agent communication. We need to note, that all information in the message **envelope** is **supporting information only**.

3.2.3 Agent Communication Language Message

FIPA specification SC00061 [3] is standardizing the form of FIPA-ACL messages. Objectives of standardizing this form are according to the specification [18]:

- To ensure interoperability by providing a standard set of ACL message structure, and,
- To provide a well-defined process for maintaining this set.

The FIPA ACL message contains the set of one or more message parameters (see Figure 3.4 for the full list of message parameters), from which only the **performative** is mandatory. It is expected that most ACL messages will also contain other parameters

[3] `http://www.fipa.org/specs/fipa00061/index.html`

such as **receiver, sender and content**. The very important feature of the FIPA is that specific implementations are **free to include user-defined** message parameters other than FIPA ACL message parameters. The semantics of these is not defined by FIPA, and is left to the implementer.

Parameter	*Category of Parameters*
performative	*Type of communicative acts*
sender	*Participant in communication*
receiver	*Participant in communication*
reply-to	*Participant in communication*
content	*Content of message*
language	*Description of Content*
encoding	*Description of Content*
ontology	*Description of Content*
protocol	*Control of Conversation*
conversation-id	*Control of Conversation*
reply-with	*Control of Conversation*
in-reply-to	*Control of Conversation*
reply-by	*Control of Conversation*

Figure 3.4: FIPA ACL Message Parameters

3.2.4 Content language

The SC00008[4] specification defines a concrete syntax for the FIPA Semantic Language (SL) content language. This syntax and its associated semantics are suggested as a candidate content language for use in conjunction with the FIPA Agent Communication Language [23], i.e. it **is not mandatory to use** FIPA SL as a content language in FIPA ACL messages, implementer can decide to use any content language.

FIPA SL is very expressive language, therefore for some cases of agent communication tasks it is **unnecessarily powerful**. To deal with this problem and allow simpler agents perform simple tasks with minimal computational load, FIPA introduces **semantic and syntactic subsets** of the full FIPA SL content language. These *profiles* are defined in increasing order of expressivity [23]:

- **FIPA SL0: Minimal subset**
  Profile 0 of FIPA SL is the minimal subset of the FIPA SL content language. It allows a representation of actions, a determination of results, a term representing a computation, a completion of an action and simple binary propositions.

[4] `http://www.fipa.org/specs/fipa00008/index.html`

- **FIPA SL1: Propositional Form**
  Profile 1 of FIPA SL extends the minimal representational form of FIPA SL0 by adding Boolean connectives to represent propositional expressions.

- **FIPA SL2: Decidability Restrictions**
  Profile 2 of FIPA SL allows first order predicate and modal logic, but is restricted to ensure that it must be decidable. Well-known effective algorithms exist that can derive whether or not an FIPA SL2 Wff is a logical consequence of a set of Wffs (for instance KSAT and Monadic).

3.3 Event-driven programming

In the past, when CPU time was very expensive and there were many computers, programmers were designing applications carefully, so that the CPU was constantly busy with processing data, often organized into batch jobs. Later, when computers became more available to the wider audience and CPU time became inexpensive, new application design becomes very popular.

Event-driven programming or **event-based programming** is a programming paradigm in which the flow of the application is determined by events, for example by messages or mouse clicks.

This means, the application is running in an usually endless loop waiting for the occurrence of some event. After recognizing the event, application reacts by performing appropriate action (e.g. generating other type of event, calling object method, etc.) and once, when event handling is complete, application goes back to waiting state (see Figure 3.5).

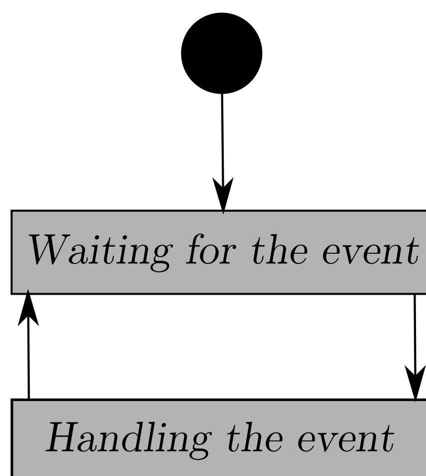


Figure 3.5: Endless event-handling loop

This scheme implies, that upon arrival of an event the CPU executes only tiny fraction

of the overall code, instead of running the entire code. The **main challenge** is **to quickly pick and execute the right code fragment**. **Main drawback** of this paradigm is that the application is especially vulnerable to **events** that **require different handling depending on the context [60].**

3.4 Singleton design pattern

Sometimes in programming we might occur situations, where one and only one instance of an object in the entire application is needed. In such case the singleton design pattern is used.
The use of this design pattern guarantees that only one instance of the class will be used in the application and a global point of access will be granted. However, the use of this design pattern brings few drawbacks with it, e.g.:

1. **that the use of this pattern will make unit testing much more difficult, as it introduces global state into the application [31], and**
2. **deleting of a singleton becomes a non-trivial problem, because ownership of the singleton instance cannot be reasonably assigned [68].**

Also initialization of the singleton instance is lazy, which means it is instantiated only when it is needed in the application (see Figure 3.6 for thread safe implementation in JAVA).

```
public class Singleton {
        // Private constructor prevents instantiation from other classes
        private Singleton() { }

        /**
        * SingletonHolder is loaded on the first execution of Singleton.getInstance()
        * or the first access to SingletonHolder.INSTANCE, not before.
        */
        private static class SingletonHolder {
                public static final Singleton instance = new Singleton();
        }

        public static Singleton getInstance() {
                return SingletonHolder.instance;
        }
}
```

Figure 3.6: Thread-safe solution of a singleton class without requiring special language constructs in JAVA by Bill Pugh [57]

3.5 Marshalling/Demarshalling

Marshalling is the process of transforming the memory representation of an object to a data format that is suitable for transmission. Demarshalling is the reverse process, i.e. transforming the data into the memory representation of an object. In multi-agent systems, this is used to send instances of message objects between agents. There are several methods used in multi-agent middlewares to marshal objects:

- Serialization in JAVA, used for example in JADE, Cougaar [29], FraMaS [1] (all of them using JAVA RMI). It is intuitively clear that an object serialized in one language can not be deserialized in another language.

- Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables separate pieces of a software written in different languages and running on different computers to work with each other like a single application or set of services. CORBA uses an interface definition language (IDL) to specify interfaces which objects present to the outer world, then specifies a mapping from IDL to a specific implementation language (CBMAS [9]).

- Simple Object Access Protocol (SOAP) is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks and it relies on Extensible Markup Language (XML) for its message format. Used, for example, in Jade [47]. You can find an example of a SOAP message in Figure 3.7.

- Marshalling based on representational languages such are KQML (RETSINA MAS [65]) or FIPA-ACL (JADE).

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 3.7: An example of SOAP message

Chapter 4

Problem description

One can think having so many different multi-agent platforms will make it very easy to find an appropriate platform for the purpose of a cognitive science research. But when we were looking for a platform matching our vision, we could not find any. We admit that our requirements might be too strict at first view, but the connection between the humans and their personal devices is getting more and more intense. The academic field needs to be prepared for such a scenario in order to adapt research. We identified several problems in current platforms, which encouraged us to design and implement our own solution. These problems are:

- **platform dependency**,
- **centralization**,
- problems of **direct communication** in a multi-agent platform,
- lack of the **ACL**, and
- system requirements.

Our aim is to design and implement a **platform independent** communication platform, which will be **decentralized**. Agents will be communicating **directly** using **ACL**. This system is aimed to run on a variety of devices, including **embedded devices**. In the following sections we will briefly describe problems that we encounter.

4.1 Platform dependency

As we already stated before, there is a need for platform independent multi-agent middleware in cognitive science. It is likely that the platform may not be running on regular computers at all, rather on small devices that people keep close to them during the day, such as mobile devices, mp3 players, etc. Furthermore, it is certain

that devices will become even smaller and they will probably become an extension to our body and senses (e.g. Google's Project Glass[1], cochlear implant[2], different brain implants, etc.) We think that agents in such devices will quickly become a part of the scope of the research within cognitive science.

A careful reader has already noticed that these agents will run in a **heterogeneous environment**. To be able to run on the vast variety of platforms and still be able to maintain communication between agents, a multi-agent system needs to be **platform independent**, preferably implemented in multiple languages. This will give a designer the possibility to choose a device based on the needs of the experiment and not because of the limits of the multi-agent platform. This, at the end, may be the difference between a successful and unsuccessful experiment. However, to our knowledge, the majority of MAS middlewares are implemented in one language only. There are several attempts to implement a multi-agent platform in more languages, but to this date these implementations are not complete. For example **Smart Python multi-Agent Development Environment (SPADE)** [28] was intended to be implemented in several languages but the process of implementation has not started yet, thus only Python implementation is available.
There are several reasons why the middleware is not implemented in more languages, therefore platform dependent, from which the most notable are marshalling/demarshalling and a discovery service.

Marshalling/Demarshalling

We have already described marshalling and demarshalling in the section 3.5, where we mentioned the platform dependent and platform independent approaches. Platform dependent approaches are usually easier to implement and the platform provides more services to programmers, while platform independent approaches need significantly more work and time to be implemented. Implementer can not rely on any specific feature of the platform, usually he needs to implement such services by himself.

Discovery service

Agents need to find other agents on the network in order to communicate with them. For this purpose agents use a **discovery service** which is usually provided by the platform or one of the agents. Sometimes, the middleware is using naming services that are platform dependent, e.g. JADE uses JAVA RMI Registry.

Aim: Our solution should be **platform independent**, not relying on any specific service tied to the specific platform. It should use technologies that are not platform

[1] `https://plus.google.com/111626127367496192147/posts`
[2] `http://en.wikipedia.org/wiki/Cochlear_implant`

dependent and are available for variety of systems.

4.2 Centralization

Humans in the real world have the ability to communicate with other agents directly, without any other intermediary. Also, for finding other agents in the environment, people are using their five senses. They do not need to write the sign 'I am here!' just to let other people around them know that they are there. We can say that the real world is a **decentralized** system. Roughly speaking, from the viewpoint of a decentralized system nothing significant is going to happen if one of the agents will be removed. It is probable that other agents will find their own way even without it. They still will be able to communicate with other agents and find other agents. Decentralized systems are more fault tolerant, as they do not rely on single source of a service (server).
On the other hand, it is easier to maintain a **centralized** system, as all the necessary information is held at a single point. The drawback for providing services for every agent in the system is that the server needs to be a quite powerful device.

Aim: Agents in an agent platform should not rely on any central point - platform should be as **decentralized** as possible.

4.3 Direct communication

Nowadays, computers connected to the Internet are using logical addresses to locate other computers. We can compare this logical address to a phone number. Every mobile phone needs one to be able to receive a call. You simply dial the number and wait for the connection, which will be established in less than a second. This *magic* actually is, roughly speaking, the process of finding the device by its logical address. But what will happen if there is not enough phone numbers for all the mobile phones? Even this magical process will not be able to find the phone without a number, it is just impossible. This is exactly what is going on the Internet with IPv4 addresses: there is simply too many devices and too little address space. However, a solution was proposed to connect every device to the internet.
NAT (Network Address Translation), a process of modifying IP addresses information in IP packet headers, is used in some routers. Devices are connected to such a router and given local logical addresses, thus these devices are able to reach the Internet (see Figure 4.1). However, such solution brings some drawbacks. The non-trivial task is to establish a connection to a device behind the NAT, because only the logical address of a router is known.

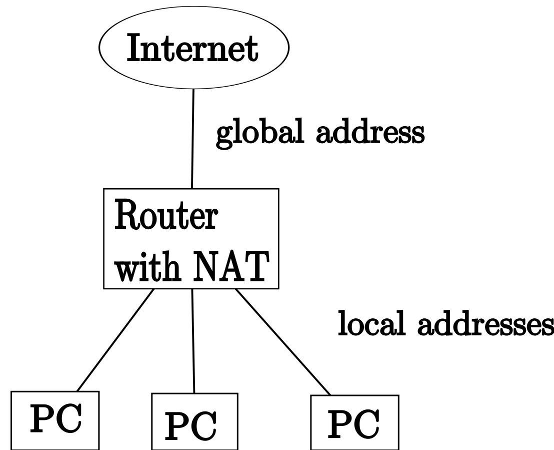


Figure 4.1: Part of the Internet architecture with NAT enabled router.

Aim: Agents behind the NAT should be able to send and **receive** messages.

4.4 ACL in MAS

As we stated earlier, most multi-agent systems are using agent communication languages to express agent's intentions, wants and needs in communication with other agent. However, not all multi-agent systems are using languages inspired by the Speech Act Theory, for example ARGUGRID [3] [8] uses Multiattribute ARGumentation framework for Opinion explanation (MARGO) [49]. This might be a major deal breaker in the process of picking the right middleware for a cognitive science research.

Aim: Agents should use an **agent communication language** inspired by the Speech Act Theory.

4.5 System requirements

Small portable devices are neither as powerful as servers nor desktop computers. Therefore, it is not easy to run middleware developed for such devices on significantly less powerful portable devices, such as tablets, smart phones or even mp3 players. There are several attempts to port middlewares for these smaller devices, e.g. LEAP[42] or JADE for Android[66]. The first runs on devices with Java 2 Micro Edition (J2ME) or Personal Java, the latter should run on Android devices, but the truth is, that so far it is running only in Android SDK emulator with at least 1024 MB RAM. We consider

[3] www.argugrid.eu

this unacceptable, because current smart phones usually have 500MHz ARM CPUs with 256MB RAM. But running on smart phones might not be enough as other mobile devices, that might be interesting for the research, can be much less powerful.

Aim: Multi-agent middleware should have **low system requirements** to be able to run on small and embedded devices, e.g. smart phones.

Chapter 5

Design and Analysis of the Solution

When designing our multi-agent middleware, we need to keep several things in mind: many different constraints we need to satisfy, goals to achieve and mostly, the reason why we are developing such a system. All of this brings challenges, either the design of the system, or the implementation.
In order to implement middleware that is able to run on multiple platforms we first need to design an abstract architecture of the platform. Every implementation of our platform should be implemented according to this architecture to ensure the compatibility across implementations.

In the first section of this chapter we propose this abstract architecture of our system and describe which parts of the system are inspired by FIPA specifications. In next sections we provide description of other components of our system, starting with the agent description in the second section. The third section deals with the Message Transport Service, component that is responsible for maintaining communication between agents. In Section 5.4 we describe a component that is responsible for finding other agents in the local network, the Discovery Service. In the last section we describe how the Platform connects all other components together.

5.1 Abstract Architecture

An abstract architecture is the guide throughout the process of implementation and it is one of the most important part of this work. Therefore we propose an abstract architecture of our platform on which individual implementations of our middleware should be based.
Our abstract agent platform is greatly influenced by the FIPA abstract architecture and it consists of:

- An **Agent** is a mandatory component of the platform. The number of agents in the platform is not limited.

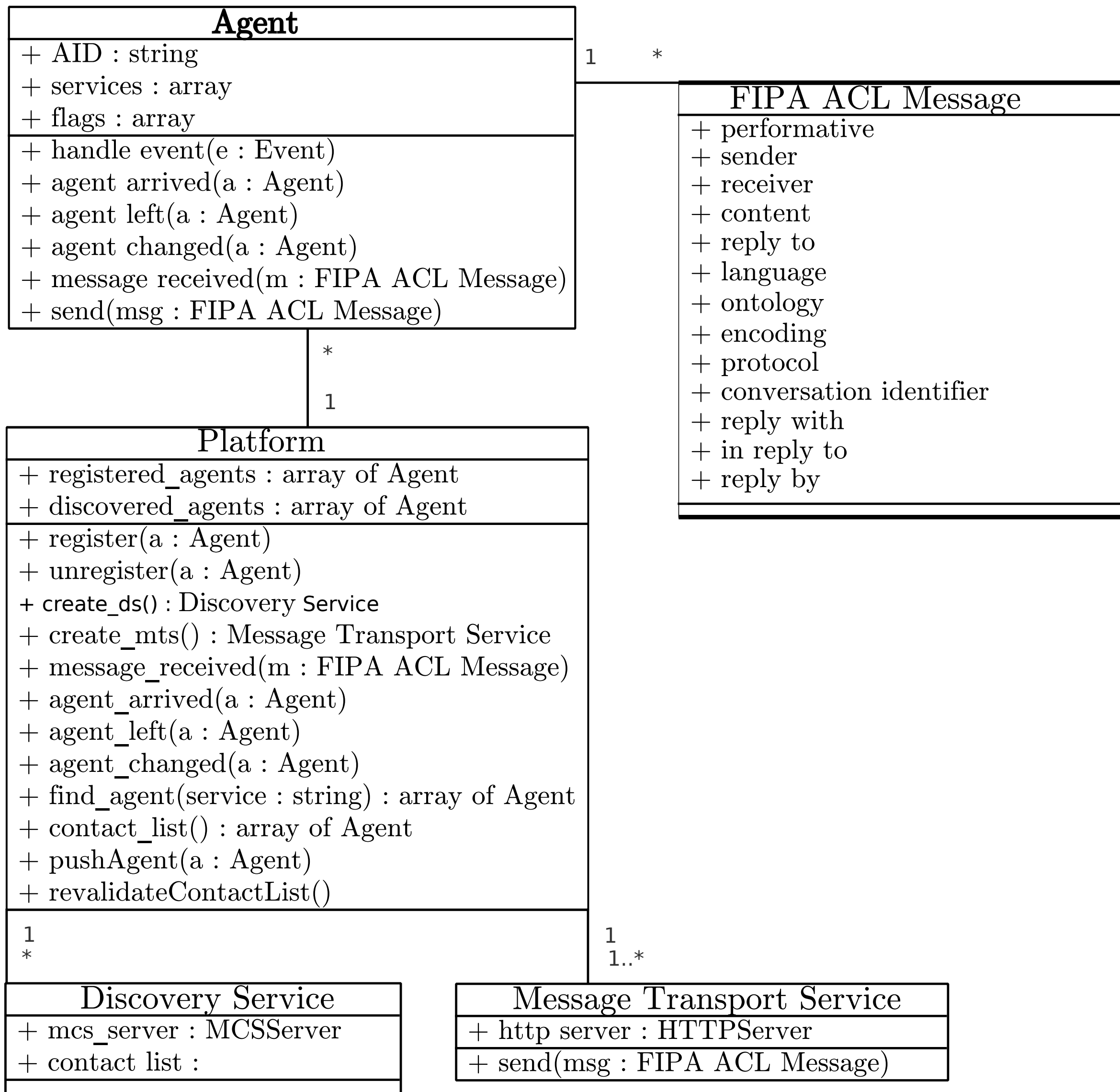


Figure 5.1: A class diagram of an abstract architecture. At least one reactive agent with several abstract methods is present in the platform. Agent is able to register and unregister within the platform. Every agent is allowed to create FIPA ACL messages and send them through the platform.

- A **Message Transport Service** is a mandatory component of the platform. It is responsible for sending and receiving FIPA ACL messages, and message polling.
- A **Discovery Service** is an optional component of the platform. It serves as white and yellow pages for agents in the platform.
- A **Platform** itself. The platform is joining all other components together. Agents are able to register and unregister within this platform. The platform is the owner of DS and MTS components providing an interface for their services to registered agents. As agents are not able to reach these services directly, but

through platform's interfaces, the platform can own several different DS and MTS components.

Figure 5.1 shows an UML class diagram of our abstract architecture. Platform is the owner of every Agent, DS and MTS in the system. Agents need to register using platform's register()/unregister() methods in order to be served by the platform. The platform is not responsible for creating and destructing agents. Agents are using platform's send() method for sending FIPA ACL Messages. On the other hand, the platform is responsible for the creation of DS and MTS components (create_ mts() and create_ ds() methods from Figure 5.1). It also provides the interface for maintaining the contact list of remote agents for DS components (pushAgent(),removeAgent(),changeAgent() and revalidateContactList()). MTS components are passing received messages to the platform via messageReceived() method. Platform invokes appropriate event in appropriate agent when it receives a message, or some changes in the contact list of remote agents have been made.

Agents are able to create FIPA ACL Messages and send them by calling appropriate platform's method. Agent is waiting for and handling events created by the platform, the handler then calls corresponding function, i.e. agentArrived(), agentLeft(), agentChanged() or messageReceived().

5.1.1 FIPA inspired

Inspired by FIPA's intention to produce MASs that are able to interoperate, we took a closer look at these specifications. However, we soon realized that implementing all of the features would be complicated and make the platform resource demanding, like JADE and other middlewares. This is exactly what we want to avoid. Because of this, we are not going to implement every FIPA specification, but only following:

- **FIPA-ACL** defined in FIPA SC00037 [1] will be used as an agent communication language in our platform. We want our platform be used in cognitive science and we think the ACL in such a platform should be inspired by the Theory of Speech Act.
- To be able to communicate with other FIPA compliant platforms, we think our best possibility is to implement **FIPA Message Transport Service (SC00067)**.
- In the implementation of our platform we are going to use HTTP protocol based on FIPA SC00084 [2] because HTTP protocol is widely used and is RESTful (see Section 3.1).

[1] http://www.fipa.org/specs/fipa00037/index.html
[2] http://www.fipa.org/specs/fipa00084/index.html

- We decided to use the **Extensible Markup Language (XML)** for the **Message Content Representation**, according to the FIPA SC00071 [3],
- and for the representation of the envelope, the **Message Transport Envelope Representation described** in FIPA SC00085[4].

So far, we are **not** going to implement **interaction protocols and FIPA SL content language** as we do not want to limit designers in any way. We want designers to be able to experiment with custom protocols and content languages. Implementation of **FIPA interaction protocols** as an optional feature is planned in future versions.

5.1.2 Heterogeneity

As we already stated before, one of the reasons we are designing and implementing our own platform is the lack of platforms connecting agents from different environments. If one wants to run agents on different devices and still be able to maintain communication between them, platform dependent services can not be used in such case. Because of this we adapted the design of our platform (see Section 5.1) and all of the services provided by our platform, for example discovery service (Section 5.4) or HTTP based communication (section 5.3).
In addition, our platform is implemented in **three different programming languages**, namely in C++, Java and Python, to be able to run on almost entire variety of reasonably powerful devices.

5.2 Agent

Agents in our platform are designed according to the description given in Section 2.1. The **default agent** is a Simple reflex agent driven by events (see Section 3.3), although deliberative behavior is possible and is left for the implementer. Agent is able to register and unregister with the platform and is able to receive events when registered. Events for the agent are generated by the platform based on inputs from Message Transport Service (see Section 5.3) and Discovery Service (see Section 5.4) components. Agents must have unique agent identifier across the platform, but this restriction is also left for the implementer to manage. Agents have the ability to manage their own services and flags. Flags and services are part of the agent identifier section in FIPA ACL message. For this purpose we extended the FIPA ACL message representation in XML. This extended specification is still FIPA compliant and can be found in Appendix A. Agents are able to create FIPA ACL messages and send them by calling the appropriate platform method.

[3] `http://www.fipa.org/specs/fipa00071/index.html`
[4] `http://www.fipa.org/specs/fipa00085/index.html`

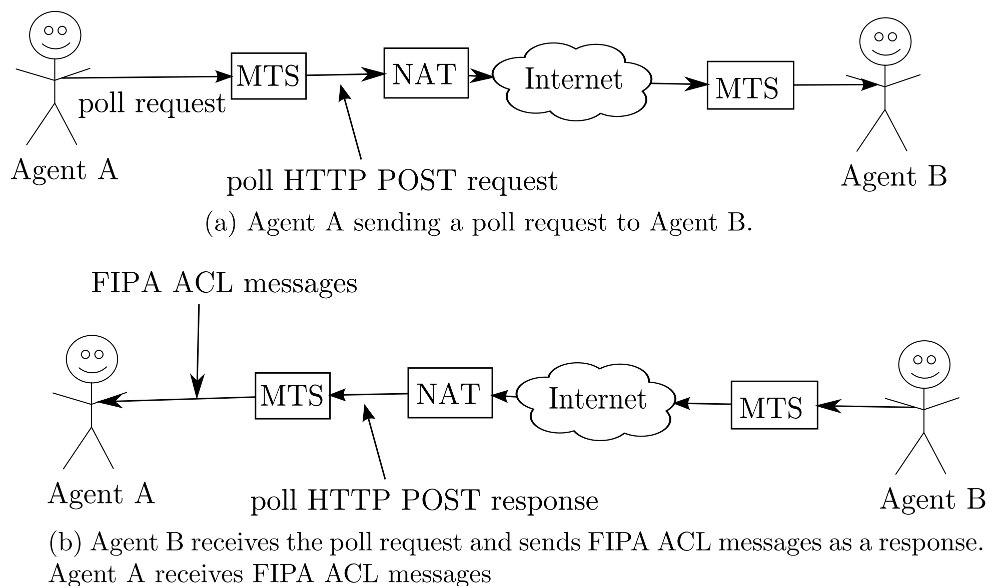
5.3 Message Transport Service

As we already mentioned in Section 5.1.1, our Message Transport Service (MTS) is greatly inspired by FIPA specifications as we want it to communicate with other FIPA compliant platforms.
Our MTS is created and owned by the platform. The platform can decide on which port the MTS listens. MTS sends and receives FIPA ACL messages. It uses HTTP protocol from FIPA Agent Message Transport Protocol for HTTP Specification [20], thus our MTS is able to send and receive messages from other FIPA compliant MTSs. MTS calls appropriate method of the platform when a message is received and offers the interface for platform to send messages.

5.3.1 Message polling

Our MTSs communicate directly and we have already described the problem of the direct communication in the section 4.3. One of the possible solutions is to use communication through the environment, but this brings the need for a quite powerful server, which we find very restrictive.

In contrast to this approach, we decided to take another way and implement new

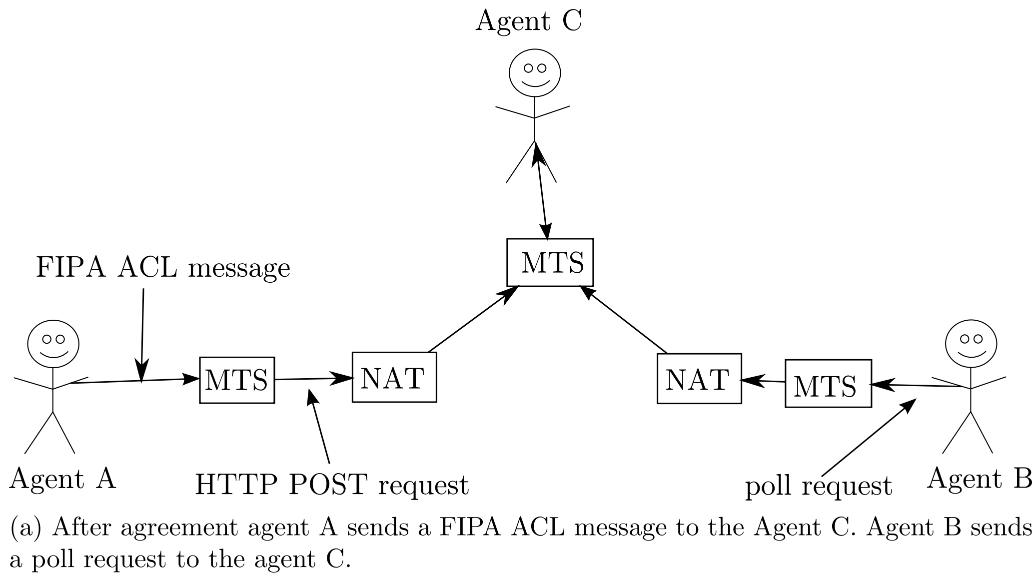


(a) Agent A sending a poll request to Agent B.

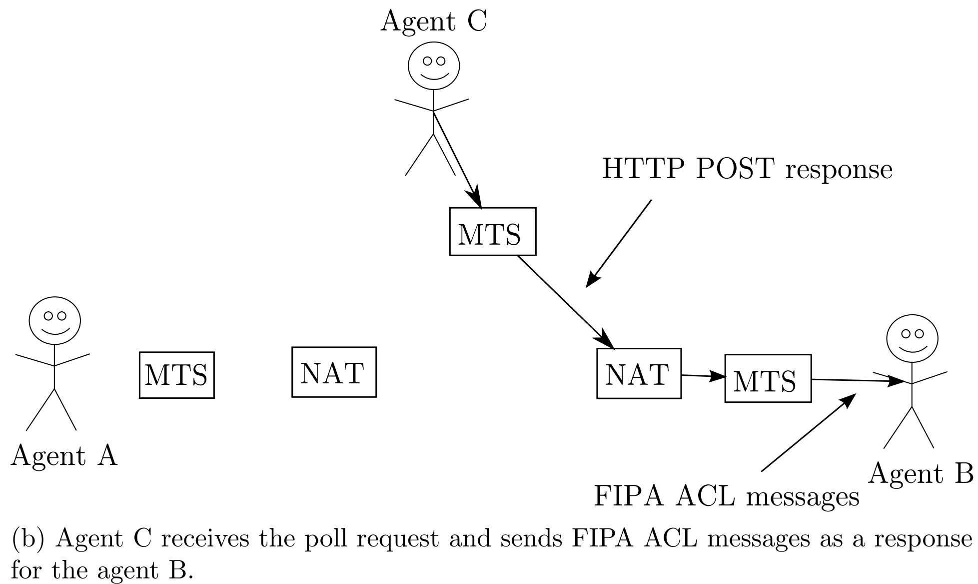
(b) Agent B receives the poll request and sends FIPA ACL messages as a response. Agent A receives FIPA ACL messages

Figure 5.2: Message polling

service that will be available in our platform. Agents that are behind the NAT are not able to receive any messages, but they are able to send them. We decided to use this feature for **message polling** - agents are not sending messages to agents behind the NAT, rather they store these in an outbox and wait until somebody picks them up. An agent behind the NAT sends a special HTTP POST request and as a response he



(a) After agreement agent A sends a FIPA ACL message to the Agent C. Agent B sends a poll request to the agent C.



(b) Agent C receives the poll request and sends FIPA ACL messages as a response for the agent B.

Figure 5.3: Message polling using other agent

receives desired messages (see Figure 5.2). We are aware that this brings the problem of centralization, but we polling will be used only in necessary cases. Nonetheless, every agent is capable of message polling which might help to distribute the load, thus increase the decentralization of the platform. Theoretically speaking, using the Speech Act Theory in agent communication agents are also able to convince other agents to pick or/and poll messages for them. See Figure 5.3 for diagram of message polling in such a case.

For the purpose of the message polling, we propose following **Message Polling extension to the FIPA Agent Message Transport Protocol for HTTP Specification**. A HTTP request for message polling comprises:

- Request Line

- The request method type that must be POST.
- The request resource identification must be a full URI where the path ends with **/poll/**.
- The request version that must be HTTP/1.1.

- Request Headers
  - The mandatory parameter Content-Type: that must be text/plain
  - The mandatory parameter Host: that must be in the form hostname or hostname:portnumber.
  - The mandatory parameter Cache-Control: that must have the value no-cache.
  - The mandatory parameter MIME-Version: that must have the value 1.0.
  - The optional parameter Content-Length: that contains the size of the request body.

- Request Body
  Simple string in format [**(aid,value1),...,(aid,valueN)**] where **aid is reserved keyword and value1 to valueN are IDs of agents for which messages should be polled.**

A HTTP **response** comprises:

- **Response Line**
  The response version must be HTTP/1.1. The response status code must either be the success code or a suitable error code

- **Response Headers**
  - The mandatory parameter Content-Type: that must be multipart/mixed and must have a boundary parameter enclosed by double quotes.
  - The mandatory parameter Cache-Control: must have the value no-cache.
  - The optional parameter Content-Length: specifies the size of the response body

- **Response body**
  The response body contains a multipart message comprised of FIPA ACL messages formatted according to [20].

5.4 Discovery Service

Discovery Service is responsible for finding other agents over the local network. It is inspired by the FIPA **Directory Facilitator (DF)** and the **Agent Management System (AMS)** specified in FIPA SC00023[5]. The Discovery Service component is responsible for both white and yellow pages that the platform is providing, thus maintaining agent names and addresses with their services and flags. There can be more than one instance of the Discovery Service object in the platform. The DS calls appropriate platform's method when an agent has arrived or left the network, or the agent has changed its flags or services.

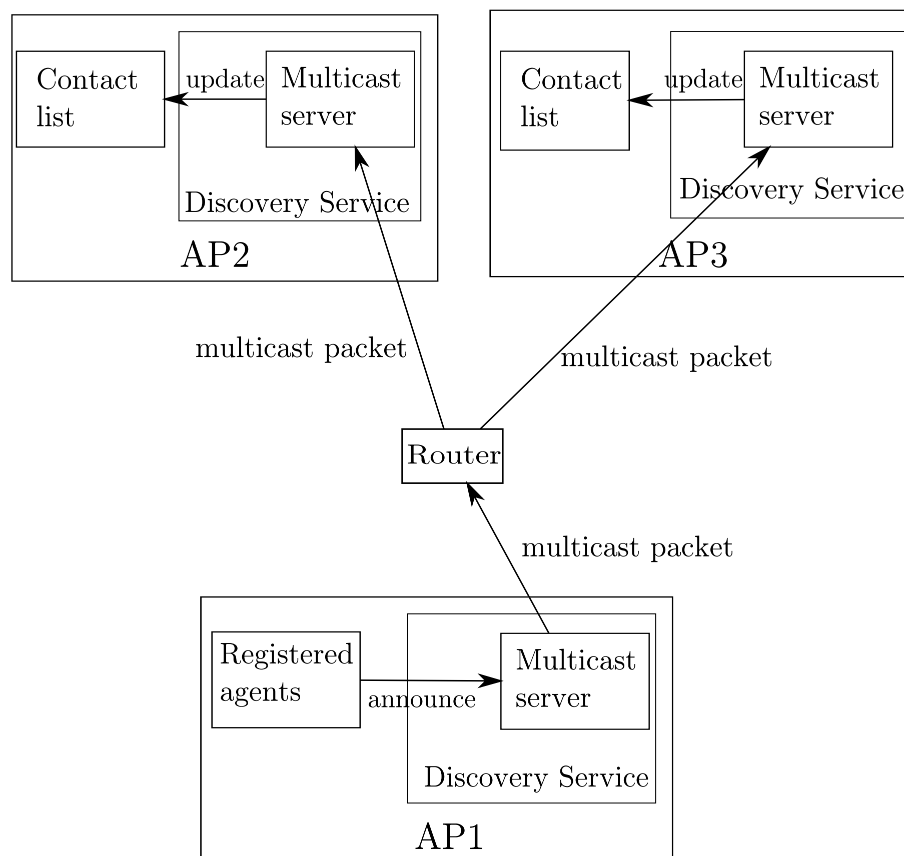


Figure 5.4: Discovery Service from AP1 sends multicast packet to DS in AP2 and AP3 according to what they update their contact list.

The DS is periodically checking the list of agents registered to the platform and sending information about them to the multicast group (see Figure 5.4). The default multicast group is at the address 239.25.25.25 on port 12345 and the default interval is one second. The DS is sending data packets in the following format:

action;aid;address;port;services;flags;

[5] `http://www.fipa.org/specs/fipa00023/index.html`

where action is one of the reserved words 'announce' and 'leave'. 'Announce' is used for announcing agent that is registered with the platform, 'leave' action is used when the agent has unregistered from the platform. Aid is an agent unique identifier. The address is the string representation of the address of the agent, if it is not specified, address is taken from the received packet. Services and flags are comma separated lists of strings representing agent's services and flags.

DS is receiving data from the multicast group. Depending on the action of the message, DS adds, removes or changes agents in the contact list of remote agents, thus maintaining this list. It also periodically checks and removes agents that were not announced for a longer period from the list. Default interval for purging the list is five seconds.

5.5 Platform

As we stated before, the platform is joining all other components together. Platform allows agents to register/unregister, thus allowing agents to use platform's services, e.g. sending messages, message receiving, white and yellow pages. Platform is the owner of all DSs and MTSs within the system. It is able to create, own and use MTS and DS components. It provides interface for agents to send and poll messages and invokes appropriate events in registered agents. The platform provides interface for both MTS and DS and handles inputs from these components. Platform keeps the list of registered agents and the list of agents found by the DSs.

Chapter 6

Implementation

To stay consistent with our statement, we decided to implement our platform in three, nowadays very popular languages. C++ and Java belongs to the top three most popular languages. Python is a powerful dynamic programming language with very clear and readable syntax and very natural expression of procedural code. Also, Python has very extensive standard libraries and third party modules. Because of this, Python is sometimes described with phrase *battery included* and source code written in it as *runnable pseudocode*. Maybe because of this Python has become very popular among programmers in the academic field. The popularity in the academic field and Python's features convinced us to implement our platform also in this programming language. The newest version can be found at the home page of our platform [67].

In the process of implementation, we encountered several major and minor problems that needed to be solved. Most of the time these problems were tied to the specific programming language. This is why we decided to divide this chapter into three main sections, each devoted to the problems and solutions tied to the specific language.

6.1 Python implementation

As we already mentioned before, Python has very extensive standard libraries. This means, that we did not use any other library when implementing our platform.

Python implementation differs from the abstract architecture as it was implemented according to an older draft of the architecture. However, this is not a problem as it uses the same DS and MTS specification when communicating.

6.1.1 Agent

Agent in the Python implementation is a singleton, thus there can be only one instance. Agent takes over the platform's responsibilities, therefore an agent is the owner of DS

and MTS components. Agent is also owner of the contact list of remote agents found in the system and it provides the interface for DS to maintain it.

6.1.2 Discovery Service

For the implementation of the DS in Python we used two threads: announcer and receiver. The receiver thread is running in a loop, binding to the specified multicast group, until the object's stop() method is called and variable m_running is set to False. However, the Python's socket call recvfrom() is blocking, therefore the 'leave' message is sent in appropriate format (see Section 5.4) from the object's stop() method to unblock it. Every time it receives the data, it parses them. If the announced agent is not in the list of remote agents the Agent's agentArrived() method is called. If this agent already is in the contact list but with different flags or services, Agent's agentChanged() method is called.

For the announcer, we used the Timer class from Python threading standard libraries. Python's Timer class is created with an interval and a method to call after this interval passes. This method is called once, therefore a new Timer is set at the end of every announce. Announcer is sending an 'announce' message to the multicast group every second with the owner's information. It has a private counter and on every fifth iteration Agent's revalidateContactList() method is called. If an agent was not announced for a longer period (5 seconds by default) this agent is removed from the list and the method agentLeft() is called.

6.1.3 IOServer

IOServer is the Python implementation of the MTS described in Section 5.3. It uses BaseHTTPServer from Python's standard library. When sending messages, MIME-Multipart message is created first. Its parts are created from FIPA ACL Message and Transport Envelope in XML. However, MIMEMultipart can not be send directly with httlib, that we used in this implementation, because the resulting message will have MIMEMultipart message headers separated by double new line , which does not satisfy the specification. Because of this we are trimming the headers from the MIMEMultipart message and set its headers when creating a connection using the httplib.HTTPConnection class.

6.2 Java implementation

Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM). This allows us to run application on every device where the JVM is available. Because of this, Java is suitable for writing multi-platform applications.

Also, applications written in Java are often able to run on Android devices with minimal changes, as Android devices use Dalvik Virtual Machine.

Java implementation of our multi-agent middleware is based on earlier version of the abstract architecture then the one proposed in Section 5.1. In this implementation the MTS and DS components are singletons and agents are using their services. In the next sections we will describe the implementation of our middleware in Java language and problems that we encountered during this process.

6.2.1 jAgent

The jAgent class is the Java implementation of an Agent from Section 5.2. jAgent registers with DS and MTS components. It keeps his contact list of remote agents and provides interface for the DS component to maintain it. This interface invokes appropriate events. jAgent's main loop is event-driven, thus waiting for events such as messageReceivedEvent, agentArrivedEvent, etc.

6.2.2 IOServer

IOServer is our implementation of the MTS from the abstract architecture. It is implemented according to the lazy singleton design patter described in Section 3.4. In the IOServer, a HTTP server is running and serving requests form other platforms. A HTTP server is not part of Java standard libraries so we decided to implement our HTTP server using Oracle's package com.sun.net. For parsing and creating multipart messages we implemented our own Multipart class according to the RFC2046 specification[1]. Individual parts of the multipart message are instances of class Part, which is an extension of the class with the same name from the Sceye-Fi Photo capture project[2] For parsing FIPA ACL Messages we are using SAX parser from the Apache Xerces™Project [3].

6.2.3 Discovery Service

Discovery Service is our implementation of the DS component form the abstract architecture. It is composed of two threads - sender and announcer. Announcer extends the TimerTask class and periodically, every second, announces information about registered agents to the multicast group. It has internal counter that is incremented with every iteration. On every fifth iteration the revalidation of jAgent's contact list is executed.

[1] `http://www.ietf.org/rfc/rfc2045.txt`
[2] `http://code.google.com/p/sceye-fi/`
[3] `http://xerces.apache.org/`

The sender thread is running in a loop until the stop() method of the Discovery Service object sets the private boolean variable is_running to false. In Java, we encountered similar problem as in Python, i.e. that calling the receive() method on MulticastSocket is blocking. Similarly, the stop() method first sets is_running variable to false and then announces the multicast group that the jAgent is leaving with data packet in appropriate format.

6.3 C++ implementation

For C++ implementation we decided to use POCO C++ Libraries[4]. According to creators, POCO C++ Libraries are modern, powerful open source C++ class libraries and frameworks for building network- and internet-based applications that run on desktop, server and embedded systems [3].

Among others, POCO C++ Libraries are suitable for creating middlewares. They are very well suited for embedded systems as POCO-based applications are able to run on 75 MHz ARM9-based Linux systems with 8 MB RAM and 4 MB Flash [3].

POCO C++ Libraries support variety of platforms including Microsoft Windows, Linux, Mac Os X, Solaris, Embedded Linux (uGlibc, glibc), iOS, QNX and others. Poco C++ libraries are released under the Boost Software License [5].

In our implementation, we use several Poco libraries where standard C++ libraries are not available. We will discuss these cases in next subsections.

6.3.1 Threading

Although thread implementation should be multiplatform in the new C++11 standard [33], it is not so widespread yet. POCO C++ Libraries offer thread implementation that is multiplatform, available for every supported platform. Furthermore, POCO also offer Runnable and RunnableAdapter classes, an interface classes for thread entry points, similar to Runnable Class in Java.

Threads are important in our implementation, as we use several of them across our platform, including the implementation of the MTS and DS. Also every agent in the platform is running in its own thread.

6.3.2 IOServer

IOServer is our implementation of the MTS described in Section 5.3. We are using HTTP server from POCO C++ Libraries. This HTTP server is based on POCO TCPServer. In our case, the server is running in one thread and handlers of requests

[4] http://pocoproject.org/
[5] http://pocoproject.org/license.html

are taken from the default thread pool. IOServer is using POCO MultipartWriter and MultipartReader to read and write multipart messages.

The owner of the IOServer is the platform, thus it is responsible for destroying the object at the end of the program. IOServer creates the Message instances by parsing received data in XML format. For this purpose, we are using POCO's XML parser with Simple API for XML (SAX). SAX parser interface is an event-driven interface, thus the document is not loaded into the memory as a whole for parsing. This allows to parse large amounts of data with minimum memory consumption.

6.3.3 Discovery Service

Discovery Service is the implementation of the DS from the Section 5.4. For this implementation we are using MulticastSocket class from POCO Libraries, that is connected to the multicast group waiting in a loop for the data to receive. We decided to transfer the responsibility to maintain the contact list from the DS to the platform because of the future easier extension of the DS and future implementations of others DSs. Because of this, our DS is calling platform's method for adding agents to the contact list every time new data is received.

When starting the DS with its start() method, another thread is started along with the receiver. This thread is used for announcing information about agents registered to the platform. For this, we are using Timer and TimerCallback classes. The Timer class is an implementation of a thread-based timer and is started with the interval of 1000 milliseconds. Every time this interval expires, the timer callback is called, in our case the DS's inform() method. This method first retrieves the actual list of registered agents from the platform, and then sends the information about them in appropriate format described in Section 5.4. The announcing thread has its own counter, in every fifth iteration the platform's revalidateContactList() method is called.

6.3.4 Platform

Platform is the owner of every instance of the DS, IOServer and Agent object, but only one instance of the platform is available - it is a singleton. Our implementation of the platform is also the owner the list of registered agents and list of remote agents found across the platform. It provides methods for managing these lists, but never the direct access. For example, agents are able to write themselves to the list of registered agents by calling the subscribe() method. The DSs are able to keep the contact list of remote agents updated by calling addAgent() and revalidateContactList(). The platform is not providing the direct access because other components are running in separate threads, thus it can not guarantee the thread-safe access. Because of this, the platform is using POCO's RWLock when accessing these lists.

Chapter 7

Conclusion

7.1 Summary

We succeeded to implement a communication platform for agents in heterogeneous environment. This platform is implemented in three languages. Agents from different implementations are able to communicate with each other. For their communication they are using messages in FIPA-ACL, the language inspired by the Speech Act Theory. To transmit these messages, they are using REST-full HTTP protocol over TCP/IP. The C++ implementation is able to run on small less powerful embedded devices. We plan to use our platform in the research of ambient systems. The newest version of the platform can be found at the home page of the project[67].

7.2 Future work

Although our platform is fully working, there are some bugs that we find out, that need to be fixed in the future. We would also like to extend our platform by implementing more FIPA specifications, including FIPA SL languages, FIPA ACL Message representations in String and Bit-Efficient encoding and different interaction protocols. Nonetheless, we would like to implement new ways of discovering agents, e.g. Zeroconf/Avahi and XMPP/Jabber. But for us, the highest priority has the concept of a gateway agent.

Gateway agents

A gateway agent (GA) is a special agent that acts as a proxy for agents from other networks. GA maintains a list of registered remote agents. When a remote agent registers, GA announces remote agent's presence on the local network with its own transport specific address. Thus any message sent to the remote agent from the local network is sent to GA, which looks up the remote agent in its database and delivers

(forwards) it. Combined with the polling service, GAs offer convenience way to connect agents from different networks. There are two basic ways to use gateway agents[12]:

- remote agents register directly with gateway agents,
- a bridge agent registers all agents on his local network with a gateway agent on a remote network (and vice versa).

Similarly, a gateway agent that acts as a bridge to other message transport systems can be created, thus enabling interoperability with other FIPA compliant platforms.

Bibliography

[1] Analía Amandi and Henri Avancini. A Java Framework for Multi-agent Systems. *SADIO Electronic Journal of Informatics and Operations Research*, 3(1):1–12, 2000.

[2] Gertrude Elizabeth Margaret Anscombe. *Intention*. Blackwell, 1957.

[3] Applied Informatics Software Engineering GmbH. POCO C ++ Libraries Introduction and Overview. Technical report, Applied Informatics Software Engineering GmbH, 2012.

[4] John L Austin. *How to Do Things with Words*, volume 23 of *The William James lectures*. Harvard University Press, January 1963.

[5] Fabio Bellifemine, Via G Reiss Romoli, Giovanni Rimassa, and Agostino Poggi. JADE – A FIPA-compliant agent framework. Technical report, Telecom Italia, 1999.

[6] M. Ben-Ari. *Principles of Concurrent and Distributed Programming (Prentice Hall International Series in Computing Science)*. Prentice Hall, 1990.

[7] Jamal Bentahar. *A pragmatic and semantic unified framework for agent communication*. dissertation, Université Laval, Québec, 2005.

[8] Stefano Bromuri, Visara Urovi, Maxime Morge, Francesca Toni, and Kostas Stathis. A multi-agent system for service discovery , selection and negotiation. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent SystemsVolume 2*, pages 4–5, 2009.

[9] Tao Cheng, Zailin Guan, Liming Liu, Bo Wu, and Shuzi Yang. A CORBA-based multi-agent system integration framework. In *Proceedings. Ninth IEEE International Conference on Engineering of Complex Computer Systems*, pages 191–198. IEEE Comput. Soc, 2004.

[10] Philip R Cohen and Hector J Levesque. Communicative Actions for Artificial Agents. In Victor Lesser and Les Gasser, editors, *Proceedings of the International Conference on MultiAgent Systems*, pages 65–72. AAAI Press, 1995.

[11] JLG Dietz. Speech acts or communicative action? *Proceedings of the second conference*, pages 235–248, 1991.

[12] Vladimír Dziuban, Michal Čertický, Jozef Šiška, and Michal Vince. Lightweight Communication Platform for Heterogeneous Multi-context Systems : A Preliminary Report. In *Proceedings of the 2nd Workshop on Logic-based Interpretation of Context: Modelling and Applications (Log-IC 2011)*, number 1, 2011.

[13] RT Fielding. *Architectural styles and the design of network-based software architectures*. Dissertation, University of California, Irvine, 2000.

[14] Pablo R. Fillottrani. The multi-agent system architecture in SEWASIE. *Journal of Computer Science & Technology*, 5(4):225–231, 2005.

[15] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management - CIKM '94*, pages 456–463, New York, New York, USA, 1994. ACM Press.

[16] FIPA. History of FIPA.

[17] Foundation For Physical Intelligent Agents. FIPA Abstract Architecture Specification. Technical report, FIPA, 2002.

[18] Foundation For Physical Intelligent Agents. FIPA ACL Message Structure Specification. Technical report, FIPA, 2002.

[19] Foundation For Physical Intelligent Agents. FIPA Agent Management Specification. Technical report, 2002.

[20] Foundation For Physical Intelligent Agents. FIPA Agent Message Transport Protocol for HTTP Specification. Technical report, FIPA, 2002.

[21] Foundation For Physical Intelligent Agents. FIPA Agent Message Transport Service Specification. Technical report, FIPA, 2002.

[22] Foundation For Physical Intelligent Agents. FIPA Communicative Act Library Specification. Technical report, FIPA, 2002.

[23] Foundation For Physical Intelligent Agents. FIPA SL Content Language Specification. Technical report, FIPA, 2002.

[24] M R Genesereth and R E Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual, 1992.

[25] Michael R Genesereth and Steven P Ketchpel. Software agents. *Communications of the ACM*, 37(7):48—-ff., 1994.

[26] Michael R. Genesereth and Nils J. Nilsson. *Logical foundations of artificial intelligence.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[27] J P Georgé, M P Gleizes, Pierre Glize, and C Régis. Real-time simulation for flood forecast: an adaptive multi-agent system staff. *Proceedings of the AISB03 Symposium on Adaptive Agents and MultiAgent Systems*, pages 1–6, 2003.

[28] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. A jabber-based multi-agent system platform. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems - AAMAS '06*, page 1282, New York, New York, USA, 2006. ACM Press.

[29] Aaron Helsinger, Michael Thome, and T. Wright. Cougaar: a scalable, distributed multi-agent architecture. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 2, pages 1910–1917. IEEE, 2004.

[30] Jeffrey Hershfield. Rule Following and the Background. *Linguistics and Philosophy*, 28(3):269–280, June 2005.

[31] Michal Hevery. Global State and Singletons, 2008.

[32] Michael N. Huhns and Larry M. Stephens. Multiagent Systems and Societies of Agents. In Gerhard Weiss, editor, *Multiagent Systems A Modern Approach to Distributed Modern Approach to Artificial Intelligence*, volume 3 of *Intelligent Robotics and Autonomous Agents*, chapter 2, page 619. MIT Press, 1999.

[33] ISO/IEC. ISO/IEC 14882:2011. Technical report, ISO/IEC, 2011.

[34] KH Jacobsen. How to Make the Distinction Between Constative and Performative Utterances. *The Philosophical Quarterly*, 21(85):357–360, 1971.

[35] Brendan Jennings, Rob Brennan, Rune Gustavsson, Robert Feldt, Jeremy Pitt, Konstantinos Prouskas, and Joachim Quantz. FIPA-compliant agents for real-time control of Intelligent Network traffic. *Computer Networks*, 31(19):2017–2036, August 1999.

[36] Dezider Kamhal. Filozofia jazyka a jej metódy podľa J. R. Searla. In *Searle, J. R.: Rečové akty.* Kaligram, 2007.

[37] Jozef Kelemen. *Strojovia a agenty.* Archa, Bratislava, 1994.

[38] Mamadou Tadiou Kone, Akira Shimazu, and Tatsuo Nakajima. The State of the Art in Agent Communication Languages. *Knowledge and Information Systems*, 2(3):259–284, August 2000.

[39] Y. Labrou and T. Finin. Agent communication languages: the current landscape. *IEEE Intelligent Systems*, 14(2):45–52, March 1999.

[40] Yannis Labrou and Tim Finin. A Proposal for a new KQML Speci cation A Proposal for a new KQML Speci cation. *Discourse*, (TR CS-97-03), 1997.

[41] Yannis Labrou and Tim Finin. Semantics and Conversations for an Agent Communication Language. *Review Literature And Arts Of The Americas*, pages 235–242, 1998.

[42] LEAP. Lightweight Extensible Agent Platform, 2002.

[43] Andrej Lúčny. BUILDING COMPLEX SYSTEMS WITH AGENT-SPACE ARCHITECTURE. *Computing and Informatics*, 23:1–36, 2004.

[44] Andrej Lúčny. *Tvorba inteligentných systémov na báze architektúry Agent-Space*. PhD thesis, Univerzita Komenského v Bratislave, 2005.

[45] Andrej Lúčny. Multiagentové systémy Implementácia MAS, 2011.

[46] Tamás Máhr, Jordan Srour, Mathijs De Weerdt, and Rob Zuidwijk. Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Tramsportation Research Part C: Emerging Technologies*, In Press,(1):0, 2010.

[47] A Micsik, P Pallinger, and A Klein. Soap based message transport for the jade multiagent platform. In *8th International Conference on Autonomous Agents and Multiagent Systems*, pages 101–104, BUdapest, 2009.

[48] G. Miller. The Smartphone Psychology Manifesto. *Perspectives on Psychological Science*, 7(3):221–237, May 2012.

[49] Maxime Morge and Paolo Mancarella. The hedgehog and the fox . An argumentation-based decision support system. In *ArgMAS'07 Proceedings of the 4th international conference on Argumentation in multi-agent systems*, 2008.

[50] Martin Ngobye, Wouter T De Groot, and Theo P Van Der Weide. MULTI-AGENT SYSTEM INTERACTIONS. *Development*, 8(1):49–58, 2010.

[51] Dawn C Parker, Steven M Manson, Marco A Janssen, Matt Hoffman, and Mathew Hoffman. Multi-Agent Systems for the Simulation of Land-Use and Land-

Cover Change : A Review. *Annals of the Association of American Geographers*, 93(2):314–337, 2003.

[52] R S Patil, R E Fikes, P F Patel-Schneider, D McKay, T Finin, Th R Gruber, and R Neches. The DARPA Knowledge Sharing Effort: Progress Report. pages 777–788. Morgan Kaufmann, 1992.

[53] Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[54] Gauthier Picard, Carole Bernon, Marie-pierre Gleizes, and Université Paul Sabatier. ETTO : Emergent Timetabling by Cooperative Self-Organization. In *Engineering Self-Organising Systems*, pages 31–45. Springer Berlin / Heidelberg, 2006.

[55] Jeremy Pitt and Fabio Bellifemine. A Protocol-Based Semantics for FIPA ' 97 ACL and its Implementation in JADE. Technical report, Telecom Italia, 1999.

[56] Stefan Poslad. Review of FIPA Specifications. Technical report, FIPA, 2006.

[57] Bill Pugh. The Java Memory Model. Technical report, University of Maryland, 2008.

[58] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 3 edition, 2009.

[59] M D Sadek, P Bretier, and F Panaget. ARTIMIS: Natural Dialogue Meets Rational Agency. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence IJCAI97*, volume 15, pages 1030–1035. Morgan Kaufmann, 1997.

[60] Miro Samek. State Machines for Event-Driven Systems, 2009.

[61] John R Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[62] John R Searle. *Expression and Meaning: Studies in the Theory of Speech Acts*, volume 49. Cambridge University Press, 1979.

[63] SEWASIE.org. SEWASIE.

[64] M.P. Singh. Agent communication languages: rethinking the principles. *Computer*, 31(12):40–47, 1998.

[65] Katia Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph Giampapa. The RETSINA MAS Infrastructure. *Autonomous Agents and MultiAgent Systems*, 7(1):29–48, 2003.

[66] Marco Ughetti. JADE ANDROID ADD-ON GUIDE. Technical report, JADE, 2010.

[67] Michal Vince and Jozef Šiška. LCP: A Lightweight Communication Platform@ONLINE. http://dai.fmph.uniba.sk/~siska/lcp/, 2009.

[68] John Vlissides. To Kill A Singleton. Technical report, 1996.

[69] Ján Šilar. Porovnanie Austinovej a Searleovej teórie rečových aktov. Technical report, Univerzita Komenského v Bratislave, 2010.

[70] H Wang, Y Li, L Jin, and Y Liu. Multi-agents based fault diagnosis systems in MSW incineration process. In *2010 International Conference on Measuring Technology and Mechatronics Automation ICMTMA 2010*, volume 2 of *International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2010*, pages 721–724. IEEE Computer Society, 2010.

[71] Michael Wooldridge. *An Introduction to MultiAgent Systems*, volume 86. John Wiley & Sons, 2002.

[72] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(02):115, July 1995.

[73] Kamer Ali Yuksel and Aytul Ercil. Augmenting Conversations through Context-Aware Multimedia Retrieval based on Speech Recognition. In *5th Workshop on Human-Computer Interaction and Information Retrieval (HCIR 2011)*, pages 1–4, Mountain view, CA, 2011.

Appendix A

Extended FIPA ACL Message Representation in XML Specification

```
<!-- Document Type: XML DTD
     Document Purpose: Extended Encoding of FIPA ACL messages in XML-->

<!-- Possible FIPA Communicative Acts. -->
<!ENTITY    %communicative-acts           "accept-proposal
| agree
| cancel
| cfp
| confirm
| disconfirm
| failure
| inform
| not-understood
| propose
| query-if
| query-ref
| refuse
| reject-proposal
| request
| request-when
| request-whenever
| subscribe
| inform-if
| inform-ref
| proxy
```

```
| propagate">

<!-- The FIPA message root element, the communicative act is
     an attribute - see below and the message itself is a list
     of parameters. The list is unordered. None of the elements
     should occur more than once except receiver. -->
<!ENTITY    %msg-param                    "receiver
| sender
| content
| language
| encoding
| ontology
| protocol
| reply-with
| in-reply-to
| reply-by
| reply-to
| conversation-id
| user-defined">

<!ELEMENT   fipa-message                  ( %msg-param; )*>

<!-- Attribute for the fipa-message - the communicative act itself and
     the conversation id (which is here so an ID value can be used). -->
<!ATTLIST   fipa-message                  act ( %communicative-acts; ) #REQUIRED
                                          conversation-id ID #IMPLIED>

<!ELEMENT   sender                        ( agent-identifier )>

<!ELEMENT   receiver                      ( agent-identifier+ )>

<!-- The message content.
     One can choose to embed the actual content in the message,
     or alternatively refer to a URI which represents this content. -->
<!ELEMENT   content                       ( #PCDATA )>
<!ATTLIST   content                         href CDATA #IMPLIED>

<!-- The content language used for the content.
     The linking attribute href associated with language can be used
     to refer in an unambiguous way to the (formal) definition of the
     standard/fipa content language. -->
```

```
<!ELEMENT   language                        ( #PCDATA )>
<!ATTLIST   language                          href CDATA #IMPLIED>

<!-- The encoding used for the content language.
     The linking attribute href associated with encoding can be used
     to refer in an unambiguous way to the (formal) definition of the
     language encoding. -->
<!ELEMENT   encoding    ( #PCDATA )>
<!ATTLIST   encoding     href CDATA #IMPLIED>

<!-- The ontology used in the content.
     The linking attribute href associated with ontology can be used
     to refer in an unambiguous way to the (formal) definition of the
     ontology. -->
<!ELEMENT   ontology                        ( #PCDATA )>
<!ATTLIST   ontology                         href CDATA #IMPLIED>

<!-- The protocol element.
     The linking attribute href associated with protocol can be used
     to refer in an unambiguous way to the (formal) definition of the
     protocol. -->
<!ELEMENT   protocol                        ( #PCDATA )>
<!ATTLIST   protocol                          href CDATA #IMPLIED>

<!ELEMENT   reply-with                     ( #PCDATA )>
<!ATTLIST   reply-with                       href CDATA #IMPLIED>

<!ELEMENT   in-reply-to                     ( #PCDATA )>
<!ATTLIST   in-reply-to                      href CDATA #IMPLIED>

<!ELEMENT   reply-by                          EMPTY>
<!ATTLIST   reply-by                          time CDATA #REQUIRED
  href CDATA #IMPLIED>

<!ELEMENT   reply-to                        ( agent-identifier+ )>

<!ELEMENT   conversation-id                 ( #PCDATA )>
<!ATTLIST   conversation-id                   href CDATA #IMPLIED>

<!ELEMENT   agent-identifier                ( name,
  addresses?,
```

```
  resolvers?,
  user-defined* )>

<!ELEMENT   name                            EMPTY>

<!-- An id can be used to uniquely identify the name of the agent.
     The refid attribute can be used to refer to an already defined
     agent name, avoiding unnecessary repetition. Either the id
     OR refid should be specified, (both should not be present at the
     same time). -->
<!ATTLIST   name                            id ID #IMPLIED
  refid IDREF #IMPLIED>

<!ELEMENT   addresses                     ( url+ )>

<!ELEMENT   url                             EMPTY>
<!ATTLIST   url                             href CDATA #IMPLIED>

<!ELEMENT   resolvers                     ( agent-identifier+ )>


<!ELEMENT   user-defined                  ( flag,service,#PCDATA )+ >
<!ATTLIST   user-defined                    href CDATA #IMPLIED>

 <!ELEMENT   flag                         ( #PCDATA ) >
 <!ELEMENT   service                      ( #PCDATA ) >
```

Appendix A

Homepage of the project

You can download the newest version of our platform at the home page `http://dai.fmph.uniba.sk/~siska/lcp/` of the project.