CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Electrical Engineering
DEPARTMENT OF CONTROL ENGINEERING

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Automation and Systems Technology
Automation Technology Laboratory

MASTER'S THESIS

INFORMATION AGENTS IN PROCESS AUTOMATION

Milan Fajt

Supervisor: **Assoc. Prof. Jan Bílek (CTU)**

Prof. Dr. Tech. Aarne Halme (HUT)

Instructor: **Dr. Tech. Pekka Appelqvist (HUT)**

Prague, 2004

Czech Technical University in Prague - Faculty of Electrical Engineering
Department of Control Engineering
&
Helsinki University of Technology - Department of Automation and Systems Technology
Automation Technology Laboratory

Submission of Master's thesis

Student: Milan Fajt

Field of study: Control Engineering

Topic of the thesis: Information Agents in Process Automation

Master's thesis elaboration:

1) Acquaint yourself with fundamentals of information access in process automation.
2) Acquaint yourself with fundamentals of agent technology.
3) Analyze and propose the use of information agents in process automation.
4) Design, implement and test a demonstration scenario of information agents in process automation.

Supervisor: Assoc. Prof. Jan Bílek
Prof. Dr. Tech. Aarne Halme

Instructor: Dr. Tech. Pekka Appelqvist

Date of submission: November 2002

Date of commitment: May 2004

Master's thesis originality statement

I pronounce that I have elaborated this Master's thesis independently and that I have used only materials (literature, projects, SW, etc.) referred in this Master's thesis.

I do not have any objection against the use of the material contained in this Master's thesis by other persons for any noncommercial purposes.

Prohlášení o původnosti díla

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

Espoo, June 25, 2003 Milan Fajt

Abstract

Agent technology's potentials to enhance current approaches of information access in process automation systems (PAS) such as OLE for Process Control (OPC) are the prime objectives of this thesis. In agent technology, here considered as a new programming paradigm, software systems are built from functionally independent program units called agents that are fully autonomous and active in contrast to the objects in object-oriented programming.

The main focus of the theoretical part is given to information agents in process automation. Individual aspects of information agents in general, such as requirements, reliability, architecture and implementation are studied and summarized from referenced materials and next they are applied and analyzed with regard to information agents in process control. Advantages and disadvantages of information agents are discussed and comparison with OPC is provided.

In the practical part of the thesis, an information access module and an agent message evaluating mechanism were implemented. Both were tested and presented in the demonstration scenario where information agents were connected to a physical laboratory system. Agents were monitoring and diagnosing the condition of the running system and they were notifying an external agent about its changes.

Contribution of this thesis is in the problem specification of information access in general and in a brief analysis of information agents in process automation. Further, developed software will be used and extended in the consecutive research of agent enhanced process automation.

Anotace

Tématem této práce je využití agentní technologie k zdokonalení stávajících informačních systémů v automatizaci procesů jako například OLE for Process Control (OPC). Agentní technologie je zde chápána jako nový programovací přístup k tvorbě komplexních systémů. S využitím agentní technologie jsou softwarové systémy tvořeny z nezávislých programových jednotek - agentů. Tyto agenti jsou plně autonomní a aktivní programy narozdíl od objektů v objektově orientovaném programování.

Hlavní pozornost teoretické části práce je věnována informačním agentům v automatizaci procesů. Ze zdrojů uvedených v referencích jsou zde shrnuty jednotlivé vlastnosti a charakteristiky informačních agentů, jako jsou požadavky, spolehlivost, architektura a implementace, a ty jsou dále aplikovány a analyzovány s ohledem na informační agenty v automatizaci procesů. Dále jsou zde zmíněny výhody a nevýhody informačních agentů a je zde provedeno jejich porovnání s OPC technologií.

V praktické části této diplomové práce byl navrhnut a implementován programový informační modul a algoritmus pro vyhodnocování agentních zpráv. Jejich funkčnost byla demostrována na laboratorním fyzikálním systému, ke kterému byli informační agenti připojeni. Agenti prováděli diagnózu stavu běžícího systému. Změny stavu systému byly agenty zasílány vnejšímu pozorovateli představovaným dalším informačním agentem.

Přínosem této diplomové práce je specifikace problému přístupu k informacím obecně a ve stručné analýze informačních agentů v automatizaci procesů. Vyvinutý software bude dále využit v pokračujícím výzkumu aplikování agentní technologie v automatizaci procesů.

Preface

This Master's thesis was elaborated with a great help of the agent team at the Automation Technology Laboratory at the Helsinki University of Technology in Finland within my exchange study program in the school year 2002 - 2003. This Master's thesis could never be created without all the support that was given to me during the time I was working on the thesis. I would like to express my profound appreciation to all of you who had a positive influence on my work.

I would like to thank Professor Aarne Halme, head of the Automation Technology Laboratory, for his permission to work on my thesis in his laboratory, for all his support, for his attitude to make things simple and in a positive mood and for the great environment that the laboratory is under his supervision.

My sincere thanks also belong to Associate Professor Jan Bílek from the Control Engineering Department at the Czech Technical University. His great readiness to help me and his overall very positive attitude, especially when dealing with my problems, made on me the best impression.

I was very happy for the honor to meet and to work under supervision of Dr. Pekka Appelqvist who was giving me all the time critical feedback and very constructive advises for my thesis elaboration. During work with him, I got lot of invaluable experience. Thank you Pekki for your patience and for your great help.

Very big thanks come to Mr. Ilkka Seilonen and Mr. Teppo Pirttioja for their great support, for many technical advises and especially, for their Finnish humor. The way they think is very logical and the variety of questions that they gave me never let me stop thinking about my work.

My thanks also belong to the personnel of the Automation Technology Laboratory. They provided me very nice and pleasant working environment.

Special thanks belong to Karoliina Auvinen, ihanan mielen omaava tyttö, jokaisesta minuutista hänen kanssaan ja saaden minut ymmärtämään ettei elämä ole pelkkää työtä ja opiskelua. Hänen kanssaan, oleskeluni Suomessa muodostui erityisen intensiiviseksi ja Suomesta tuli jopa vielä kauniimpi.

I am glad that I can express here my gratefulness also to all my friends, especially to Jirka Dlouhý, Jaromír Sršeň, Michal Doleček, Karel Honzl, Daniel Švanda, Daniel Ehrenberger and many others for being there outside and for letting me to know them.

Finally, I would like to thank to my whole family that gave me all necessary space and support I needed.

Espoo - Finland, June 17, 2003 Milan Fajt

Table of Contents

Table of Figures

Table of Tables

Table of Abbreviations

ACC	Agent Communication Channel
ACL	Agent Communication Language
ACLs	Access Control Lists
AMS	Agent Management System
BDI	Belief Desire Intension
CAA	Content Adaptation Agent
CIA	Cooperative Information Agents
CCL	Constraint Choice Language
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CycL	Language of formal logic
DA	Data Access
DCOM	Distributed Common Object Model
DCS	Distributed Control Systems
DDE	Dynamic Data Exchange
DF	Directory Facilitator
DLL	Dynamic Link Library
DSS	Decision Support Systems
DX	Data eXchange
EXE	Executable
FIPA	The Foundation for Intelligent Physical Agents
FIPA-OS	FIPA Open Source
FIPAPC	FIPA for Process Control
GUI	Graphical User Interface
HDA	Historical Data Access
HTTP	Hyper Text Transfer Protocol
JESS	Java Expert System Shell
JINI	Java-based set of APIs and runtime conventions for building of distributed systems
JVM	Java Virtual Machine
KIF	Knowledge Interchange Format
MAS	Multi Agent System
MES	Manufacturing Execution System
MS	Microsoft
MTP	Message Transportation Protocol
MTS	Message Transportation System
ODL	Ontology Definition Language

OLE	Object Linking and Embedding
OLTP	On-Line Transaction Processing
OML/CKML	Ontology Markup Language / Conceptual Knowledge Markup Language
OPC	OLE for Process Control
OPM	Object-Process Methodology
P2P	Peer to Peer
PAS	Process Automation System
PID	Proportional Integral Derivative
QoS	Quality of Service
RDF	Resource Description Framework
RETSINA	Reusable Environment for Task-Structured Intelligent Networked Agents
RMI	Remote Method Invocation
SCADA	Supervisory Control And Data Acquisition
SL	Semantic Language
SOAP	Simple Object Access Protocol
SPC	Statistical Process Control
UDDI	Universal Description, Discovery and Integration
VB	Visual Basic
VBA	Visual Basic for Applications
WMTP	Wireless Message Transport Protocol
WSCL	Web Services Conversation Language
WSDL	Web Services Description Language
XML	Extensible Markup Language

1 Introduction

1.1 Background of the thesis

Information access in process automation constitutes data transmission among plant-floor devices, higher-level control, management and monitoring applications. Data from an automated process are important for analyses of product qualities and for controlling and monitoring the process itself and its behavior. Typical industrial processes are complex systems consisting of a number of various devices designed and created by different manufacturers. All of these devices need to be controlled and synchronized to make the whole system running properly. Information and control systems have to collect, understand and evaluate data from all parts of the process. This means, they have to contain the overall knowledge about the process and therefore they can be very sophisticated and their design difficult and prone to errors. Changing properties of such complex systems is deceptive, especially at runtime.

Agent technology introduces autonomous entities called agents that behave and act according to their goals and with loosely coupled external control. To exhibit desired behavior, agents make decisions based on the knowledge of the situation in the surrounding environment. This knowledge is obtained through receptors, through communication with other agents and often it is partially pre-programmed. Agents can serve for many purposes and therefore they can be implemented in many different ways that best fit the particular requirements. They can possess physical bodies or be purely software agents. The important property that agents have is the ability to intelligently communicate with each other. By intelligent communication is understood human-like communication that is based on a set of words and rules with the possibility to express high variety of information. Communication allows agents to share knowledge and intentions they have and thus to do rational decisions corresponding to actual situation in their environment. The meaning of the words used in communication and their mutual relations are specified by ontologies. Using the same ontologies ensures that all agents understand to messages equally.

The work of this thesis is a part of the research where application of agent technology in process automation is studied. In this research, the FIPA for Process Control (FIPAPC) agent platform was designed and implemented as a testing tool of agents in process automation systems (PAS). The FIPAPC is an extension to the FIPA Open Source (FIPA-OS) agent platform and an introduction to it and its description can be found in [Pirttioja, 2002]. During this research, it has revealed that information access in process automation is a subtopic big enough to be treated separately in its own research and thus a theme for this Master's thesis has been established.

1.2 Objectives

The goal of this thesis is to give an introduction to agent technology in information access in process automation and to implement and demonstrate an example application of information agents with the use of the FIPAPC agent platform. Agent technology has many characteristics, based on which agents are expected to prolong the service life of the process automation systems and to increase their safety, reliability and efficiency. Application of information agents can, for example, include sophisticated data preprocessing such as diagnosing of the working state of a process automation system. It is common in process automation that product samples are regularly taken and manually analyzed in the quality laboratories to guarantee a required quality level. Results from these analyses are stored in databases. Information agents can thus as well access and process offline data from a process automation system to improve current product quality and to extract hidden coherencies among various process factors.

The introductory part of this thesis summarizes the information sources dealing with information access in process automation and with agent technology. All sources, from where details were taken, are referenced at the end of this thesis.

The theoretical part of this thesis deals with information agents in PAS. Information agents are regarded as autonomous and independent parts of PAS. They are proposed to form a hierarchical structure corresponding to the structure of the PAS. Every agent has information depending on the system part that it is responsible for and it provides this information to other agents. Agents possess device-specific information and algorithms used for controlling, diagnosing or monitoring. Thus they are able to provide higher level information by evaluating data and knowledge they have about PAS. These agent systems are also able to satisfy higher variability of information queries due to the agent communication languages they use.

In the practical part of this thesis, information agents are implemented to show their advantages over other information access technologies. A demonstration of a scenario, where agents provide complex information about a laboratory PAS to another agent, is presented. The laboratory PAS consists of two water tanks, a water jet pump and water valves that are represented by hierarchically structured agents. Every agent is responsible for its particular system's part and holds information relevant to it. In the test scenario, the highest-level agent, that represents the whole physical system to the outside world, is asked by an external agent to monitor the running state of the system and to send a notification in the case that the state has changed. The highest-level agent resends the query to all subagents. Every agent that has received the query starts periodically to collect data that is necessary to evaluate the condition of the system's part the agent is responsible for. In the case, when an agent observes that the running state has changed, a notice message is sent to a higher-level agent and so on until the message propagates through the top level agent to the external agent. In the implementation of this

demonstration scenario, only one agent carries out real system diagnosis. Other agents automatically evaluate the running system state as being correct.

An important part of the practical work consists from a mechanism for agent message interpretation. This mechanism enables interpreting of messages based on parsing their grammar tree representation. With the help of this mechanism, agents should be able to process a great variety of simple messages.

As a conclusion of this thesis, agent technology enables decomposition of a complex information system of a large PAS into a number of autonomous, independent and simpler parts represented by information agents. Therefore, complexity of a traditional central information system is distributed over the whole PAS. Individual parts of this information system can be designed separately and more easily in contrast to current approaches used for information access in process automation such as OPC. This is shown in the presented information agent demonstration, which is simple on one hand but which is showing the proposed concept of information agents in a real PAS on the other hand.

Technical knowledge about PAS is not stored centrally but is distributed among individual agents. This should lead to higher scalability, reliability, flexibility and usability of information systems. Thanks to agent-based information systems, client applications will not need to possess any technical details about PAS. They will mainly focus on presenting data to human users.

1.3 Structure

This thesis has the following structure:

Chapter 2: *Information access in process automation.* This chapter is a literature review of information access, its purpose and requirements on it. The current approaches for information access in process automation are introduced, specifically the DDE protocol and the OPC Specifications.

Chapter 3: *Agent technology and information agents.* This chapter is a literature review of agent technology. In the first part, basic concepts, ideas, advantages and architecture of agent technology are presented. The second part deals with the role of agent technology in information access and how agent technology can enhance the current approaches for information access.

Chapter 4: *Concept of information agents in process automation.* This chapter is the theoretical part of this Master's thesis. Use of agent technology in information access in process automation is studied. The advantages and disadvantages of agent-based approach over traditional approaches are discussed.

Chapter 5: *Designing information agents.* This chapter together with the following chapter is the practical part of this Master's thesis. It presents the proposition of an information access module for the FIPAPC agents. Also fundamentals and design of an agent-messages evaluation mechanism are introduced.

Chapter 6: *Implementation and the test of information agents.* This chapter presents a description of the testing environment and the scenario that was proposed for demonstration of the information access module. The results obtained from running the test scenario are at the end of this chapter.

Chapter 7: *Conclusions.* This chapter evaluates the results obtained from tests and summarizes the advantages and the disadvantages of agent technology in information access in process automation.

2 Information access in process automation

As stated in the Merriam-Webster Dictionary, one of the many definitions describing information is: "*knowledge obtained from investigation, study, or instruction*". Information is necessary for farther development of all human activities and therefore there is a tendency to continuously increase the amount of collected and stored information. This fact forces researchers to develop and improve tools for storing and accessing information. The following chapter is a literature review of information access and information retrieving architecture with regard to process automation.

2.1 The concept and the purpose

The amount of stored information is said to double every year. This development creates big challenges for information technology. Not just where and how to store information but also how to access this information needs to be addressed. Without a fast and efficient information access, the benefits from large quantity of stored information would not be so significant.

Information access means retrieving of data from data sources. To accomplish information access, a data source, an information channel and a data request that initiates a data retrieving process, are involved. The data request starts the data retrieving process according to requirements on sought information, for example a relation of requested data to a specific domain.

An example of information access can be the Manufacturing Execution System (MES). It is used to provide information about production activities across an enterprise to the Enterprise Resource Planning (ERP) system. Provided information enables optimization of production process from accepting a new order to completing products and thus it corresponds to faster responses to changing output and output's property requirements. The purpose for collecting data is, in this case, divided into the following categories:

- *Quality assurance* - data corresponds to product qualities which are compared with the required ones to achieve a desired product quality level.
- *Performance analysis* - data from manufacturing process represent individual states of manufacturing process and helps to conduct performance analysis.
- *Maintenance management* - data contain values of process variables that can be used to track and foresee a system failure.
- *Documentation* - data serve for statistics about process running and made products.
- *Product traceability* - data relating to production condition of an individual product.
- *Supplier management* - logistics data serve for material order planning.

In total, information from the manufacturing process and it's appropriate evaluation improve among others production flow, an on-time product delivery, the amount of created products and thus mainly and purposefully money profit. [Fortu, 2002].

In process automation, accurate process control depends on reliable and fast enough retrieving of data representing the current process state. Data from the control system, actuators and sensors themselves are also needed to ensure reliability of the control process. Besides the process level, data are also needed at the operational and management levels for example by a process operator or production manager. In this case, data are often visualized in the form of curves and dynamic graphical objects representing state of the running process. Another use of data is for later evaluation and statistics.

2.2 Requirements

Information access system provides data for many purposes and for many different users' roles. In general, information access has the following properties whose importance depends on the individual cases where information access is employed:

- *Reliability* - corresponds to the functionality of information access itself and depends on the implementation, i.e. on the used technologies and the way they are applied, for example whether is used back-up or not.
- *Accuracy* - means the ability to provide data according to given requirements. Accuracy depends on the format in which data are available and on meta-information, if any is provided with data and which describes data meaning.
- *Promptness* - corresponds to the time needed to obtain required data. It depends on the computing power of the information access system, algorithms used to search data and throughput of the transmission channels.
- *Flexibility* - means the ability to provide data from different data sources through different information channels to different users in different formats.
- *Data validity* - corresponds to the ability to provide useful valid data.

Reliability of information access to the parts of the system directly participating in control is extremely important in process automation. Automation systems, where running-process interruption would be very expensive, must contain alternate information channels. Other systems have to posses at least an algorithms or a sequence of actions to be performed in the case of an information system's failure.

Accuracy of information received from sensors depends on a resolution with which information was sensed. The resolution of measurement depends on two key

factors. Firstly, on the physical process that is being controlled and its characteristics. Different processes require different accuracy of measurements. For example in the food industry, cooking time does not need to be measured with higher accuracy than seconds. Secondly, the resolution depends on the sensors themselves, on their capabilities. A measurement should not be finer than the measurement error of a used sensor. In all cases, accuracy of measured information needs to enable proper running of the given process and not be limiting.

Promptness with what information needs to be provided depends on individual cases of automation processes. Promptness has to be faster than the dynamics of the controlled processes, in process automation at least ten times faster.

Flexibility of data provided from and to different devices is very important in process automation. The current market is characterized by products from many companies and their mutual communication is necessary for their corporate use. Moreover, nowadays many applications (SCADA packages, databases, spreadsheets, etc.) are already in use and they also must be taken into account when designing information system architecture. Data are usually provided in the format that is chosen at the design time of the controlled system and later enhancements to the system are designed to use the same format.

Data validity plays important role in all types of information access and is mostly connected with the dynamics of a data source. If data are not delivered at time, they become useless and even harmful when used for control.

From more general point of view, information access itself is a part of information system architecture that represents the whole framework of physical and software systems enabling information access from miscellaneous data sources. Requirements, which such information system architecture has to address, are according to [Barbuceanu et al., 1994]:

- *Information access* - provides data from data sources connected through a corporate network; allows information to be inferred from various data sources and provides data in the required format.
- *Consistency maintenance* - allows detection of inconsistencies among data models and stored data and provides tools and methods for removing them.
- *Monitoring and automation* - monitors information changes and automates information updates, informs interested people about specific events, for example when specified information is obtained.
- *Cooperative work* - allows people and computers to work together as a team.
- *System integration* - allows a system to be built from independently developed components and to be easily used and maintained.

The amount of data collected from a process depends on two factors. First, it is the complexity of controlled systems. It is a common practice to collect more data than it is necessary to increase robustness of control. Second, it depends on the frequency with what data are collected. This is associated with the dynamics of the controlled system and on the precision of control.

[OPC Overview, 1998].

2.3 Users

Users of the information system in process automation are those who specify the requirements on it. They have different roles in the automation process maintenance and management and therefore individual users need different types of information with different requirements. Their roles can be split into:

- *Online operator*
- *Production manager*
- *Technology manager*
- *Service-man*
- *Researcher*
- *Mobile remote operator*
- *Program applications and devices*

The online operator directly supervises a process automation system whether it runs correctly. He uses monitoring systems that display the current values of variables and their boundaries textually or graphically. The online operator is one of the first persons who know about any system failure and who make the first actions to supersede the failure. He needs an access to the current values of all system variables, their set points and their limitations. He can also need a description of all variables, time development of variables, relations among individual variables and relations among variables and actions that is the process automation system capable to perform.

The production manager is interested in data statistics and in hidden data relations that can be obtained with help of the data mining techniques. The production manager is responsible for material ordering, production planning and negotiating orders with customers. Required information represents offline system data such as a description of data, historical values of data and their statistical properties, relation between amount of input material and out-coming products, time severity of the production of individual products and statistical data about system failures and wasters.

The technology manager is responsible for improving the process technology. He needs both data statistics and variable developments. He evaluates this information and according to it he is able to change process constants or a process structure. The technology manager needs offline data such as historical system data and its statistics, the quality of products in individual production phases and time development of all system variables.

The service-man is required to find and remove system defects. He needs data representing system variable values and their developments and deviations from expected developments, information about other failures that have happened in the past and limitations that have been broken. According to this information, the service man is able to find the reason of a system failure or to anticipate it in the future.

The researcher is studying behavior of a running system and is testing his theories. It is not common that the researcher can freely manipulate with the system's settings according to his needs. It is necessary to keep the system running and to prevent production losses. Therefore, the researcher's work is limited. Data required are offline data representing time dependencies of individual variables for different system constants settings, descriptions of constants and variables and data about physical properties of the system.

The mobile remote operator can represent the previous mentioned roles. The difference is in how and from where data are accessed. Mobile remote operators use personal user assistants (PDA), mobile phones or other similar devices to wirelessly connect to an information system and to retrieve data from it. This type of connection is characterized by a relatively slow data transfer and by limited display capabilities of used devices. Remote operators need to access system settings, the current values of all system variables as well as statistical data. They want to be informed about selected events that happen in the system for example a shortage of input material or a system failure.

Program applications and devices represent information, control and other program applications and devices that are part of the automation process system. Their data requirements vary from their tasks and purposes but their functionality is hard coded at design-time and does not change much in production process.

2.4 Implementation

Implementation of the information system corresponds to its hardware and software representation. Implementation spans from the hardware platform and appropriate software tools selection to the high-level knowledge-based problem solving. Nowadays, hardware and software components are being rapidly developed and information system architecture designers have many options to choose from. The final choice depends on experience, which the designer has with particular hardware devices

and software tools, on the suitability of individual products for the given task and on the price of individual products. In general, implementation of the information system is a compromise of the mentioned requirements.

In practice, information system architecture incorporates many heterogeneous data sources. To ensure reliable communication to/between them, designers have the following modern tools that are platform independent and allow future system enhancements, further development and modifications.

As for the data format, the last few years increasing attention is paid to the XML format. XML is a simple text code using marks to express the inner data structure according to a few primitive rules. The XML inner structure can be described and validated by another XML code called the XML Schema. XML is software and platform independent format supporting interoperability between different parts of the information system.

As for the data transfer, Distributed COM (DCOM), Remote Method Invocation (RMI) and Web Services are few of the examples. All of them are suitable for communication among distributed systems. DCOM technology is mainly used in the MS Windows platforms. It uses a binary format in which data are sent. RMI is Java based technology that can be used in all platforms running the Java Virtual Machine (JVM). Sent data are also in the binary format. Web Services use the worldwide spread HTTP protocol to carry data in the textual Simple Object Access Protocol (SOAP), which is in the XML format.

Information access implementation can be done in many ways how it is, actually, done in reality. Generally, information architecture in process automation is shown in Figure 2.1. This architecture is divided into the following layers:

- *Field Management* - contains mainly actuators and sensors, provides information about the state of controlled process, about the states of individual devices, configuration parameters, etc.
- *Process Management* - contains Distributed Control Systems (DCS) and Supervisory Control And Data Acquisition (SCADA) systems that control and monitor running processes.
- *Business Management* - uses collected process information in business management and provides this information to other applications.

[OPC Overview, 1998]

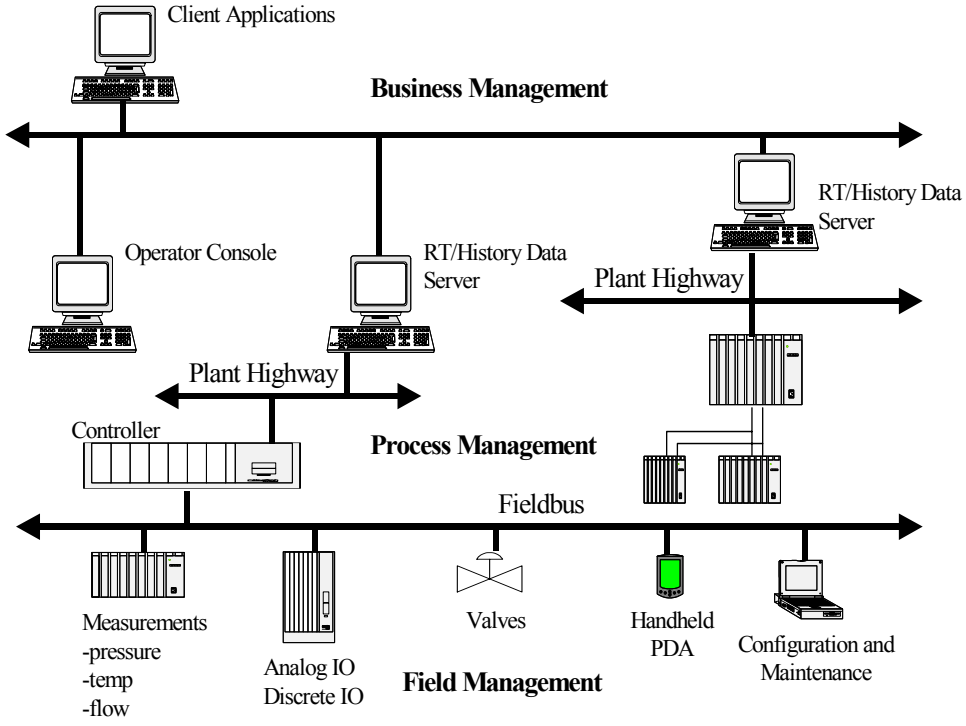

Figure 2.1 Process control information architecture (adopted from [OPC Overview, 1998]).

2.5 DDE

DDE stands for the Dynamic Data Exchange. It is a protocol represented by set of Windows-based messages and guidelines. Through this protocol, Windows applications can exchange all sorts of data for example text strings, binary data blocks or commands. One application is always providing some specific data (this application is called a DDE server) to another application (a DDE client). Applications can use DDE for single data transfers or for continuous data exchanges. Windows messages contain only two parameters (wParam and lParam) that can be used for sending only a few bytes of data. For passing larger data structures, these parameters need to contain pointers to these data structures. The DDE protocol specification exactly defines the use of wParam and lParam parameters for most of the possible situation of the data transfer.

Because DDE is a message-based protocol, whole communication is managed by passing defined DDE messages between the DDE server and the DDE client. The DDE protocol also allows the DDE client application to send commands to the DDE server. What commands is the DDE server executing, if any at all, depends on the particular DDE server and it should be well documented in the user manuals. The scope of use of the DDE protocol can be extended from a single computer to the in-network-connected computers by using a DDE network version the NetDDE protocol.

Every data item provided by the DDE server is uniquely identified. An identifier is composed from three parts. The first part specifies an application name. This is mostly the DDE server's application name without the .exe extension. The second part represents a DDE Topic, which determines a group or a data category in the server. The third part is a DDE Item Name, which is unique in the given DDE server.

Data exchange between the DDE server and the DDE client is called a conversation. Every successful conversation consists from the following parts:

- *Initiation.* It is started by a DDE client application by broadcasting a specific DDE message to all the running applications. The DDE message contains the DDE Application Name, the DDE Topic and the DDE Item Name. In the case that a server application that provides requested data is running, an answer message is sent and the Windows operating system opens a DDE link between these two applications.

- *Exchange.* During the exchange phase data are transferred by one of the following method:

  - *Cold link* - the client requests data and the server supplies them.
  - *Warm link* - the client requests data and also wants to be informed whenever they change. The server informs the client about data change but the transfer is made only when the client approves it.
  - *Hot link* - the client requests data and also wants to be informed whenever they change. In the case of a change, data are transferred automatically.

  The data request can also represent a command. In this case, the server performs the requested operation.

- *Termination.* It can be done by either the server or the client application. Opened channels are closed and the established DDE links are terminated. Closing one application involved in the communication also causes termination of all DDE links.

The DDE protocol can be used to implement broad range of features in the Windows applications. For example linking to real-time data, such as to stock market updates, scientific instruments, process control or creating documents with charts and tables using DDE for their dynamic update and many others are possible.

In summary, DDE communication is relatively easy to implement comparing to other technologies (COM). The DDE server can serve to a wide range of clients and can be updated without any need for their change. DDE is immune to the problems associated with different versions of DLLs on different machines. It can be small and

self-contained. The DDE clients do not threaten correct running of the DDE server by being busy or slow because DDE communication is asynchronous and is carried out by the standard message handling.

[Angelfire, 2003], [MathWizards, 2003], [MSDN, 1992], [TALtech, 2003].

2.6 OPC Specifications

OLE for Process Control (OPC) is a standardized set of interfaces that allow well defined and unified communication among different clients and different devices. It was created by the OPC Foundation that published its first specification in 1996. The OPC Foundation board members are represented by people from major industry-control companies such as Siemens AG, Rockwell Automation, Honeywell, Toshiba, Microsoft, etc. Nowadays, OPC is the state-of-the-art in information access in process automation.

2.6.1 Motivation

Before OPC, information access in process automation was consisted of various applications that were using their own drivers to access data from devices. Much effort was given to development of the similar drivers by different companies. Created drivers did not support all the features of individual devices. For new devices, new drivers had to be created and already existing client applications had to be modified. Access conflicts occurred when two different drivers were accessing the same device. Manufacturers creating drivers for their devices were not able to satisfy requirements of all the client applications. Therefore, OPC was created to provide a common way for data access to numerous data sources represented mainly by devices on the factory floor.

2.6.2 Implementation

Today, manufacturers create only one OPC Server for the specific device or group of devices. The OPC Server is an application that knows how to access data from certain devices and how to provide them to client applications. OPC Servers serve to all kinds of clients (an on-line operator application, an office application, etc.). On the other hand, clients need to know only one way how to access data from all OPC Servers, see Figure 2.2.

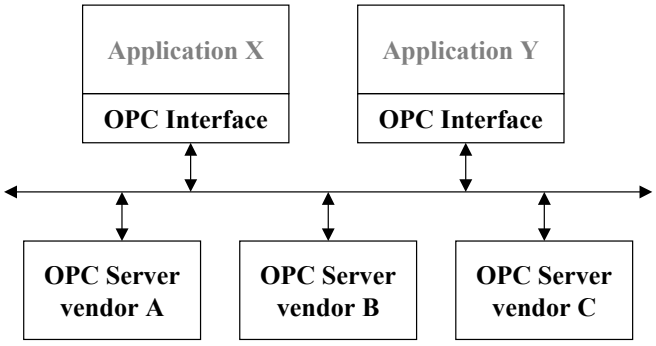

Figure 2.2 Applications Working with Many OPC Servers, (adopted from [OPC Overview, 1998]).

OPC uses the Microsoft COM and DCOM technologies and the ability of object oriented programming to define functionality of objects by interfaces without specifying the inside implementation. The OPC Foundation has standardized in OPC specifications many of these interfaces for many different purposes. Every specification always contains Custom Interfaces and Automation Interfaces. The Custom Interfaces, which must be always implemented by the OPC server, are mainly for server and client programmers using C++ programming language. The Automation Interfaces, which are optional, are for client programmers developing with higher-level programming tools such as the Visual Basic (VB) and the Visual Basic for Applications (VBA). VBA is used for example in the MS Excel. In both cases, efficiency of code writing and application performance are in contrast. The OPC specifications define only interfaces and their behavior; therefore concrete implementation is up to every server's vendor.

OPC Foundation has created several interface specifications that are meant to be used for different purposes. The OPC server does not need to implement all of these interfaces but when the OPC server is implementing an interface it has to implement all its functionality. The most important specifications that the OPC Foundation has standardized or that are in the standardization process are:

- *OPC Common*
- *OPC Alarms and Events*
- *OPC Batch*
- *OPC Data Access*
- *OPC Data Exchange*
- *OPC Historical Data Access*
- *OPC Security*
- *OPC XML Data Access*

2.6.3 OPC Common

This specification defines interfaces that have to be implemented by all OPC servers, except the OPC Security server. OPC Common specifies functions for setting and querying LocalID, which is an identifier used for defining individual server-client sessions. [OPC Common, 1998].

2.6.4 OPC Alarms and Events

This specification defines interfaces of OPC Event Servers. With help of these interfaces, OPC clients are able to find out what specific events and alarms are supported by individual OPC Event servers and they can register themselves to be notified about their occurrence. Alarms and events can, for example, indicate trespass of the safety limits of device variables or other abnormal situations.

Within the specification an alarm is a special case of a condition that needs an extra attention. A condition is a denominated state of the OPC Event Server or one or more of its objects. An event is a detectable occurrence that is significant to the OPC Event Server, to the device the server represents or to OPC clients. The event does not have a direct object representation as the alarm does. There are three types of events in the OPC Event Servers:

- *Condition-related events* - they are associated with defined conditions.
- *Tracking-related events* - they are not associated with a condition but with interaction between the OPC Event Server and its process related objects. These events can be generated for example by a system operator when he changes the set point of a system variable.
- *Simple events* - they are all the other events than those mentioned above, for example a component failure.

Alarms and events are generated by the OPC Event Server that is represented by COM objects. The OPC Event Server implements the IOPCEventServer interface that provides clients the following functionality:

- Determining the types of events supported by the specific OPC Event Server.
- Entering subscriptions to the individual events.
- Registering the client's call-back interface that is used when the OPC Event Server is shutting down.

The OPC Event Servers can play several roles in the whole information system. The key ones supported by the OPC specification are:

- An OPC Event Server that can detect alarms and events and send them to one or more clients.
- An OPC Event Server that can collect alarms and events from multiple sources (for example by subscribing to another OPC Event Server) and report them to one or more clients.

A picture, which is illustrating the two types of OPC Event Servers mentioned above, is in Figure 2.3. Clients of OPC Event Servers are usually:

- operator stations
- event/alarm logging components
- event/alarm management subsystems

[OPC AE, 2002]

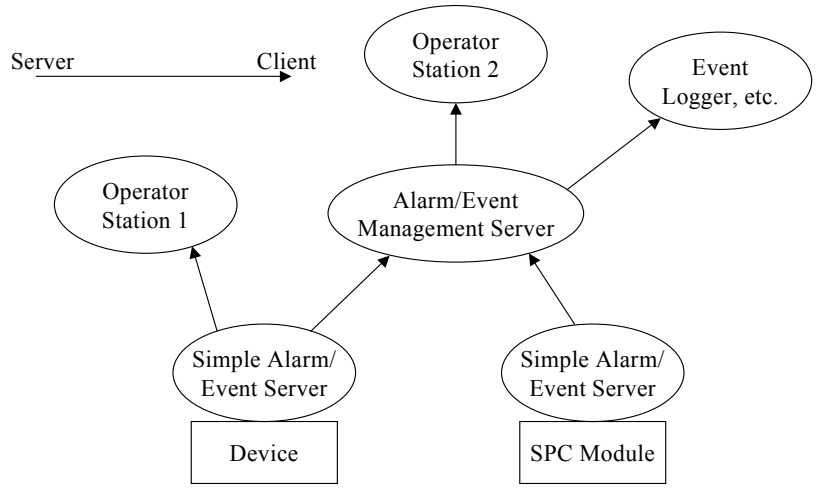

Figure 2.3 Interaction between several OPC Alarm and Event Servers and Clients (adopted from [OPC AE, 2002]).

2.6.5 OPC Batch

This specification provides description of COM objects, interfaces and namespaces used by OPC Batch Servers and their OPC clients. They allow exchanging the following types of data:

- *Current runtime batch information*.
- *Equipment information* necessary for understanding the context of the runtime batch information.
- *Historical records of batch execution*.
- *Master recipe contents*.
- *Batch related events*.

The OPC Batch specification does not specify the solution for batch control problems but it provides tools that enable to deal with these problems. Integration with the OPC Alarms and Events Servers can be done through so called Batch Specific Attributes. These attributes support mapping of events that are generated during a batch processing and thus they enable to OPC Alarm and Event servers to provide these events to OPC clients. [OPC Batch, 2001]

2.6.6 OPC Data Access

The OPC Data Access (DA) specification is the most often implemented one. It defines server and client objects and interfaces that allow the OPC client to connect to the OPC DA server and through it to read and write data from running industrial processes, for example current variable values, control parameters, information about the current state of network connections etc.

The OPC DA specification defines requirements for both the OPC DA server and the OPC DA client. In both cases, there is a specified group of interfaces that need to be implemented. According to the specification, the OPC DA server is represented by these objects:

- *OPCServer object*
- *OPCGroup object*
- *OPCItem object*

The *OPCServer* object maintains information about the OPC Server. It stores the OPCGroup objects and provides tools to clients for managing these objects (creating, deleting, etc.) and it provides means for writing and reading process data that are administered by the OPC Server. The types of available process data depend on the particular server implementation.

OPC Group objects contain information about themselves and they provide tools for creating and organizing OPC Item objects associated with them. After connecting to an OPC Server, a client creates and registers an OPC Group object in the server. The client stores data items that are chosen from all OPC items offered by the server in this OPC Group object. With help of the OPC Group object, the client formulates its interest in the selected items and can manage these items at once. The OPC Group object can be activated and deactivated and it provides a mechanism for the client to subscribe to the set of items so that it can be notified when they change. A notification is sent to the registered client's callback function. The OPC Group object can be of two types, private or public. The private one is only accessible by the client who created it. In contrast, the public one is for sharing by multiple clients.

The *OPCItem* object is an internal object of the OPC Server object and can, for example, represent the data type or the current value of a variable. Every OPCItem object has its ItemID that identifies it in the OPC Server. From a client point of view, the OPCItem object is not accessible directly. Instead, the client has to use the OPC Group object where the item is referenced. The OPCItem object represents more an address of a data source than the data source itself.

Every OPCItem object has its Value, Quality and TimeStamp parameters. OPCItems defined in the OPCServer object are usually structured according to their relation to the process they belong to. An example item structure is in Figure 2.4. In this Figure, there are three OPCItems, the first one with ItemID that is equal to AREA1.REACTOR10.TIC1001.CURRENT_VALUE.

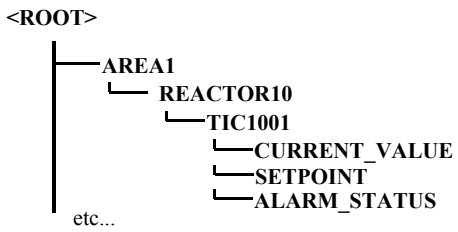

Figure 2.4 Example data structure in the OPC Data Access server (adopted from [OPC DA, 2002]).

The OPC DA specification also defines what interfaces the OPC DA client has to implement. These interfaces contain callback functions used by the OPC DA Server to send notifications about completed operations and other events. [OPC DA, 2002]

2.6.7 OPC Data Exchange

The standardization process of the first version of the OPC Data Exchange (DX) specification was finished in February 2003 but was not accessible for public at the time of writing this thesis. In contrast to OPC DA, where data are transferred vertically among sensors, actuators and controllers at the factory floor and business and monitoring applications at the highest level of a plant, OPC DX supports a horizontal peer-to-peer data exchange and remote server configuration. Functionality that is provided by OPC DX interfaces includes:

- *A server to server communication* - OPC DX defines interfaces that extend OPC DA functionality to allow a direct data transfer between OPC DA Servers. Previously, this had to be done by client applications.
- *A remote server configuration* - OPC DX clients can be used to configure OPC DX devices on a network using standard DCOM and XML based interfaces.
- *Support for web technologies and communication over various networks* - OPC DX supports the newest communication technologies such as Simple Object Access Protocol (SOAP) and takes advantages of the Microsoft .NET framework. OPC DX also serves as a bridge between different network systems, for example Profibus, Fieldbus, Ethernet networks, etc.

[OPC DX Press Release, 2002], [OPC DX Vision, 2002]

2.6.8 OPC Historical Data Access

The OPC Historical Data Access (HDA) specification defines interfaces enabling to retrieve both on-line and off-line data from the OPC Historian server. This server stores measured or by another way gained process data. Data are recorded along with other attributes that allow clients to later determine when data were measured, what they

represent, what their quality is, whether they were changed after measurement, etc. The OPC Historian server also offers so called aggregates. They are operations over measured sets of data, for example minimum and maximum value, variance value and others. What data, data attributes and aggregates an OPC Historian server offers is up to particular server implementation and up to its specific use in the given industrial environment. The OPC HDA interfaces allow a client at runtime to discover what concrete data an OPC HDA server provides and what data operations it supports.

The OPC Historian servers can support various features. This is determined by implementation of optional OPC HDA interfaces. The main two types of servers supported by the OPC HDA specification are the following:

- *Simple Trend data servers.* These servers simply store measured data together with time and quality attributes and provide them to clients in the same format.
- *Complex data compression and analysis servers.* These servers compress data before storing them. They provide aggregates, support for historical data updates and possibility of storing annotations along with historical data.

[OPC HDA, 2001]

2.6.9 OPC Security

The OPC Security specification allows the OPC Servers to implement security issues in a common standardized way. It also provides guidelines for the OPC Clients how to communicate with security aware OPC Servers.

Security is provided with help of two optional interfaces that can be implemented by all types of OPC Servers. The OPC Server providing security has to implement one or both of these interfaces. The IOPCSecurityNT interface is for clients running at the MS Windows NT or later platform. It allows a client to connect to the server and to use NT credentials that are associated with the logon user. Therefore, security issues can be for NT users using this interface transparent and hidden. The IOPCSecurityPrivate interface is for clients using private credentials, which are necessary for example when connecting to OPC servers running on a non MS Windows NT platform. It always requires a user to provide a user ID and a password. The OPC Security specification does not specify what data objects are secured. This is always up to every OPC Server implementation. The OPC Security specification is based on the OPC Security Reference Model, which comes from the MS Windows NT Security Model. [OPC Security, 2000]

2.6.10 OPC XML Data Access

The OPC XML Data Access (XML-DA) specification was not yet finished at the time of writing this thesis and the preliminary version was not accessible for public.

The OPC XML-DA specification defines Web Services that provide clients the same kind of plant floor data as the OPC DA server does. Thanks to the use of the widely spread and platform independent XML format and the SOAP protocol, data can be accessed by clients from wherever in the world where is a connection to the Internet. Comparison between OPC DA and OPC XML-DA is done in Table 2.1.

	OPC XML-DA	**OPC DA**
Based on	Web Services	COM / DCOM
Data transfer	XML, longer data and conversion overhead	Binary, allowing fast data transfer
Inter-platform	XML, SOAP is not platform specific and widely available	DCOM availability is very limited on non-Windows system and the binary data transfer may cause problems
Connection	Designed for non-permanent connections	Permanent connection required

Table 2.1 OPC XML-DA compared with OPC DA (adopted from [Haus, 2003]).

OPC DA has performance advantages over OPC XML-DA but OPC XML-DA is suitable for higher-level applications where flexibility is more important than transfer efficiency. OPC XML-DA supports connection between different operating systems and at very long distances. OPC XML-DA functionality provides only a non-permanent connection and individual functions are independent on each other.

[Haus, 2003]

2.6.11 Advantages

OPC allows manufacturers to create only one set of software components that can be used by all client applications. Client developers can use the support-component-writing languages and they do not need to rewrite their code when devices are changed or new ones are created. OPC provides efficient communication and high level of functionality. It allows Plug & Play functionality in process automation. It reduces training, custom development and maintenance costs.

[Karhela, 1999], [OPC Overview, 1998] and [OPC Overview, 2002].

3 Agent technology and information agents

3.1 Agent technology

Nowadays, agent technology represents a fast-growing research discipline with high expectations. The bases of agent technology lay in artificial intelligence research and distributed computing. Agents in many ways resemble human behavior and hence, other disciplines such as philosophy, logic, economics, social sciences, biology, language study and others play also important role in agent technology. Computer science here constitutes the role of a building tool that implements the ideas given from the disciplines mentioned above. In this literature review chapter, a definition of an agent is given, the reasons why agent technology was founded is discussed as well as what agent architecture is and how it is nowadays implemented. Farther, there are mentioned advantages and disadvantages of agent approach, where agent technology is used and what its future is. For more detailed information about agent technology, a reader is advised to follow the references at the end of this thesis.

3.1.1 Agent definition

Nearly every article dealing with agents provides its own agent definition. Some definitions are very abstract ones and some of them define agents according to the area where they are studied. A general definition can be taken, for example, from [Ferber, 1999]:

"*An agent is a physical or virtual entity*

a) *which is capable of acting in an environment,*
b) *which can communicate directly with other agents,*
c) *which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize),*
d) *which possesses resources of its own,*
e) *which is capable of perceiving its environment (but to a limited extent),*
f) *which has only a partial representation of this environment (and perhaps none at all),*
g) *which possesses skills and can offer services,*
h) *which may be able to reproduce itself,*
i) *whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and the communications it receives.*"

The definition above addresses all types of agents. In this thesis, only software agents are treated if not specified differently. Another definition, this time of the software agent, taken from [AgentBuilder, 2003] says that agents are the next technology development step in computer science:

"*An agent is simply another kind of software abstraction, an abstraction in the same way that methods, functions, and objects are software abstractions. An object is a high-level abstraction that describes methods and attributes of a software component. An agent, however, is an extremely high-level software abstraction which provides a convenient and powerful way to describe a complex software entity. Rather than being defined in terms of methods and attributes, an agent is defined in terms of its behavior. This is important because programming an agent-based system is primarily a matter of specifying agent behavior instead of identifying classes, methods and attributes. It is much easier and more natural to specify behavior than to write code.*
There is a minimum set of common features that typify a software agent. A software agent is:

- *autonomous; the agent is capable of operating as a standalone process and performing actions without user intervention.*
- *communicative; it communicates with the user, other software agents, or other software processes.*
- *perceptive; it is able to perceive and respond to changes in its environment.*"

3.1.2 Agent characteristics

For the current developing methods, it is very difficult to model complex distributed systems, to define their structure and the relation among their individual parts. Control unit implementation of such complex systems, which are capable to respond correctly to whatever situation that happens in the whole system, is almost unfeasible. Designers cannot treat all the different states a system can have because there are simply too many of them (an interaction in unpredictable time among unpredictable components is impossible to analyze at the design time). Thus, if any technology is able to tackle the problems specified above, then it can be said that such technology can substantially enhance the current approaches for dealing with large systems.

Agent technology is proposed to be a tool for designing and implementing large complex systems. Agents are supposed to replace sophisticated monolithic software systems. Individual agents are less intelligent than the central control unit and they are mainly focused on specific problems. On the other hand, agents are able to intelligently communicate with each other and thus they are able to achieve the same functionality as the central system has. Moreover, agents are supposed to create more transparent and more robust systems and be better scalable and better controllable than the central systems.

Because agents are autonomous and they know how to behave by themselves, the overall control complexity is reduced by being redistributed into individual agents and also coupling among system components becomes of lower degree. Therefore, the agent approach is suitable for decomposition of a task into smaller ones. Dealing with

smaller tasks is easier for programmers and it is less prone to create mistakes in program code.

The ability to decompose a system into smaller parts makes agent technology also suitable as a designing tool of large systems. With agent technology, a higher level of abstraction is achieved.

The possibility to create intelligent interrelationship among individual agents is another advantage of agent technology. For example, the agents sharing the same objectives can be grouped together and cooperate with each other. According to actual needs, agents can form suitable groups dynamically to cope with the actual situation more effectively. A group of agents can have an agent representative who stands in for these agents and serves as their coordinator. By grouping agents together, an arbitrary level of agent hierarchy can be achieved. In case of a failure of one of the agents, the other agents from the group look for its substitution by negotiating among each other.

Agents enhance currently used static data objects by making them active, Figure 3.1. As active objects, agents are a natural development step in programming technology. The world around us is full of objects that actively behave in the surrounding

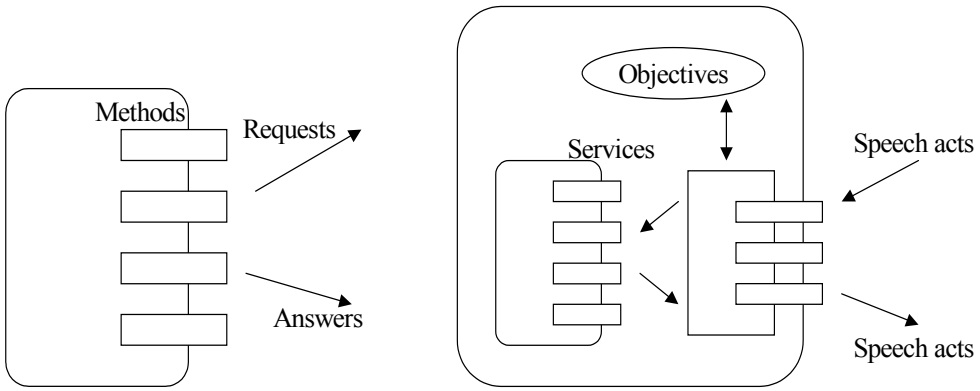

Figure 3.1 An object responds directly to request corresponding to its methods, while an agent encapsulates its skills (or services) through supplementary mechanisms, which 'filter' external communications and manage dialogues. Agents are also driven by personal objectives (or tendencies) (adopted from [Ferber, 1999]).

environment and communicate with it and with each other. Static objects have minimal support for defining and managing organizational relationship. Also, they do not encapsulate an agent choice. Static objects perform actions according to the methods that are invoked without any reasoning about it. In contrast, every agent has its own goals to achieve and all its behavior is devoted to meet these goals. Static objects, unlike agents, do not exhibit social behavior. According to [Jennings, 1999]:

"*Agents have control both over their internal state and over their own behavior (one of things that distinguish agent from objects, although objects encapsulate state and behavior, more accurately behavior realization, they fail to encapsulate behavior activation or action choice. Thus, any object can invoke any publicly accessible method on any other object at any time. Once the method is invoked, the corresponding actions are performed. In this sense, objects are totally obedient to one another and don't have autonomy over their choice of action)*".

Agents can communicate, negotiate and coordinate each other. The agent can disagree and refuse to perform an action, which it was asked to accomplish, or the agent can propose a modified or another else action instead of the original one. This decision making at run time makes engineering of complex systems much easier because designer does not need to analyze all potential system states at design time. [Jennings, 1999], [Maturana et al., 2002].

According to [Young, 2001], the agent approach is better than the currently used ones because it brings:

- *Higher productivity* - for instance agents are able to accomplish more sophisticated tasks just by themselves.
- *Distributed computing* - agents behave autonomously.
- *Economical savings* - single computer with large computing power is replaced by many any cheaper ones.
- *Less network traffic* - agents can group similar data and queries together and send them at once or they can make preprocessing of data before sending them and thus reducing the amount of data.

Agent technology has many advantages over the currently used methods but on the other hand, it brings a number of new problems. These problems are caused by demanding agent communication and by agent's social behavior.

A group of agents is more than a summary of individual agents and therefore an agent system cannot be decomposed and still having the same properties as the whole system. At run time, an agent system can produce an unpredictable agent-collective behavior as a result of interactions among many agents at the same time. That is the reason, why it is very difficult and almost impossible to debug agent code in an old fashioned way. Instead, new approaches need to be introduced. Social science starts to play an important role in studying of agent systems.

Another problem is security and how to introduce trustworthy measures for agents. Agents need to recognize to who they can trust and thus where to send sensitive data. A difficulty is also the lack of standards for agent communication. Agents need to share the same communication language to be able to understand each other.

3.1.3 Agent architecture

Agents are actively communicating with their environment and with each other either directly or through the environment, see Figure 3.2. Agents can share their

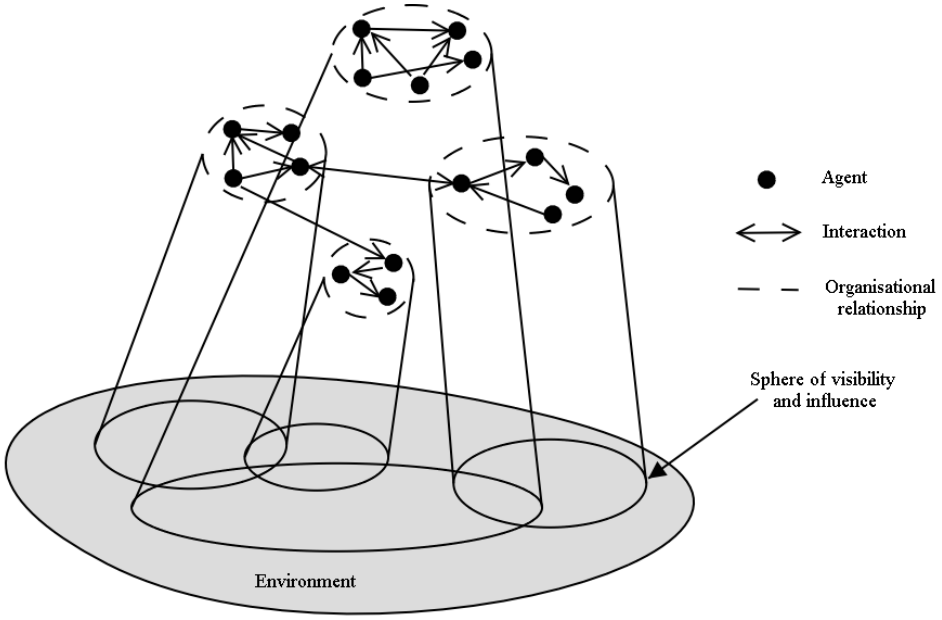

Figure 3.2 Agent hierarchy and environment (adopted from [Jennings, 1999]).

objectives within a group of agents. According to different tasks, agents form different relational structures. The most common are:

- *Centralized hierarchy* - one agent behaves as a super agent and coordinates other agents.
- *Proper hierarchical form* - as in centralized hierarchy plus there are more super agents, that are again coordinated by "a super-super agent" etc.
- *Modified hierarchical form* - as in the proper hierarchical form plus the super agents can communicate with each other - so called horizontal communication.
- *Heterarchical form* - as in centralized hierarchy plus there are more super agents that can communicate with each other as in the modified hierarchical form
- *No hierarchy* - agents are not grouped in any hierarchy

Individual hierarchical structures have their own advantages and disadvantages and are suitable for different environment. For example, stock market can be represented by agent sellers and buyers with free relations and no hierarchy. On the other hand, the industrial process requires more stable architecture such as centralized hierarchy.

Agents that communicate with each other and affect each other form the Multi-Agent System (MAS). To facilitate agent implementation, number of diverse MAS has been created. Individual implementations are called agent platforms and they provide

both agent-programming tools and agent running environment. With help of MAS, programmers can more focus on implementation of individual features of agent behavior than on how to implement that behavior. Agent platforms also provide mechanisms for agent communication and agent registration. The tools offered by a platform depend on the purpose for what the platform was created. One of the possible subdivisions of agent platforms is in Figure 3.3. There exist already a large number of agent platforms in the world. This gives programmers the possibility to choose a platform that best fits to their needs. High platform diversity also means that many different communication techniques are used. This can be a problem when communication among different agent platforms is required. This is the reason why great attention is paid to establishing widely acceptable communication standards. [Pirttioja, 2001], [FIPA, 2002], [Poslad et al., 2001], [Mangina, 2002].

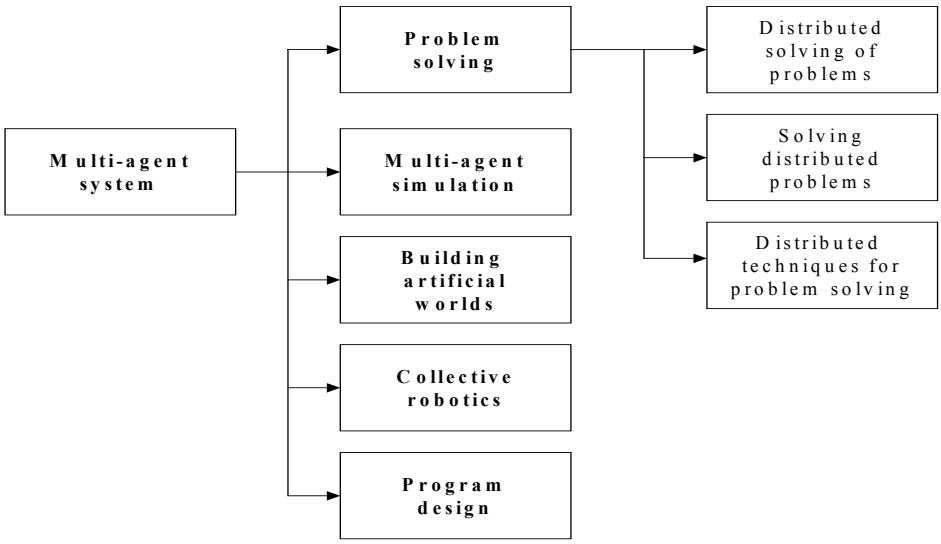

Figure 3.3 A classification of the various types of application for multi-agent systems (adopted from [Ferber, 1999]).

Agents sense different parts of their environment and hence they possess different information. To exchange information, agents communicate among each other. Communication can be done through exchanging messages among agents or by observing other agents' behavior. An agent message consists of a schema and the words that agents understand. The same words can have different meanings in different communication contexts. The meanings of words for a specific domain are determined by ontologies. According to [McEntire, 1999] an ontology is: "*Specification of a conceptualization that is designed for reuse across multiple applications. By conceptualization, we mean a set of concepts, relations, objects, and constraints that define some domain of interest.*"
[Bailin and Truszkowski, 2001], [Gomez et al., 2001], [Labrou et al., 1999], [FIPA, 2002g].

Several models of agent architectures have been proposed as a result of studies of an agent-based computing during the last ten years.

First, agents can be purely reactive (behavioural). These agents have a direct connection from their receptors to their actuators. Therefore they behave in a reflexive way. The response of these agents to a stimulus is very fast but the complexity of their behavior is limited. Into this group belong for example agents using the subsumption architecture of Brooks.

Second, agents are deliberative. They plan their actions. In this category are belief-desire-intention (BDI) agents. They have a model of their surrounding environment and based on this model and on information from sensors they can produce complex behavior. The disadvantage of these agents is that their reactivity is slower comparing to reactive agents and that their model has to be kept updated with changes in the surrounding environment. Nevertheless, these agents are prevalent.

The third type of architecture is based on the previous two types and it is called hybrid architecture. It takes advantages of fast responsibility of reactive architecture and prediction capabilities of deliberative architecture. Architecture of both deliberative and reactive agent can be seen in Figure 3.4. [Luck et al., 2002].

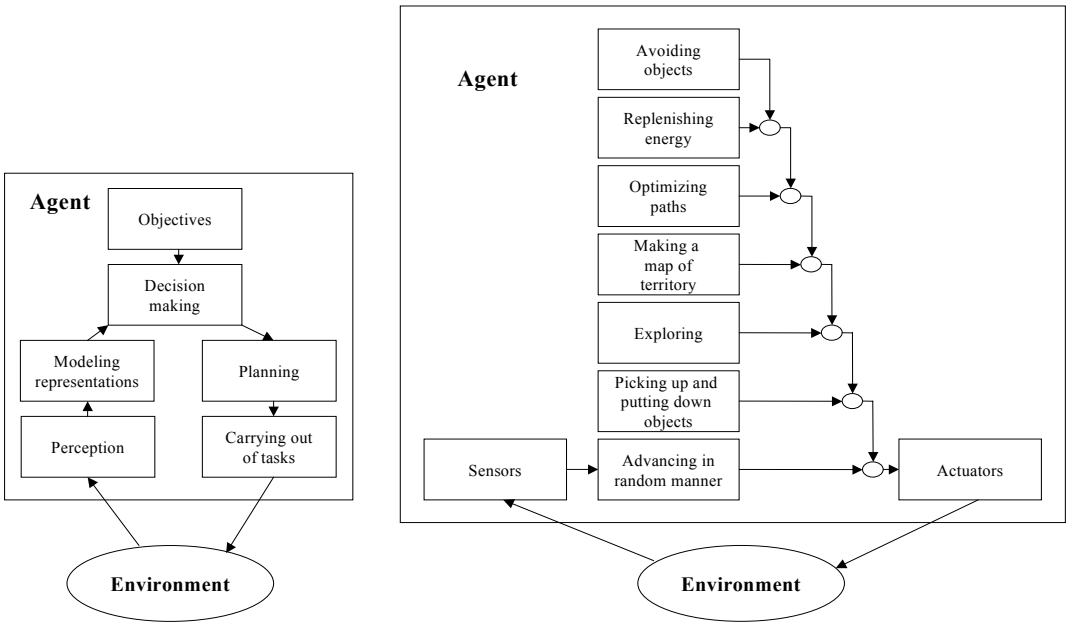

Figure 3.4 Representation of deliberative agent with horizontal modular architecture (left) and reactive agent with subsumption architecture (right) (adopted from [Feber, 1999]).

3.1.4 Agent implementation and related standards

Agent technology is not being developed to compete with any current programming technology but instead, it takes advantage of them. It combines

programming technologies with knowledge from other non technical areas, especially from the human studies. Agents and agent platforms can be implemented in almost any programming language. Nevertheless, only programming languages that are widely spread and have great support, for example in good development tools and many code libraries, are used. Currently, the most used languages for agent programming are C++ and Java.

The essential part of every agent platform is a communication module that is responsible for communication with other agent platforms. To achieve compatibility among platforms, technologies supporting mutual communication, public platform independent protocols, data formats and public standards are used. It is important to mention the FIPA (Foundation for Intelligent Physical Agents) organization. It establishes, among others, standards for agent communication languages (FIPA-SL, FIPA-RDF, FIPA-KIF) and standards for communication protocols (FIPA-Request communication protocol etc.).

The most used languages for exchanging knowledge among agents are Ontolingua, CycL, Ontology Markup Language (OML)/ Conceptual Knowledge Markup Language (CKML), Object-Process Methodology (OPM), Extensible Markup Language (XML) / Resource Description Framework (RDF) and Ontology Definition Language (ODL). Ontolingua is a language based on KIF and it is the ontology-building language used by the Ontolingua Server. CycL is a formal language for representing assertions in Cyc - the world biggest knowledge base. OML/CKML languages express ontologies in the XML based format. OPM is an object-oriented data model used to describe single and multi-database schemas and queries. RDF is designed to encode metadata concerning web documents. ODL is a language for a common representation of objects in object-oriented databases and programming languages.

An ontology and agent knowledge can be either hard-coded into agent code, it can be acquired from other agents or a combination of both of these. The first approach is relatively easy to implement. Agents possess code, which expresses instructions how to behave in situations represented by specified conditions. The problem is how to behave in unknown situations. In the second approach, agents are more flexible to cope with situations unknown at design time. Here, the problem is the implementation of the learning process.

Agent communication takes advantage of tools created for distributed network computing. The most known of them are the Common Object Request Broker Architecture (CORBA), Web Services and JINI. CORBA and Web Services are architectures that enable remote object creation and remote methods invocation. JINI provides a simple Java based mechanism that enables devices to connect together and to provide services to each other. A relatively new protocol is the XML based Simple Object Access Protocol (SOAP) that gains increasing popularity coming from its

platform independence and the possibility to be sent through the widely spread and for most networks transparent HTTP protocol. SOAP provides a simple way for exchanging structured data in a distributed environment and it is a fundamental element of the Web Services. For advertising agent services and agents themselves to other agents in a network, Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL) and Web Services Conversation Language (WSCL) have been developed. UDDI is a platform-independent open framework for describing and discovering services. WSDL is a grammar for describing network services and WSCL enables definition of abstract interfaces for the Web Services. [Labrou et al., 1999], [FIPA, 2002c], [FIPA, 2002d], [FIPA, 2002e].

3.1.5 Areas of applications

Agent technology is still at the beginning of its development despite of effort and attention that was given to it. Nowadays, almost every technical university has its own research team working on this topic. There are several conferences in the world every year (HoloMAS, CIA and others) and new collaborations between companies and universities are established. Agent technology is studied how to be used in different industrial domains from economics to telecommunication, see [Hayzelden and Bourne , 2001].

The agent-based systems can be divided into three broad categories according to their use:

- *Assistant agents* - they act as secretaries, they collect information, execute transactions on behalf of a human principal or otherwise else help to their master.
- *Multi-agent simulation systems* - agents model and simulate real world domains. They typically simulate systems composed from a high number of components of different types where the system level properties cannot be obtained from individual component's properties, for instance economy systems or human and animal societies.
- *Multi-agent decision systems* - agents make mutual decisions based for instance on the auction mechanism.

Currently, use of agent technology is studied for example in: manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, human capital management, skills management, (mobile) workforce management, defense, entertainment, medical care and others.

Examples of real applications of agent technology can be found in [Parunak, 1998]. For instance, agents coordinate discrete-manufacturing process in the Autonomous Agents for Rock Island Arsenal project or agents regulate the individual room temperature in a building in the Market-Based Climate Control project.

It can be expected that agent technology will play more and more significant role in many different areas in the next few years. To be able to fully use the power of agent technology, it is still needed to resolve many problems associated especially with agent communication and knowledge storing and sharing. Predictions of the next agent technology development-steps can be seen in Figure 3.5. It is said that the main attention

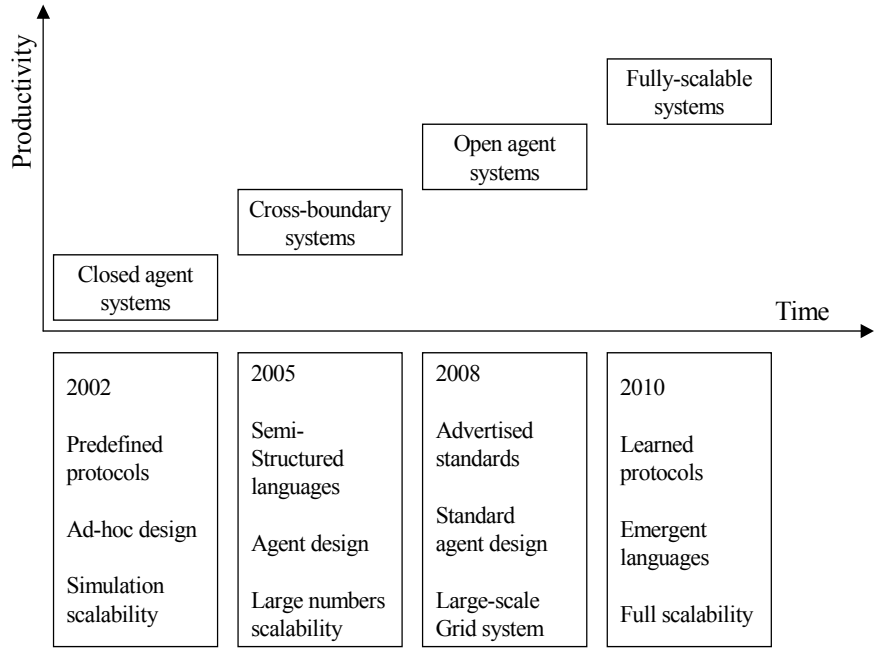

Figure 3.5 Development of agent technology research (adopted from [Luck et al., 2002]).

is going to be paid to establishing the communication and the knowledge managing standards. At the end of this process, agents should be able to communicate with other agents they have never met before. They should be able to exchange new information among each other, to understand this information and mainly to make a reasonable decisions and actions based on obtained information. Applications, where agent technology will play important role in the future, include:

- *Ambient intelligence* - programs possess intelligent interfaces for human users or other devices and they generally behave intelligently when communicating and processing data.
- *Grid computing* - it is related to intelligent task decomposition and intelligent network distribution of its subtasks and thus enabling efficient use of computing resources.
- *Electronic business* - agent-based automatic business contracts, business negotiations and transaction performing.
- *Semantic web* - agents intelligently process user requests and cooperate with each other in searching for information to get the most suitable information about requested topic.

- *Bioinformatics and computational biology* - agents support intelligent data searching through the fast growing medical databases.
- Others like monitoring and control, resource management, space and military applications, etc.

[Luck et al., 2002].

3.2 Agent technology in information access

It is common that companies during the time of their existence gather abundance of information relevant to their work activity. Companies constantly need to access this information and work with it. Their profit is often dependent on the time with which relevant information can be obtained. Information is usually stored in heterogeneous sources in a semi-structured form with the lack of formal semantics. This causes serious problems to the current information technologies. First, searching for right information is difficult in such data sources. Second, maintaining existing information with semi-structured and distributed nature is time demanding. Another problem is that information sources are more often used by users with minimal computer knowledge who do not know how to use information technologies. Nevertheless, the current situation is that most from the information processing, including searching, browsing, accessing, interpreting, maintaining, and generating, is devoted to human users. [Fensel, 2002].

3.2.1 Information agent

The agent approach brings new possibilities and advantages into information access by introducing information agents. From general point of view, an information agent is the agent that was introduced in the previous sections and has all there mentioned properties. The information agent specializes itself on information providing in a more intelligent and autonomous way than the traditional information access approaches. It means that the information agent is able to analyze given data requests and, based on them, effectively search and provide data. The power of information agents comes from their cooperation and the ability to semantically process and understand given data requests. Based on [Barbuceanu and Fox, 1994], information agents provide support for sharing of stored information, deductive capabilities (inferring new information from existing information), automatic, content-based routing and distribution of information to the agents that need it, automatic retrieval, processing and integration of information, checking and maintaining various forms of consistency of the information and performing information access control functions (specifying who has or has not the right to see and change available information).

A general definition from [Babbage simmel, 2003] states that the information agent is an event-driven delivery mechanism. This definition is very wide. It does not specify the subject of the delivering process. In this thesis, the information agent is regarded more according to [Klusch, 2003]: "*an information agent is a computational software entity that has access to one or multiple, heterogeneous and distributed*

information sources, pro-actively searches for and maintains relevant information on behalf of its human users or other agents preferably just-in-time."

3.2.2 Information agent architecture

In [Decker, 1997], importance of design-template architecture is mentioned for each class of agents because it makes implementation of agents easier for programmers, it leads to better understanding of the agent characteristics and it supports interactions among agents developed by different agent designers. A set of reusable behaviors (particular approaches for a goal accomplishing) that the information agent needs to implement, is proposed. These basic behaviors are:

- *Advertising behavior* - an information agent exists in an environment and has to provide a description of information and ontologies it provides by registering to a matchmaker or a broker who act also as the information agents but they provide only yellow pages information.
- *Message polling behavior* - an agent continually asks for new incoming messages to process.
- *Information monitoring behavior* - an agent monitors information and when a given condition is true it starts a transition of selected data.
- *Query answering behavior* - the ability to answer simple information requests.
- *Cloning behavior* - in overloaded working conditions an agent can clone itself to divide its burden to be able to satisfy other incoming requests in reasonable time.

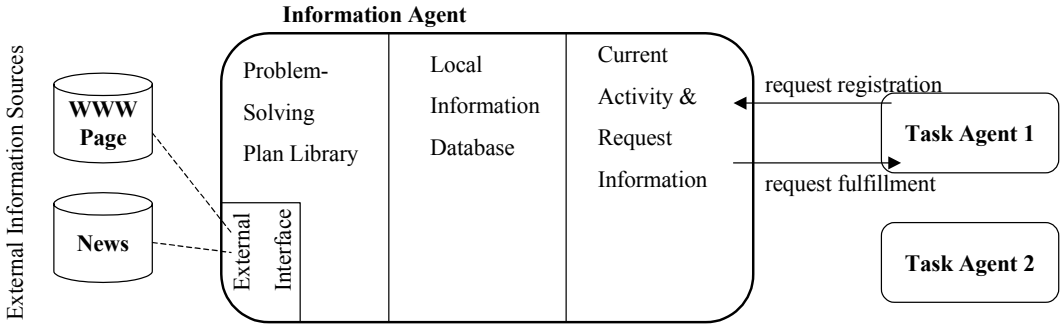

Figure 3.6 An information agent's knowledge structures and environment (adopted from [Decker, 1997]).

The proposed information agent architecture is in Figure 3.6. The information agent is divided into four modules. The Current Activity & Request Information module is responsible for communication with other agents, monitoring agent environment and fulfilling information requests through the Local Information Database module. This module stores information from the previous data requests and, in the case that it does not have requested information, it uses the Problem Solving Plan Library module to retrieve this information. The information format in the Local Information Database module does not depend on the source from which the information was received. The

Local Information Database module has three important functions. It is able to group multiple related queries and thus to limit access to external data sources and decrease the network load. Then, it serves as a buffer in the case of unexpected problems with external sources. Third, it can provide historical and statistical data. The External Interface module is responsible for accessing heterogeneous databases.

3.2.3 Current use of agent-based information access

Probably the most studied application of agent technology in information access is the use for searching data through intranets and the Internet. Semantic webs and information agents are their examples. Semantic webs can be understood as data stores keeping data in structured and semantically described forms. This allows to create information agents that are capable of intelligent data searching. The current information access research in this area is aimed on creation of means and standards for semantic data description of information and on tools providing more efficient access to existing information.

Another class of applications covers programs that generate new information. They create new data on behalf of current needs or given requests. For example data mining applications working over large amount of distributed data and carrying out analysis. Another example are monitoring applications collecting and evaluating information from given systems.

According to the current research, agents should be capable of adaptive information retrieval; reasoning with imperfect information; advanced, personalized 3-d visualizations of information spaces; cooperation in real-time and open environments; collaboration in peer-to-peer networks; agent-based distributed ontology learning and many others in the near future. [Klusch, 2003].

4 Concept of information agents in process automation

The goal of this chapter is to combine the ideas and the principles from the previous chapters and to give a proposal of how agent technology could be used, designed and implemented in information access in process automation.

4.1 Introduction

Agent technology is suitable for building distributed systems. It is an appropriate tool for designing dynamic systems where parameters and structure change at runtime. Currently, information agents are mainly used in the Internet. Their study and their application in process automation are still at the very beginning and actually no such a paper was known to the author at time of writing this thesis. Traditionally, areas, where great emphasis is given to safety and reliability, are rather conservative to new technologies.

In process automation, agents are proposed to represent individual sub-processes and/or devices of the controlled systems. Information agents communicate and cooperate with each other. They read information from a system, calculate information from measured data or they query information from other information agents. Information agents should allow a user or another agent to make a subscription for being notified about selected events. Information agents should also allow changing of the system settings but controlling the system should be left to a controller that does nothing than controlling and therefore it is faster and more reliable.

4.2 Motivation

Information access in process automation is being implemented in many different ways. Usually the biggest leading companies in this domain use their own solutions. On one hand, their own approaches allow them to tailor data communication optimally to their own developed products, but on the other hand, communication between devices from different manufactures can be very difficult to establish.

Communication among active devices on a plant floor is implemented with help of industrial networks such as Profibus, Fieldbus, CAN and many others. This means, that all used devices in one network have to support given communication protocols and rules and therefore, designers of automation systems have limited choice when searching for suitable components.

Communication between a process system and high-level business and control applications uses server-client approach. The server application has access to plant data using a specific industrial network and it provides this data to other client applications using communication standards such as DDE or OPC for example. DDE clients have to know in advance a DDE server name, item names and names of the commands supported

by individual DDE servers. The DDE client applications are thus a DDE server dependent and have to be configured manually, which requires depth knowledge of a specific DDE server's capabilities. An OPC client can at runtime iterate automatically through data and services offered by an OPC server but what individual data represent and what is their relation to the process system is up to a user to decide.

In both cases where DDE or OPC are used, client applications have to be preconfigured in advance to be able to display and process right data in the right way. Client applications have to possess knowledge about data they use and therefore, they are tightly coupled with individual data servers, see Figure 4.1. For every process system, a user needs a special client that understands data provided by the system. The current approaches specify how to get data but not what that data represent. The objective of this chapter is to show whether agent technology is able to cope with problems specified above.

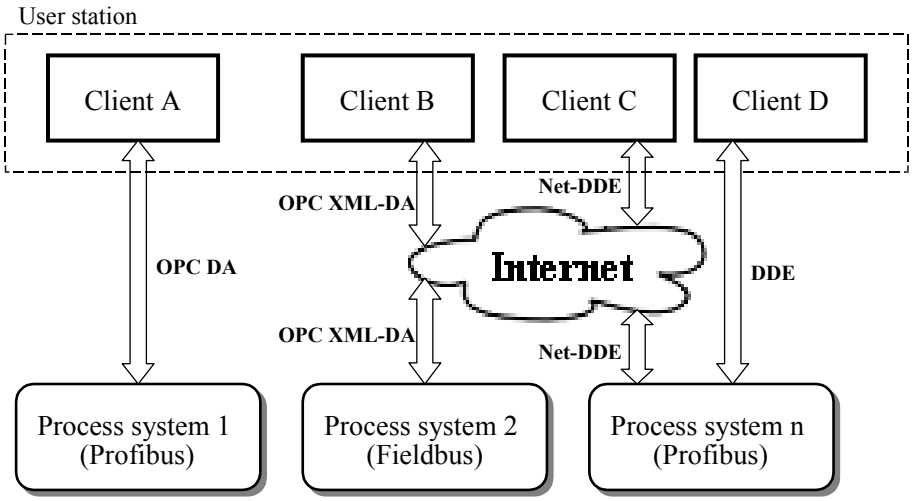

Figure 4.1 Legacy clients tightly coupled to physical systems.

4.3 Analysis of the concept

Information retrieving from process automation and from the Internet is rather different. At the first place, it is safety and efficient running in process automation and therefore, information agents need to be designed in this sense. In contrast to the Internet, an automated system is a limited source of information regarding both the diversity of data and the amount of data. On one hand, information agents in process automation do not need to learn user's behavior or user's personal interests but on the other hand, agents need to monitor and adjust themselves to a process automation system's characteristics. They can learn from the previous correct and incorrect system behavior to anticipate and inform about significant events in the future.

Queries that information agents could be capable to answer are for examples:

- What measurements do you provide?
- What devices are in the system?
- What is the relation between action f and value of a variable x?
- What is the unit of a variable x?
- What are statistical data of a variable x?
- Send me a variable x whenever a condition y is true?
- Change the condition y to a condition z?
- What is the percentage of faultless running of a device x?
- What is the current state of a device x?

4.3.1 Architecture of the concept

From the agent point of view, the information system in process automation is regarded as a multi agent system where individual agents are responsible for different parts of the process system. Such a concept can be implemented in three ways.

First, the automated system is constructed from passive sensors and actuators. Agent technology is implemented by agents running in an agent platform and communication with the passive devices is done through I/O modules of a process

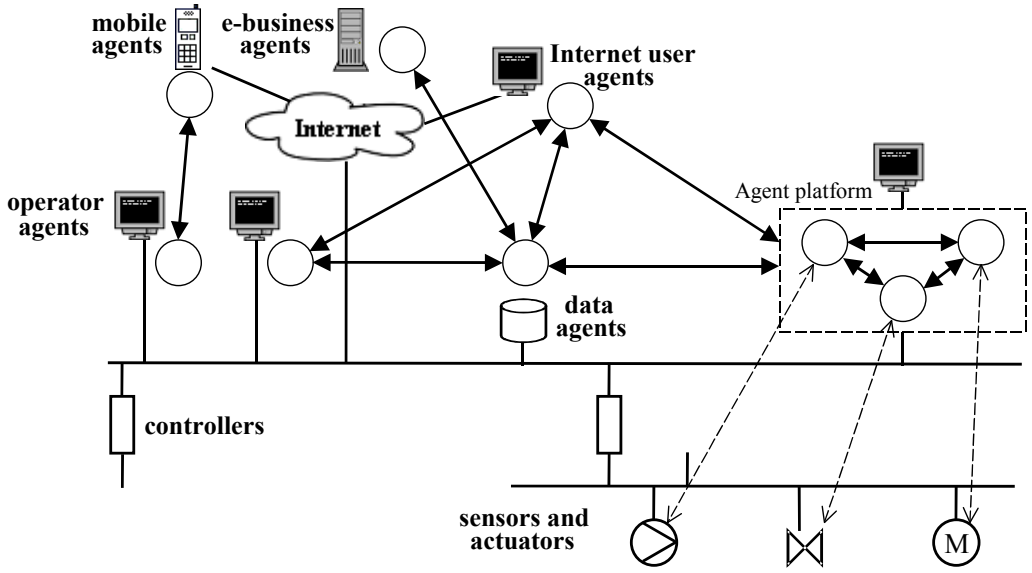

Figure 4.2 Agent augmented process automation system.

controller, see Figure 4.2. This concept refers to an agent augmented process automation system that was proposed and it is in research at the Helsinki University of Technology (HUT) at the Automation Technology Laboratory, see [Appelqvist et al., 2002], [Seilonen et al., 2002c] and [Pirttioja, 2002]. The general architecture of the agent augmented process automation system is in Figure 4.3. Agents create an abstract layer above the process automation system and provide intelligent interface to the system. This

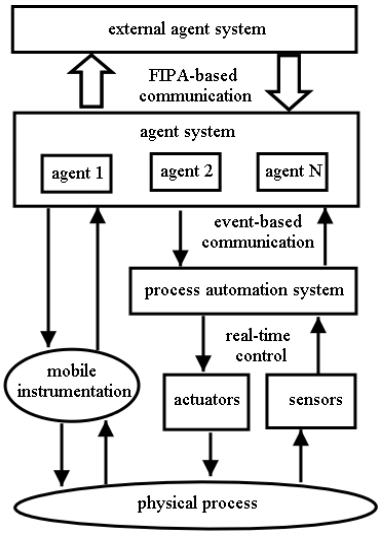

Figure 4.3 Architecture of agent-augmented process automation system.

concept is necessary for the current non-intelligent devices and requires an agent platform where software agents, corresponding to individual devices or subsystems, are running. An example of implementation of such an agent platform is the FIPA for Process Control (FIPAPC) developed at HUT. This platform is based on the FIPA-OS agent platform, which is extended by functionality to access data from an automation system through OPC technology. The agent augmented process automation system requires that agents are configured manually by the FIPAPC programmers according to technical characteristics of the system.

Second, the process automation system is composed from intelligent devices. These devices possess all technical information about themselves and behave as agents, see Figure 4.4. Advantages of this concept consist in the unified solution which means that a device and its software agent constitute one unit. This approach allows use of the agent devices in a Plug-n-Play manner and it cancels need for an agent platform. Nowadays with the progress in the electronic industry, so called embedded systems are becoming more and more popular in process automation. They are dedicated electronic devices that contain a powerful computational unit and scalable and configurable operating software. Embedded systems are suitable candidates for agent devices. They 'just' need to implement agent features. Agents themselves can be regarded as one type of embedded software systems. Currently, the disadvantage of the second approach is higher complexity of intelligent devices and thus their higher price.

Third, both approaches mentioned above are combined in the process automation system. The passive devices are encapsulated by the agent layer and thus they are incorporated into the agent system consisting of intelligent devices implementing agent features. In all cases, information agents need to implement the same communication standards to understand to each other.

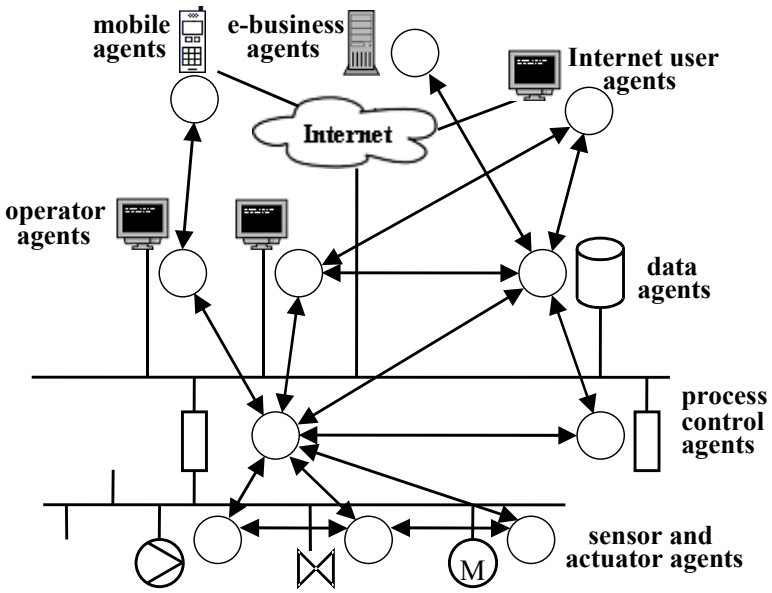

Figure 4.4 Intelligent devices with agent behavior in process automation.

4.3.2 Ontology

When implementing information agents in process automation, one approach is to program agents to have whole knowledge about all devices in the system and to be able to understand to all possible users' information-requests. This approach requires a lot of programming and it is not flexible to changes in the system. In the second approach, agent architecture is ontology independent (it does not contain hard coded meaning of terms used in agent communication) and it possesses mechanism for working with new ontologies from various domains. In this case, ontologies are downloadable by agents when necessary from other agents or from ontology depositories. To enable the mechanism of manipulation with ontologies, an ontology manipulation language and ontology protocols have to be specified.

In both approaches, ontologies should be separated according to the domains they represent. Relations between ontologies have to be specified so information agents have a mechanism to find ontologies they need for dealing with individual requests. One possible approach is to store ontologies in a hierarchical structure, see Figure 4.5. The root ontology, which is common to all other ontologies, is so called information meta-ontology. This ontology represents a general base of terms used to obtain information about what data are provided by an information agent representing particular device or a subsystem. It can also specify terms that are used to get information common to all information agents. The information meta-ontology has to be a domain independent ontology and has to be understood by all information agents. Examples of queries that the information meta-ontology supports can be the following:

- What measurements do you provide and in what ontology they are specified?
- What actions do you provide and in what ontology they are specified?

- What is the description of the device you are responsible for?
- What is the device current state?
- How long is the device running?
- When was the last device's maintenance-check?

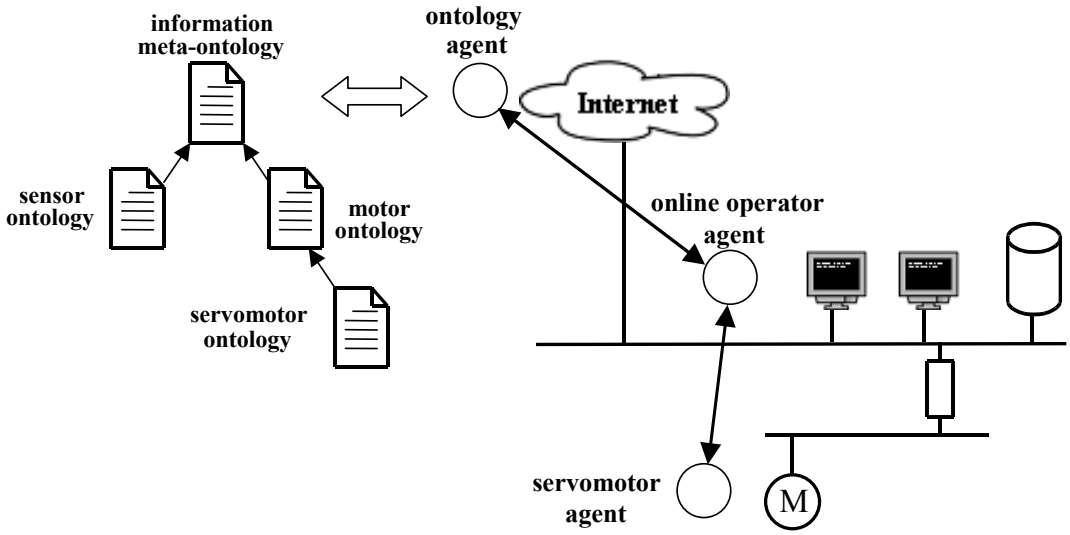

Figure 4.5 Ontology retrieving from an ontology repository.

Other ontologies in the ontology hierarchy are already domain dependant. They define terms related to specific processes, devices or a group of devices. The further in the hierarchy tree ontology is, the more domain specific terms it defines. In the case that two similar ontologies are defined, ontology bridges matching similar terms in both ontologies are proposed to be used. Hierarchical approach for organizing ontologies allows different manufactures to incorporate their own ontologies into the public ontology tree at the place that corresponds to their domain of use. This enables agents to search and to use the required ontologies by specific communication process.

A typical scenario, where ontologies can be exchanged, is in Figure 4.5. An online operator wants to get information from a servomotor. It asks an information agent representing the servomotor what type of ontology is needed to understand servomotor information. Because the online operator agent does not possess the required ontology, it asks an ontology agent, which serves as an ontology repository, to provide the server motor ontology. Once the online operator agent has the required ontology, it can start communication with the servomotor information agent.

4.3.3 Agent communication language

The agent communication language can be regarded as a mathematical model of the simplified human language. In contrast to object method calls, the agent language is not limited as for the number of methods and types of methods' parameters. Its scope of use is being extended with knowledge that agents acquire. The agent communication language should be able to express information efficiently, unambiguously and in well-

supported format. A suitable candidate could be derived from the FIPA-SL language and the XML format which would combine the expressive power of the FIPA-SL with the well supported and widely recognized XML format. A necessary presumption for interoperability among various agents is a standard of such a language.

4.3.4 Communication protocols

In a simple information system, where information agents only need to request specific information, action or create a subscription to an event, three communication protocols are actually needed. They are query, request and subscription protocols. Examples of these protocols can be the corresponding FIPA protocols, see [FIPA, 2002]. Other protocols, such as auction protocols, are more suitable for systems with higher freedom in relation among agents, for example as it is in the market systems.

Another class of protocols, that information agents will need, is for exchanging ontologies or information about ontologies and protocols for exchanging the protocols themselves. The former ones enable to automatically extend process systems in a Plug-And-Play manner. For example, when a new device is added into a process system or some device is replaced by another one which is using an ontology that is new for the current information agents, the information agents need to use the ontology protocols to get the required ontology. Protocols, that enable exchanging of description about protocols and exchanging the protocols themselves, enable automatic extension to agent communication mechanisms. An example can be very close to the previous one. An information agent can need to go through an iteration process with other agents to deduce required information. But the problem is that not all agents know the required iterative communication protocol. The initiator agent thus provides the iterative protocol to other agents or instructs other agents to get the required protocol from a protocol repository. Finally, agents start to use the right protocol. Well-defined standards and mechanisms are again needed to ensure such functionality.

4.3.5 Agent architecture

Internal architecture of information agents can be realized in many ways. In all cases, it should enable to easy extend agent behavior or to change the current one. Therefore, an agent should be built from independent modules that can be updated at runtime and configured through the configuration files without need for recompilation.

Information agents can be implemented as special agents whose only tasks are information retrieving and providing. Another way is to create an information module that can be added to existing agents. In process automation, emphasis is given to short time responses and to reliability of agents. Therefore, individual agents should not be too complex and their functionality should be devoted only to one domain of tasks.

Very general information agent architecture is proposed in Figure 4.6. This architecture is based on the BDI agent model and it enables an agent to process different messages and to remember acquired information. A description of individual parts of this architecture is given from the fundamental point of view.

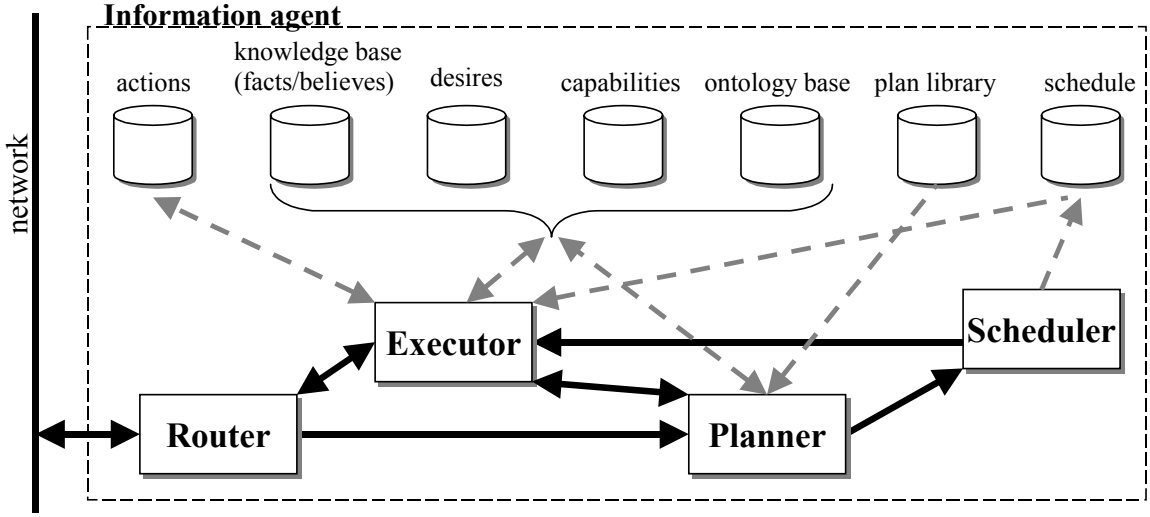

Figure 4.6 Information agent architecture based on RETSINA and [Finin et al., 1994] agent architectures.

The Router module is responsible for outgoing and incoming message routing. In the case, that the outgoing message does not contain a recipient's address, the Router module uses matchmaking or brokering services of a facilitator agent. Incoming messages of starting conversations are delivered to the Planner module and the other ones, which are already a part of some conversation, are delivered to the Executor module. The Router module can also be responsible for the security issues.

The Planner module creates a set of independent tasks representing a received message and it gives this set to the Scheduler module. To carry out the message decomposition, the Planer module needs knowledge about the external world state, the right ontology to understand the message content, a plan library with templates of tasks decomposition, information about agent's capabilities and information about agent's goals.

The Scheduler module, based on constraints of individual subtasks that were received from the Planner module, creates an order according to which the subtasks have to be executed. The execution order is stored in the schedule base from where it is picked up by the Executor module.

The Executor module executes tasks from the schedule base and keeps information about running tasks in the action base. The Executor module updates particular info bases when it calculates or receives results from other agents. In the case,

that the Executor module has to communicate with another agent to accomplish a particular task, the Executor module passes the incoming messages to the Planner module to process them.

Information agent architecture should not have any influence on agent external behavior in the sense that agent architecture has to be transparent for agent communication. It means that agents of different architectures have to be able to communicate with each other.

4.3.6 Multi agent architecture

Multi agent architecture corresponds to relations among individual agents and roles that agents have in these relations. Architecture has great influence on agent behavior. The type of agents' relations is established by the environment where agents operate. For example in a market system, agents do not create any tightly-coupled relational hierarchy. On the other hand, information agents in process automation tend to form a hierarchy system that better corresponds to the process systems. It is up to an information system designer what type of a hierarchy he chooses. In principle, the designer can choose from multi-agent architectures shown in Figure 4.7 and their mutual combinations.

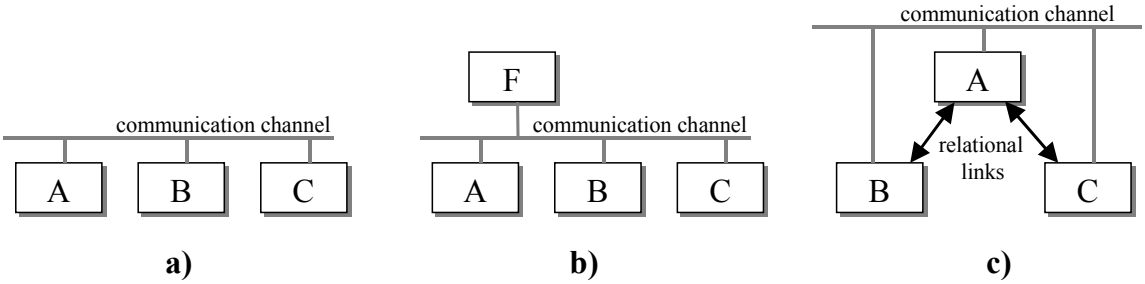

Figure 4.7 Agent architectures for information access. a) Flat Peer-to-Peer (P2P) architecture. b) Flat architecture with agent facilitator (adopted from [Finin et al., 1994]). c) Hierarchical architecture.

A facilitator agent F, in Figure 4.7b, provides information about or from agents to other agents. The facilitator agent is described and used in [FIPAOS, 2003]. After their creation, individual agents register themselves to the facilitator agent so they can be discovered by other agents. The facilitator agent informs on demand individual agents about other agents and their services. The facilitator agent can also serve as a broker that handles information queries by itself and returns only final information. In process automation, the facilitator agent can be used after system startup to initiate information agent structure configuration and/or it can be responsible for Plug and Play behavior by informing the current agents about a new agent and vice versa.

The non-hierarchical agent system without an agent facilitator, Figure 4.7a, is also called peer-to-peer system (P2P). Every agent has its own objectives that it tries to

accomplish. Agent knows their neighbors by observing the environment or from replies to broadcast messages. Agents ask each other to provide information or to perform an action. In the case that neighbor agents are not able to accomplish given queries, they can delegate those queries to their neighbor agents and so on. This architecture provides the highest level of reliability because no agent has a special role. Individual agents are entirely independent. A failure of one agent has just a little influence on the whole system. On the other hand, this type of architecture is difficult to control and to foresee the whole system's behavior. Information access in process automation has to be deterministic and fast responding and therefore P2P system is not proper information architecture in process automation.

In *the non-hierarchical system with an agent facilitator*, Figure 4.7b, one or more agents have the role of the agent facilitators. Agent facilitators can decrease the total number of messages sent among agents but they can also become communication bottlenecks in systems with many agents. According to [Ben-Ami and Shehory, 2002], large agent systems without a facilitator are faster communicating than systems with a facilitator. Information agents using the agent facilitator can create fast and flexible information systems where the number of agents is not high. The potential disadvantage is that all agents rely on the facilitators, which in the case of their failure can bring problems to the whole system.

The *hierarchical agent system* (based on tree or modified-tree hierarchical architecture) presumes that agents hold information about relations to other agents. These relations can be pre-configured or they can be dynamic. The latter case is more difficult to implement (probably with help of an agent facilitator) but it provides higher flexibility to system changes. Hierarchical architecture naturally represents process systems which can be divided into independent functional blocks that can be farther divided into smaller independent blocks and so on until they are represented by individual sensors, actuators and parts of the physical system itself. Information agents in hierarchical architecture possess information about their subagents (acquired by learning or by pre-configuration) and they can request them to provide specific data or perform a specific operation. Agents have links only to their sub-agents and the super-agent in agent system hierarchy. The whole automation system is represented by the top-level agent. The top-level agent serves for external agents as an entry point to the system and provides information to them. The top-level agent can provide addresses of its sub-agents to allow establishing of direct communication with them, or the top-level agent can encapsulate the whole automation system and can act as an exclusive information provider to the system. The similar behaviour may have all information agents that have any sub-agents. Encapsulation of a system part is suitable when information from individual subparts is bound together. Then, it is up to the super agent to evaluate all data together and to provide consistent information to other agents.

The facilitator agent can serve as an alternative or as a support agent for hierarchically structured information agents in process automation. For example, when a user agent wants to know a specific device's variable value it can ask the facilitator for the address of the information agent that is in charge of that device instead of asking the process agent that would have to propagate this task to all its subagents until the specific device.

Mobile agents that are able to travel from one system to another play a special role in agent information architecture. In information access in process automation, mobile agents could for example represent a client diagnostic application. Such a diagnostic agent can travel to a factory information system to perform diagnostic operations that can require high data transfers. All operations can thus be performed inside the company's information system and therefore they are secured and they do not require a permanent Internet connection with the mobile agent's home. When a task is completed, the diagnostic agent returns back with a result.

4.3.7 Message processing

Information agents process messages according to the type of the messages, knowledge they possess and the relation they have with the message sender. The information agent can either have all the necessary information to answer the message or not. In the second case, the information agent has to be able to get required information from the same or from other agents. The incoming message then needs to be decomposed into independent sub-queries that may even include querying another information agent. A complete answer may thus be composed of data from several information sources represented by other agents. In such a case, the information agents have to know how to merge different data types if necessary and how to formulate the correct answer. Data from various sources are of the same type or not. In the first case, data can be merged for example by statistical functions (for instance, when an information agent retrieves temperature values from different sensors in the same area, the agent can calculate an average value from them) that can use weighting coefficients to give different importance to different data sources. The weighting coefficients can be preprogrammed or they are learnt from the communication process with other agents. Data from various sources can also be simply put into a data set that is included in the answer (for example when an information agent is asked to return the names of all devices in a process system, the information agent retrieves from other agents individual names and creates a set from them). In the second case, when data returned from several data sources are of different type, an information agent needs to understand the relation among individual data and their particular importance for the answer (for example, when an information agent is asked if a system is ready for startup, the information agent needs to ask other agents for values of their variables to evaluate the current system state).

An example of data decomposition is demonstrated in a process system represented by three hierarchically structured information agents, see Figure 4.8. The

process agent is the highest-level agent representing the whole process system. It has two sub-agents, the pump and the tank agents. The process agent is sent a message querying the readiness of the process to start running. The message written in the FIPA-SL language can have the following form:

(iota ?x (= ?x (process state)))

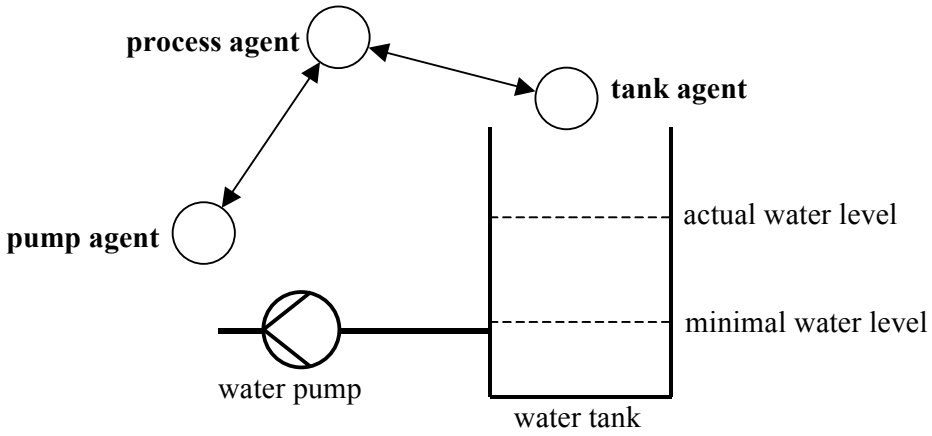

Figure 4.8 Water system and its information agents.

The process agent, based on knowledge about its subagents, decomposes the message into two separate messages. In the first one, the process agent queries the pump agent whether the state of the pump is in order and in the second message, the tank agent is asked what the water level is:

(iota ?x (= ?x (device state)))
(iota ?x (= ?x water_level))

The answers in both cases are of different types, in the first one it is a string and in the second one it is a number:

(=(iota ?x (= ?x (device state))) ready)
(=(iota ?x (= ?x water_level)) 523)

The process agent has to understand to both answers as well as it has to know what the minimal water level is to start the system.

4.3.8 Security

As it is discussed in [Castelfranchi, 2001], information does not represent only data but it has also social nature. Specific information has different importance for different users. It needs to be secured so only authorized user has an access to it. Because the agent information system in process automation is a cooperating system, where all information agents are trusted, the main security bottleneck is user access to the

information system both from a company's intranet and from the Internet. Communication among information agents in a plant in most cases does not need to be secured and is thus not slowed down.

An agent information system has to provide services that allow authentication and authorization of user agents and coding and signing messages. With regard to short time responses to queries, it is not favorable that every agent in the agent information system possesses a mechanism for securing and verifying messages. Instead, special security agents should be used that serve as mediator agents between user and information agents. After connecting to the agent information system, for every user agent is instantiated a security agent that authorizes the user agent and then handles all the user agent's requests. The security agent provides access only to authorized information agents and data. This type of security requires the security agents to understand the user agents' queries and to be able to evaluate their competence. In larger systems, where many users are involved, also functionality for trust delegation needs to be implemented. This functionality includes giving rights to other user agents for performing certain queries, to specify time constraints of given rights and to specify what user agents can delegate what rights and to whom, etc.

Another way how security could be implemented into the agent information system is the use of permissions for certain queries. These permissions would be issued by authorization agents and required by information agents (and for example implemented in the router module of agent architecture shown in Figure 4.5). In this case, access to every information agent would be secured even inside the agent information system but the total response time would be higher than in the previous case thanks to additional security processing.

In all cases, security mechanisms should use the world security standards to not become a stumbling block in agent communication. A suitable approach is to make the agent-applicable security mechanisms publicly available from agent repositories so agents can negotiate over a specific security level and tools to reach this level before the data retrieving.

4.4 Implementation

When implementing information agents in process automation, it is important to ensure that agents that influence process control have to satisfy all message delivery-time limits. Nevertheless, information agents are not meant to be a part of the control system's core. Technologies used for implementing of information agents in the Internet, such as the XML data format and the SOAP protocol over HTTP, are not necessarily the most suitable. They are designed for communication among different platforms and over large distances which is seldom the case of information access in process automation. In process automation, distances among individual devices and thus among agents are mostly up to several hundreds of meters. The control system is usually implemented with

use of network technology provided by one manufacturer and therefore there are no problems with communication incompatibility. The emphasis is given more to reliability, deterministic behavior and on short time delivery. Determinism in sending messages is hard to implement in agent communication and hence other agent advantages have to overcome this limitation.

Currently, communication among controllers and user applications is done for example with help of the OPC specifications through DCOM technology. Communication among controllers and intelligent devices is done through industrial networks and their protocols and communication among controllers and passive actuators and sensors is done directly through IO modules. In automated systems composed fully from agent enhanced devices, data transfer does not occur so often due to the autonomy of the agents. This fact balances the disadvantage of asynchronous agent message delivery mechanism. In Figure 4.9 are depicted the traditional and the agent based information access communication network used on a factory floor. In systems, where passive and agent enhanced devices need to be used together, agent wrappers will be necessary for the passive devices.

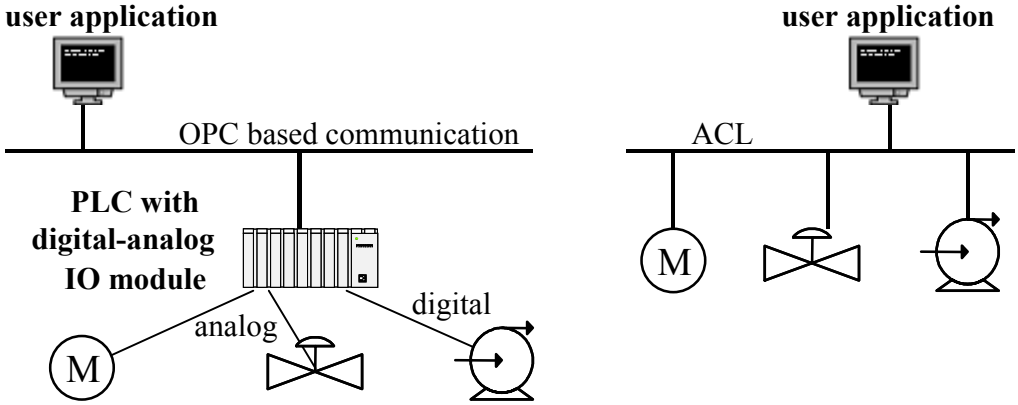

Figure 4.9 Traditional (on the left) and agent based (on the right) information access to sensors and actuators in process automation.

Information agents that do not affect process control can take full advantage of technologies described in the chapter 3. User agents connected wirelessly to a process system can use the special designed Wireless Message Transport Protocol (WMTP) that provides to agents efficient and reliable communication. More information about WMTP can be found in [Laukkanen, 2002].

4.5 Comparison with OPC

OPC technology is currently one of the state-of-the-art technologies for communication mainly among process controllers and user applications. It offers process data access, an event and alarm notification, historical data storing and access. It supports server-to-server communication, remote data access over the Internet and capabilities to secure data access and transfer. OPC technology is built on DCOM technology used at

the MS Windows platforms. The biggest advantages of OPC are the standards that ensure intercommunication between applications written by different development teams, high functionality provided by individual OPC interfaces, support for new technologies coming with increasing use of the Internet and support for automated data access that enables programmatically enumerate OPC server data and its functionality.

Information agents are able to offer the same functionality as OPC with regard to the quantity of provided services. Moreover, agent functionality is not bound to any specifications that define concrete services as it is in the case of OPC. Agent functionality is fully extendible as the agent communication language is by ontologies. For example, information agents can provide information about relation between two different process variables, variable limits, variable unit etc. Agents are able of high-level data preprocessing and reasoning due to intelligence and knowledge they possess. In contrast to OPC, information agents form an information system with distributed knowledge of a process system. With advent of more and more intelligent control system devices such as embedded systems, information agents will become their natural representation in information systems.

On the other hand, in agent information system, messages are sent exclusively asynchronously. That is, delivery times are unpredictable. This can cause problems when information agents constitute a part of the control system. Also, message processing is slower than DCOM calls used in OPC. However, agent technology is not meant to replace OPC. Agent technology is more regarded as a higher layer using and working over OPC.

4.6 Advantages and disadvantages

Information agents bring many new features into the information systems in process automation over the currently used technologies. Their implementation is also rather different. Whether information agents are going to succeed in process automation depends on the advantages and with them associated profit emerging from adopting agent technology.

The main advantages that information agents have over the current information technologies are the following. First, agents distribute knowledge of a process system into the corresponding parts of the system. It is natural that individual system components possess knowledge that is relevant to them. It means that device manufactures, which have the most detailed information about their devices, can equip them with this information at the factory where the devices are assembled. Control system designers can thus focus more on the functionality of the whole system than on the issues associated with communication and diagnoses of individual devices. Therefore, the designers do not need to be experts on individual products from individual companies and they can invest their time in studying more general problems.

Second, information agents divide a process system into independent and autonomous parts that are able to give information about themselves. Provided information is not limited to simply-measured data but it can be preprocessed according to a requestor's needs and knowledge that the information agents possess. Higher-level agents coordinating other agents thus do not need to know their physical properties.

Third, information agents not only provide information from a process system but they also understand to this information. They can offer a description of provided information to a human user and/or they can be queried by other agents for an ontology associated with that information. With help of the ontologies, agents can understand various queries and they can form various messages.

Further, information agents are suitable for human-machine interfaces. They make information access usable by non-technical users. Agent-based information access moves device specific knowledge from clients to agents. Clients thus can be simple applications focusing mainly on presenting information and on friendly human user experience. Information agents cooperate with each other and therefore they allow creation of flexible systems that can be modified at runtime and that automatically reconfigure themselves according to their surrounding environment and their new tasks.

On the other hand, information agents are still in research. It is known what behavior and functionality can be expected from agents but their implementation still needs to be studied. Number of standards need to be established before information agents will be able to fully demonstrate all their predicted capabilities. Integration of the information agents into the legacy systems and vice versa will require additional software. Moreover, agent communication including ACL message processing and evaluating is much slower than simple function calls with known parameter types used in the current information systems.

4.7 Future vision

Information agents will be intelligent software components. Due to the capability to understand and exchange their knowledge with other agents, they will be able to communicate with each other regardless knowing each other before. Information agents in a process system will be able to retrieve, extract, analyze, filter, monitor and update data in that system. They will allow information synthesis and presentation. Information agents will adapt to the surrounding environment for example to changes in the structure of the system and they will behave in an optimal way according to given requirements.

It is hard to predict whether and when information agents will be a common part of process automation. Use of information agents in process automation is demonstrated in Figure 4.10. Information agents will be associated with every device, controller and user application in the information system in process automation. Information agents will be sold by device manufacturers along with process-control devices. Such agents will possess all detailed technical information about devices such as variable limits, device

constants, self-diagnostic tests, etc. and they will be able to automatically cooperate with other agents from other manufactures. Information agents will allow plug and play functionality for process devices by virtue of the ontology databases where domain specific information will be stored. Information agents will store all technical knowledge about individual process devices. Therefore, user applications will be very simple and their main task will be to represent data from a process system. Clients will be able to connect to a process system and to automatically display data from the system both textually and graphically. Clients will be represented by all different devices and applications, their location will not be important. When it is necessary, communication between agents will be secured and access into the information system will be monitored and restricted only to the authorized users and/or agents.

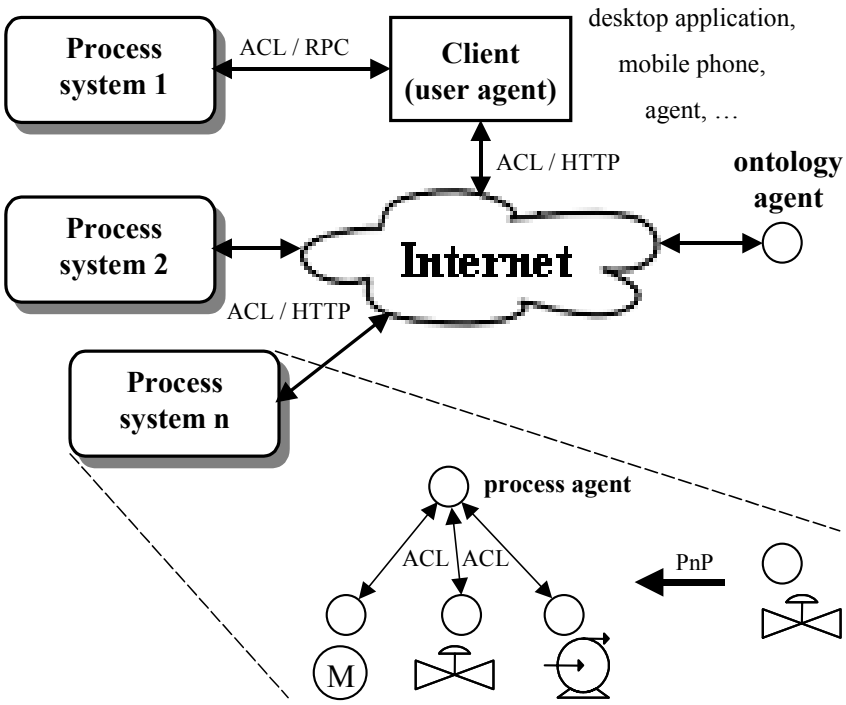

Figure 4.10 Future vision of the agent information system in process automation - a versatile client application and Plug and Play process device installation.

5 Designing information agents

5.1 Introduction

This chapter introduces one possible way of information agents design. The design starts from the FIPAPC architecture, developed at the HUT Automation Technology Laboratory, which itself is based on the FIPA-OS agent platform. The information agents extend the FIPAPC agents by, here proposed, an information access module. This module serves as an add-on feature that enables to the existing FIPAPC agents to deal with messages containing the information access ontology. In addition, a proposed mechanism for agent message interpretation is here described.

5.2 Underlying architecture

5.2.1 FIPA-OS agent toolkit and platform

The FIPA-OS is both an environment for running agents and a set of tools for creating the FIPA compliant agents. It is written in Java programming language. Inner architecture of the FIPA-OS agent platform corresponds to the FIPA reference architecture model ([FIPA, 2002]), see Figure 5.1a. The Directory Facilitator (DF) and the Agent Management System (AMS) are represented by agent-management agents.

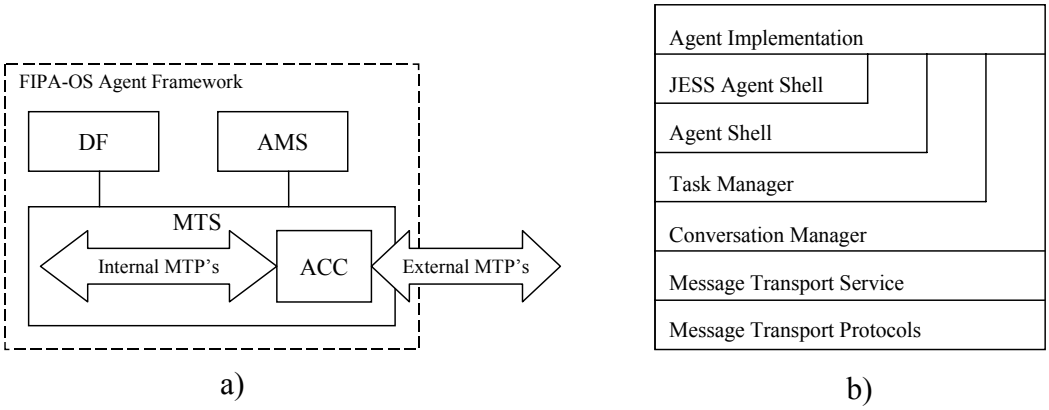

Figure 5.1 a) FIPA reference model (adopted from [FIPAOS, 2003]), b) the main components within FIPA-OS agent platform (adopted from [Emorphia Ltd., 2002]).

The DF provides "yellow pages" services to agents running in the platform and the AMS provides agent lifecycle management. The Message Transportation System (MTS) with use of the Internal and External Message Transportation Protocols (MTP's) and Agent Communication Channel (ACC) are responsible for message delivery between agents within one or several agent platforms.

The main components of the FIPA-OS agent programming toolkit and their individual relations are in Figure 5.1b. The Message Transport Protocols and the

Message Transport Services are hidden to a programmer. They ensure communication among agents in the local and the external platforms. The Conversation Manager tracks agent messages at the performative level and assures that the order of messages in a certain conversation follows the particular conversation protocol. The Task Manager administers running of Tasks. The Task is a Java thread that performs specific operations, it is able to return a result and it can send and receive messages to and from other agents. The Agent Shell provides the `FIPAOSAgent` class that is used as a base class for all agents. The JESS Agent Shell gives to agents possibility to use the JESS expert reasoning system.

5.2.2 FIPA for Process Control (FIPAPC)

The FIPAPC is built on the top of the FIPA-OS. It extends the FIPA-OS agents to be easily employable in process control and it serves as a foundation to the proposed information access module. The FIPAPC agent platform was designed and implemented by the agent group at the HUT Automation Technology Laboratory, see [Appelqvist et al., 2002], [Seilonen et al., 2002c] and [Pirttioja, 2002]. The previous attention in the FIPAPC was given to the distributed process control planning. Agents were hierarchically structured to represent a process system. They were programmed to cooperate with each other and to plan their actions to achieve desired process system behavior. For example to start up the process system, agents first planned what actions and in which order had to be accomplish and then agents performed individual actions according to the agreed plan. Further in the project, agents were used for automatic fault recovery where they negotiated with each other about changes of their set points to decrease the impact of a device failure on the whole system behavior.

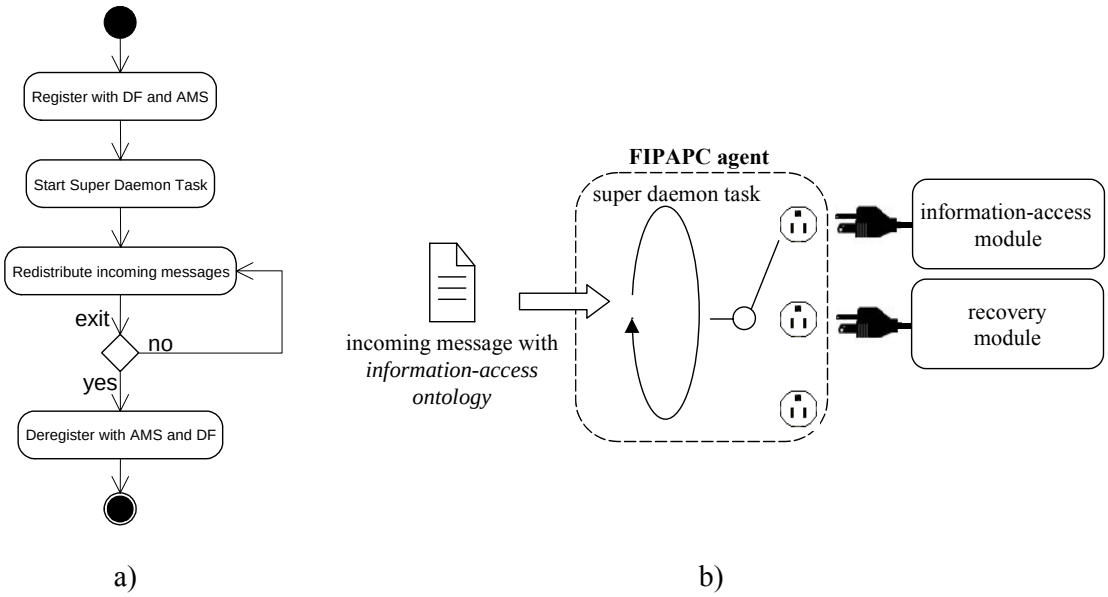

Figure 5.2 a) the life cycle of the FIPAPC agent, b) the FIPAPC agent module architecture.

The FIPAPC agents implement their individual abilities in independent modules, for example the recovery module or here designed the information access module. Because most of the agent functionality is in modules, FIPAPC agent code itself is simple and it includes mainly agent initialization. The principal operations that the FIPAPC agent performs during its life are shown in Figure 5.2a. After being started, an agent registers itself to DF and AMS agents. Then the Super Daemon Task is started. It runs individual ability modules according to configuration settings in a profile file and then it redistributes incoming messages to appropriate modules, see Figure 5.2b. Every agent message contains a name of the ontology that is used in the message content. This ontology name is used for searching the right module that is able to handle the message. All modules are inherited from the `Task` class, which allows the Super Daemon Task to maintain all modules in the same way.

5.3 Design of the information access module

Individual agent modules represent agent's ability to deal with messages from different domains. Agents can be 'taught' to process messages with new ontology by installing and using appropriate ability module. This philosophy of ontology implementation is rather different from that one mentioned in the chapter 4, where all agent modules are proposed to be ontology independent. There, ontologies are expressed with use of an ontology language that agents know to interpret. In this design, an ontology is represented by functionality stored in Java classes that corresponds to the terms and relations specified by given ontology. This approach is not flexible to changes in ontologies but it is many times easier to implement.

The principle of processing messages in the information module starting from the

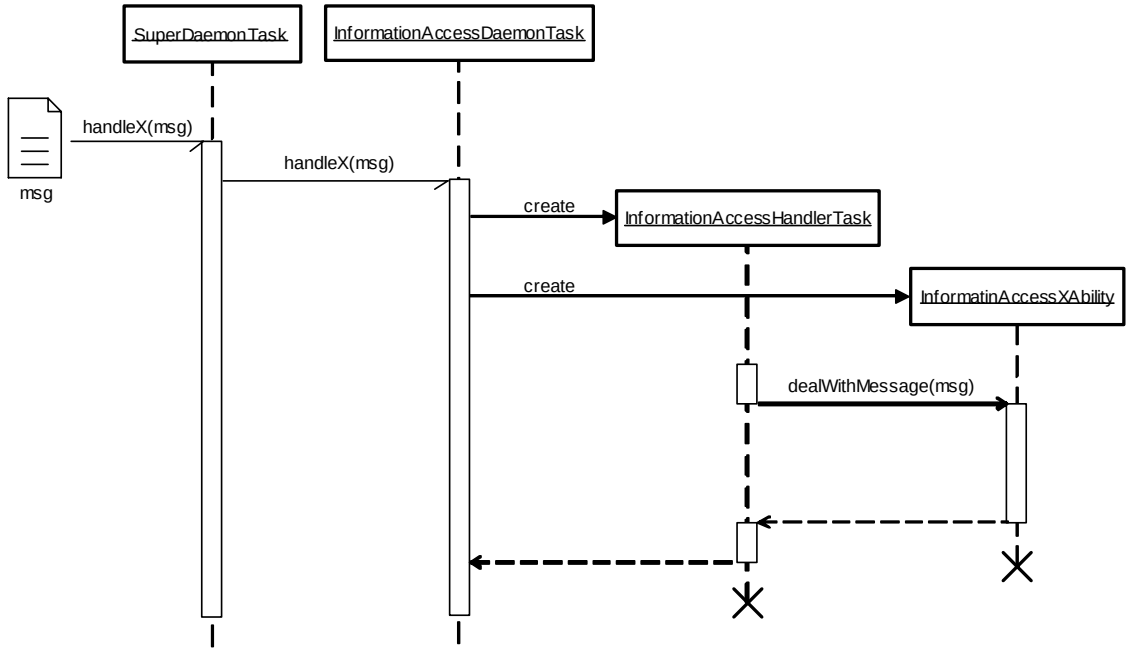

Figure 5.3 Message processing in the information access module.

FIPAPC architecture is in Figure 5.3. First, ability modules supported by an agent are configured in the agent XML configuration file referred to as an agent's profile. For the information access module, the agent's profile contains the name of the information access ontology: "informationaccess" and the name of the class representing the entry point to the information access module: "InformationAccessDaemonTask"

```
<DaemonTasksProfile>
     <DaemonTaskDescription
          daemonTaskClassName="InformationAccessDaemonTask"
          ontologyName="informationaccess" />
</DaemonTasksProfile>
```

When a message with a header containing the performative X (determining the type of the message such as: query, subscribe, etc.) and with the `informationaccess` ontology is sent to an information agent, the agent's Super Daemon Task calls method `handleX(...)`, which is provided by the `InformationAccessDaemonTask` class and to which the received message is given as a parameter. Every message processing is done in a separate program thread that is started within the `handleX` method and that is represented by the `InformationAccessHandlerTask` class. This class is independent on the message content and it uses the `InformationAccessXAbility` classes (referred to as the ability classes) to deal with concrete messages according to the X performative used in the message header.

Functionality of ability classes corresponds to the types of the message performatives. Ability classes are responsible for sending messages according to the rules given by a conversation protocol that is associated with every message. Some performatives requires a special treatment with the message. For example according to the FIPA standards, the `subscribe` performative requires that a message has to be stored along with the result obtained from message evaluation and regularly reevaluated to monitor the result changes and to notify a subscribing-agent about these changes. Such an extra functionality needs to be specially implemented and if a requirement comes later for its implementation it unfortunately means changes in many already existing Java classes of the information access module.

The ability classes cover all functionality that is required for dealing with particular performatives and conversation protocols but they do not contain code for message content processing. This functionality is provided by a special set of message interpretation classes, see Figure 5.4. The most important class in Figure 5.4 is the `InterpreterBuilder` class. This class provides the method `createInterpreter` with string parameters representing a message content and the name of the agent language in which the message content is written. As a result of

calling the `createInterpreter` method, an instance of an object with the `Interpretable` interface is returned. The `Interpretable` interface provides the `interpret()` and the `isInterpretable()` methods. The `interpret()` method interprets a message content and returns a result. The `isInterpretable()` method only checks whether the message content is interpretable or not. With help of the `InterpreterBuilder` class, message processing in the ability classes is represented only by calling these two methods and by processing returned results.

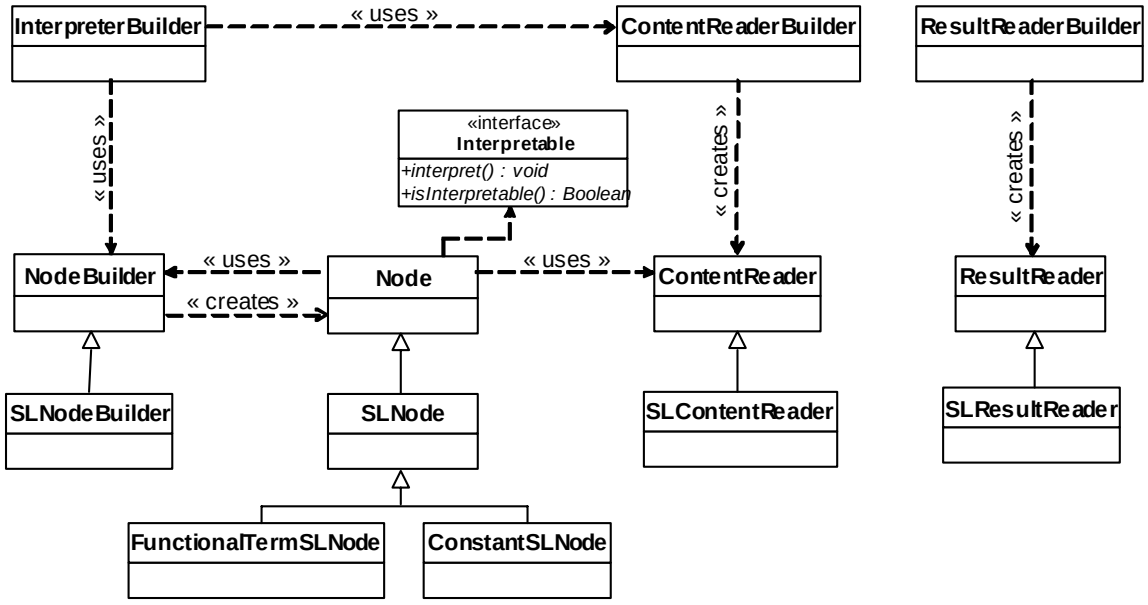

Figure 5.4 Class diagram of message content interpreting classes.

The mechanism of message content processing is based on building a grammar tree that is representing the message content. A grammar, which specifies a language syntax and which enables analysis and evaluation of sentences written in that language, has to be defined for every agent communication language. Messages can be interpreted as trees according to these grammars. For example in Figure 5.5, there is a sentence written in the SL agent communication language and its grammar tree representation. The grammar tree contains names of Java classes that match the corresponding grammar elements. Every tree node object is inherited from the `Node` class and thus it implements the same `Interpretable` interface, see Figure 5.4. Individual `Nodes` possess functionality according to the grammar elements they represent and they know what kind of child nodes they can be related to. A reference to the grammar tree root `Node` object is returned to an ability class after calling the `createInterpreter` method. The ability class interprets the message content by calling the `interpret` method of the root `Node`. The root `Node` object in its `interpret` method implementation calls the `interpret` methods of all its child nodes, which again do the same procedure with

their child nodes and so on. The returned results from child nodes are processed and the final result is returned to the ability class.

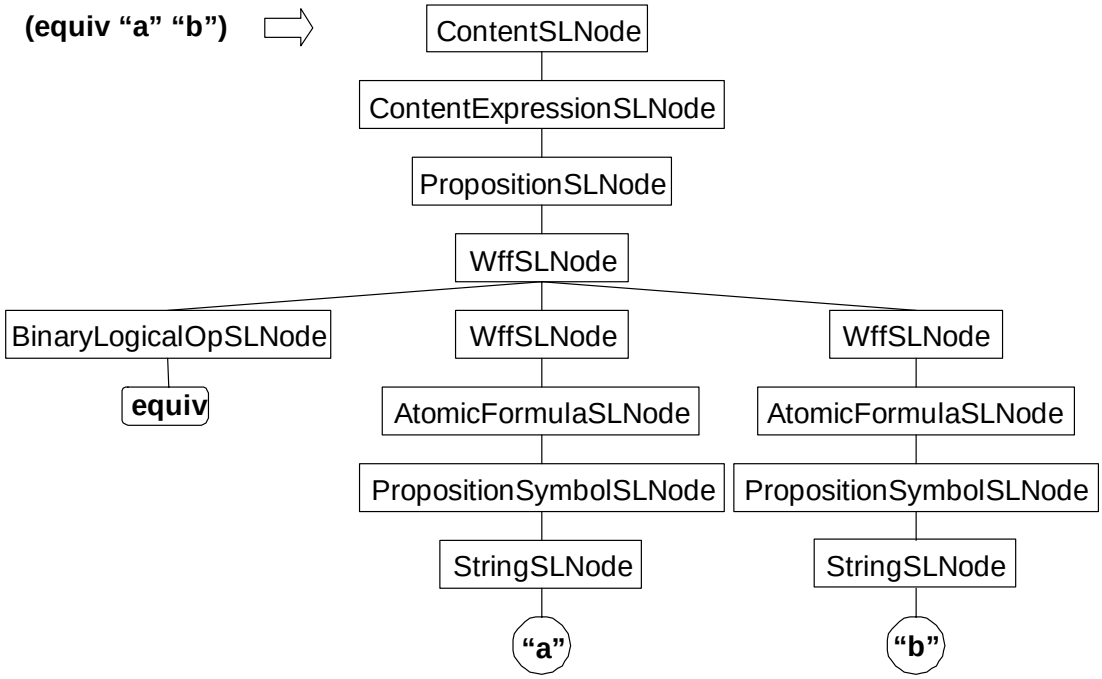

Figure 5.5 SLNode object tree.

Because grammars of all languages are represented by objects inherited from the same `Node` class, the ability classes always use the same `Interpretable` interface to evaluate a message content independently on the used agent language. In this thesis work, only the SL language grammar was partially implemented. The grammar elements that were implemented were chosen according to the messages that were sent among agents in the demonstration scenario, see the chapter 6. The SL language grammar specification determines a fixed set of terms that correspond to the core of the grammar. Further, SL language sentence can contain constants, variables, functions, etc., according to the domain where an agent is used. Therefore, if whatever sentence from a certain domain needs to be interpretable then all grammar nodes have to have their `Node` class implemented as well as all the specific terms used in that certain domain. When an agent needs to learn new ontology, it needs to get the `Node` classes representing the terms from that ontology and when an agent needs to learn new agent language it needs the `Node` classes representing the grammar core of that language.

An example of proposed SL-language sentence processing is in Figure 5.6. A query sentence is first divided into separate terms. Then, the SL language parser creates a static tree representation of the query according to the SL grammar. The structure of the grammar tree is simplified in Figure 5.6. The full tree structure is in Figure 5.5. Next, a tree consisting from the `Node` objects is created and the `interpret` method of the root

node is called. According to the SL language grammar, the final result contains the original message with an evaluated result.

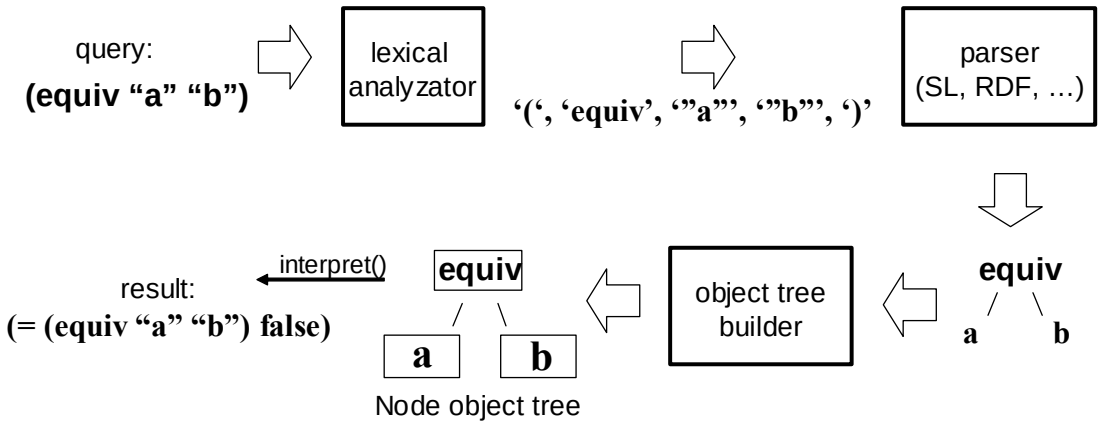

Figure 5.6 SL language sentence processing.

The last thing to discuss, as for the proposed design of the information module, is creation of object tree representations from agent language sentences. After the `createInterpreter` method of the `InterpreterBuilder` class is called, instructions from the XML configuration file are read.

```
<BuilderProfile>
     <BuilderDescription
          contentLanguageType="FIPA-SL"
          nodeBuilderClassName="SLNodeBuilder"
          contentReaderClassName="SLContentReader"
          resultReaderClassName=" SLResultReader" />
</BuilderProfile>
```

The attribute `contentLanguageType` specifies the type of the agent language. The attribute `nodeBuilderClassName` determines the name of a class that is inherited from the `NodeBuilder` class. The `NodeBuilder` class provides methods for dynamic creation of `Node` objects based on the name of the grammar elements they represent. The attribute `contentReaderClassName` contains the name of the class that creates static grammar tree representation from an agent sentence and that allows to iterate through this tree. The attribute `resultReaderClassName` contains the name of the class that is able to return a result from interpreted sentence. This functionality is necessary because the results of a sentence evaluation contain also the original sentences. All classes specified in the configuration file are loaded dynamically at runtime with the help of Java reflection functionality. Therefore, ability

to process new agent languages and ontologies can be added to an agent at runtime without any need for its restart or code modifications.

When creating the grammar object tree representation of an agent language sentence, the `InterpreterBuilder` class first creates the `NodeBuilder` and `ContentReader` objects according to the configuration file. Then the `ContentReader` object is used to get the name of the root node from the static tree representation of the sentence and the `NodeBuilder` object creates a `Node` object corresponding to the root node's name. Every Node object has the ability to create its own child nodes according to the provided `ContentReader` object. Creation of the rest of the grammar object tree is done through delegation of the `ContentReader` object to child nodes and repeating the procedure above.

5.4 Design analysis

The designed mechanism for message processing is extendible to messages written in any kind of agent language. Message content is represented by an object tree that corresponds to the used grammar. Ontologies are implemented by set of Java classes and learning an ontology means copying corresponding classes to an agent and to reconfiguring the agent's profile. Moreover, every agent possesses classes that implement its knowledge of a particular device for which the agent is responsible for. For example, a pump agent will have a class for starting the pump and a temperature sensor agent will have a class for reading a temperature value.

. This approach is relatively easy to implement and for demonstration purposes in this thesis, which are not focused on the work with ontologies, it is sufficient. According to the chapter 4, agent architecture should support all kinds of communication protocols. The designed information access module deals with different protocols in separate ability classes. Adding a new ability class means changes in information access module code and requires recompilation and an agent restart. Functionality that would allow runtime protocol exchanging is far beyond the goal of this thesis.

The information access module is supposed to be used mainly by the information agents that represent data sources from individual parts of a process system. Other functionality such as human friendly communication was not considered. Also security considerations were not taken into account in the design. Nevertheless, securing the communication with external agents can be provided by special security agents as it is proposed in the chapter 4.

6 Implementation and test of information agents

The concrete use of the information access module described in the previous chapter is demonstrated in a test scenario. Information agents are connected to a real physical system. They read data from this system to evaluate its current running status and they provide this information to other agents.

6.1 Test environment

For demonstration of information agents, a real laboratory physical system that is shown in Figure 6.1 was used. This system was constructed for research purposes. It

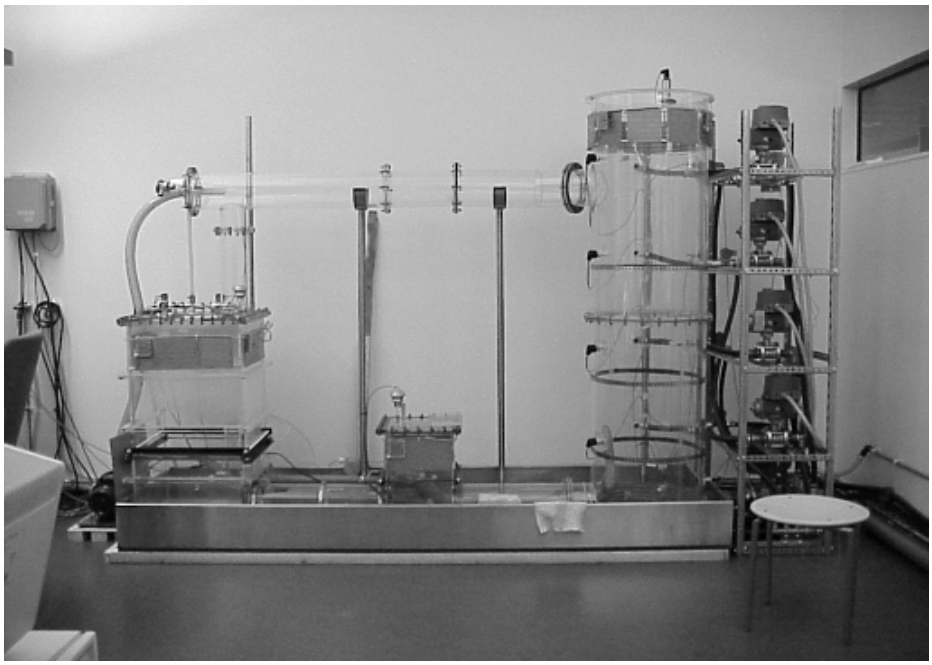

Figure 6.1 The laboratory physical system.

consists from two water tanks of the total volume 800 liters that are interconnected at the top and at the bottom with tubes. Inside the right water tank, different temperature in the lower and in the higher part is induced with help of four automatically controlled valves with cold and hot water. Water circulation in the whole system is managed by a water pump.

All valves and the pump are interconnected with the Smar controller (manufactured by the Smar International) through the Fieldbus industrial network. The Smar controller is further connected through an IO module to thermometers. Information from the controller is read by the Smar OPC DA server running at the Windows 2000 platform. From this OPC server, information is provided to all OPC DA clients such as monitoring applications and also to agents running in the FIPA-OS platform. To enable communication between Java agents and the OPC server, the Java JOPCClient library is used.

6.2 Demonstration system

6.2.1 Description

A common task that needs to be implemented in process automation is diagnosis of a system status. This task is usually done by a client application that is connected to a controlled system and that collects and evaluates system's data. A disadvantage of this solution is in the tight connection between the client and the system. Therefore, the client update is required when the system changes.

The implemented scenario demonstrates the use of information agents for a system diagnosis. Agents are supposed to monitor the system state and to report its changes to a user. In later research, information about the system state should be utilized by agents implemented in [Chakraborty, 2003] to start the automatic fault recovery process. The diagnosed system is represented by the laboratory physical system that is described in the previous chapter. The physical system is virtually divided into three functionally independent parts, the pump, the lower and the higher parts of the right

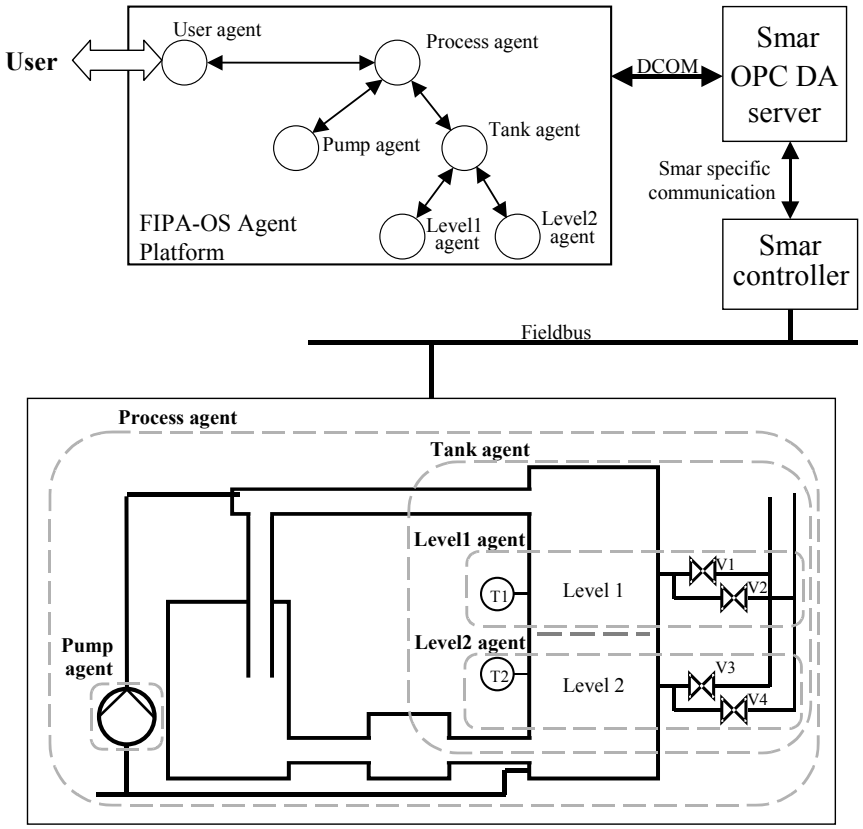

Figure 6.2 The real physical process system.

water tank, see Figure 6.2. Each of these parts is represented by an information agent. Moreover, the whole right water tank and the whole physical system itself are also represented by information agents.

Information agents possess technical parameters and functionality for diagnosing those system's parts they represent. In the test scenario, information agents are asked by a user agent to monitor the status of the physical system and to send a notification when the status has changed.

6.2.2 Implementation

In the demonstration system, one user agent and five information agents were implemented, see Figure 6.2. In contrast to the information agents, the user agent does not use the information access module. Functionality of this agent is rather passive. It only sends and receives messages to and from other agents and displays these messages in a Graphical User Interface (GUI) application to a human-user. The GUI was adopted from the FIPA-OS IOTestAgent and modified for use in the test scenario. The user agent sends messages based only on the human-user's commands.

Information agents are organized in a hierarchical structure according to the relations that are among the physical system parts which the agents represent, see Figure 6.2. Individual relations are specified in the agent profiles by a list of sub-agent names. For example the process agent's profile contains the following data:

```
<FIPAPCAgentProfile>
     <subAgent>pumpagent</subAgent>
     <subAgent>tankagent</subAgent>
</FIPAPCAgentProfile>
```

The whole physical system is represented by the process agent, which is the parent agent to all other information agents. The user agent communicates only with the process agent and it does not know about other information agents. This approach allows the user agent to be very simple and to be able to connect to whatever other physical system represented by a process agent.

For the test scenario, an ability class supporting the FIPA Subscription protocol (see Figure 6.3) and the protocol itself were implemented. In the demonstration system, the user agent utilizes this protocol to subscribe itself to the process agent to get the current status of the system state and to be informed about any future changes. The response of the process agent and its sub-agents to the subscribe message can be separated into two parts.

First, it is a subscription part. After receiving the subscribe message, the process agent resends this message to its sub-agents who resend it to their sub-agents and so on. Before accepting the subscription request, all information agents try to evaluate the message content. It means that they check the status of the system part they are responsible for. If the evaluation is successful, agents send back an inform reply with the obtained result. In the case that the massage evaluation fails, agents send back a refuse reply and they also send a cancel message to all its sub agents. Therefore, before sending

the inform reply, agents, which have any sub-agents associated with them, wait for the replies from all of these sub-agents to be able to incorporate received results into the final answer. Hence, the process agent sends back the inform/refuse reply as the last one. In a real application, instead of sending the refuse reply, agents should try to provide at least incomplete information (default, historical or deduced value). In process

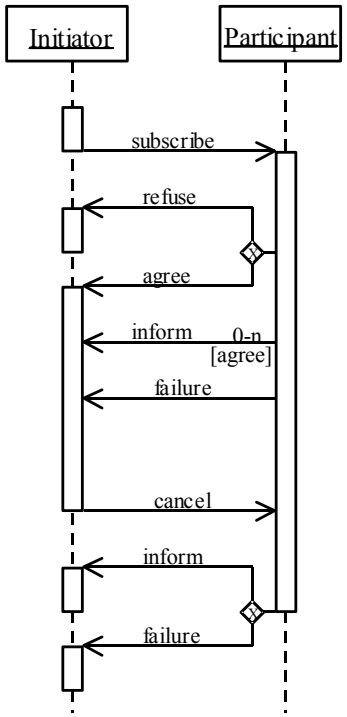

Figure 6.3 Communication protocols (adopted from [FIPA, 2002]).

automation, agents should never totally refuse their services. It is safer to use not precise information than no information at all.

Second, it is an evaluation part. After the successful subscription, information agents regularly evaluate all subscribed messages. If results differ from the previous ones, the inform reply with the new result is sent to the upper agent. That is, the inform message is not sent directly to the user agent but always by the process agent. This allows parent agents to process the inform message first and when needed to take a necessary measures regarding the result in the message. They can also reevaluate the result taking into account information they possess. In the test scenario, parent agents simply resend the inform reply to the agent that is on the way to the user agent.

The subscription and the inform messages used in the demonstration scenario are in Figure 6.4. Messages are written in the FIPA ACL language, see [FIPA, 2002b]. Every message contains identifiers of the sending and the receiving agent. These identifiers are agent names by which agents are registered in the FIPA-OS platform. Further, the messages include the `content` parameter, which here contains the request

```
(subscribe
        :sender (agent-identifier :name externalagent )
        :receiver (agent-identifier :name processagent )
        :content "((iota ?x (= ?x (Device state))))"
        :ontology informationaccess
        :language fipa-sl
        :protocol fipa-subscribe-full
        :conversation-id ID123
)

(inform
        :sender (agent-identifier :name processagent )
        :receiver (agent-identifier :name externalagent )
        :content "(= ((iota ?x (= ?x (Device state)))) STATE)"
        :ontology informationaccess
        :language fipa-sl
        :protocol fipa-subscribe-full
        :conversation-id ID123
)
```

Figure 6.4 FIPA ACL messages. The term STATE stands for one of: OFF, OK, WORKING_BUT, ERROR.

for monitoring the system state. Another parameter is `ontology`, which contains the name of the information access ontology. This value ensures that these messages are handled by the information access module. The parameter `language` specifies the content language and it is used by the `InterpreterBuilder` class to create the right `ContentReader` and `NodeBuilder` objects. In the current implementation, only and partially the FIPA SL content language is implemented. The parameter `protocol` contains the `fipa-subscribe-full` value. This string characterizes that the messages are parts of the fully implemented FIPA Subscribe protocol (by default not provided by the FIPA-OS platform). This parameter is used by communication services of the FIPA-OS platform to ensure that agents send messages in the sequence specified by this protocol. The parameter `conversation-id` specifies to what conversation the ACL messages belong. This is necessary because agents are communicating with their sub-agents and with their parent agent simultaneously. When an agent receives a subscription message, it redistributes this message to its sub-agents with the `conversation-id` parameter created from the original one and from the name of the sub-agent.

The argument of the `content` parameter: `((iota ?x (= ?x (Device state))))` in the subscribe message expresses an agent's request for the value of a device state. The string `iota ?x` defines the variable `?x` in the expression `(= ?x (Device state))` that assigns the value of `Device state` to this variable. The term `Device` represents a name of a function and the term `state` serves as a parameter

to that function. When the message content is being evaluated, first the `SLNode` object tree is built according to the grammar tree that the message represents. The FIPA SL content language grammar allows using any number of user specified functions, variables, constants, etc. but in messages only those ones, for which `SLNode` classes exist, can be used so the messages will be interpretable. Terms `Device` and `state` are the terms defined by the `informationaccess` ontology and they are represented by objects in the grammar tree according to Figure 6.5. When the SLNode object tree is interpreted, the object `DeviceSLNode` first checks if its parameter has the value `state.` If yes, necessary variables' values are read and evaluated from the part of the

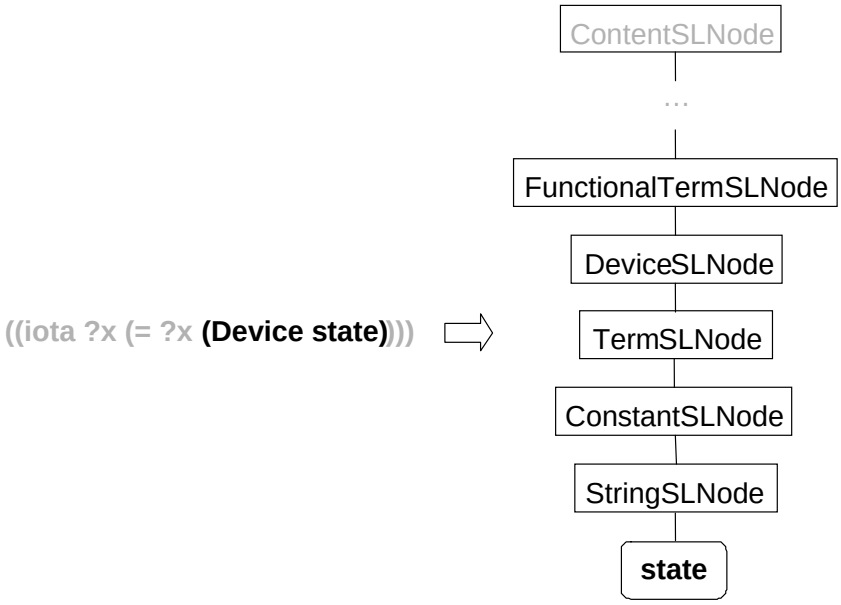

Figure 6.5. SLNode object tree for expression: `(Device state)` in the FIPA SL content language sentence.

physical system, which the agent executing the `Device` function is responsible for. This means, that every information agent has to have its own `DeviceSLNode` class because diagnosis code for every part of the laboratory system is different. On the other hand, all `SLNode` classes that represent only the SL content language grammar are for all agents the same. In what class namespaces are the agent specific and the agent general `SLNode` classes defined, is determined in the agent's profiles. When creating `SLNode` objects, the `SLNodeBuilder` first attempts to instantiate the objects from the agent specific and second from the agent general namespace using Java Reflection functionality. This technique enables to override the functionality of general `SLNode` classes by individual agents.

As a result of a diagnosis, the `DeviceSLNode` objects return one of the following strings: `OK`, `OFF`, `WORKING_BUT`, `ERROR`, see Figure 6.4. The string `OK` means that the diagnosed system's part is running without any problems. The string `OFF` indicates that the system or its part is switched off. The string `WORKING_BUT` indicates

that the system is working correctly but some values of system's variables are, for example, close to their critical values. This result can also be generated when a combination of system variable values is suspicious, etc. The string `ERROR` represents the case when some variable has reached its critical value in the system or some device is not responding and so on. Because information agents are hierarchically structured, upper agents are responsible for larger parts of the physical system than their sub-agents. When agents receive a message from their sub agents, they can reevaluate the message according to information from other parts of the system. For example an agent receives an `ERROR` result from its sub-agent. But the agent knows that the sub-agent is representing a physical system part that is back-upped by another sub-system, which is working correctly. Therefore the agent can change the result from `ERROR` to `WORKING_BUT` and send it to the higher agent.

To enable an agent to reason over the results from sub-agents, the meaning and the priority of all the possible results have to be implemented. In the demonstration scenario, individual results, their priorities and a method for determining priorities are defined in the `InformationAccessConstants` class. When an agent receives results from its sub-agents, the result with the higher is chosen. The priorities are in the following order: `OK`, `OFF`, `WORKING_BUT`, `ERROR`.

In the demonstration system, only the cold water valve in the lower part of the right water tank is under supervision. That means, only the `DeviceSLNode` class for the Level2 agent, see Figure 6.2, is carrying out system measurements and diagnosis, see Figure 6.6. First, the control mode and the opening value of the cold water valve are read. If the control mode is the manual regime, the result of evaluation is `OFF`. If the

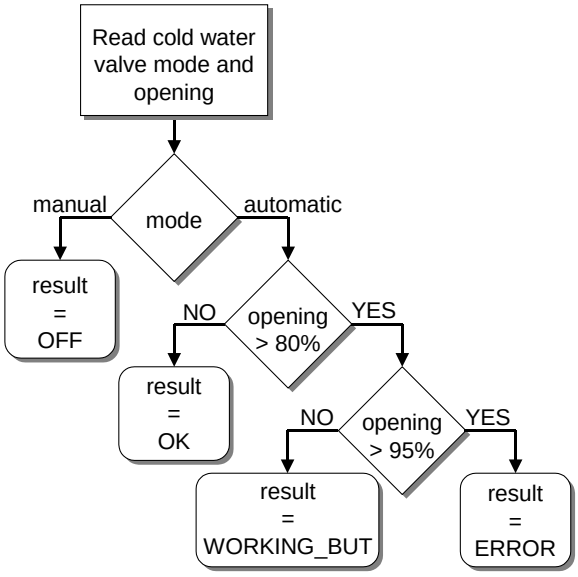

Figure 6.6 The implemented cold water valve diagnose algorithm.

control mode is set to automatic, the valve opening value is further compared. If the value is under 80%, the result is `OK`, for values between 80% and 95% the result is `WORKING_BUT` and for values over 95% the result is `ERROR`. The `DeviceSLNode` classes of other agents are automatically evaluating the system state as being `OK`. The goal of implemented diagnose functionality is not to show its complexity but to demonstrate functionality of the information access module.

6.2.3 Running and results

At the beginning of the demonstration, the manual control for all water valves, temperature set points 30°C and 27°C for higher and lower halves of the water tank were set, see Figure 6.7. Then the FIPA-OS platform with all information agents and the user agent was started. The user agent, referred to as a PersonalAssistant (PA) agent, was manually commanded to send the subscription message to the process agent. All sent and received messages by/to the PA agent are in Figure 6.8.

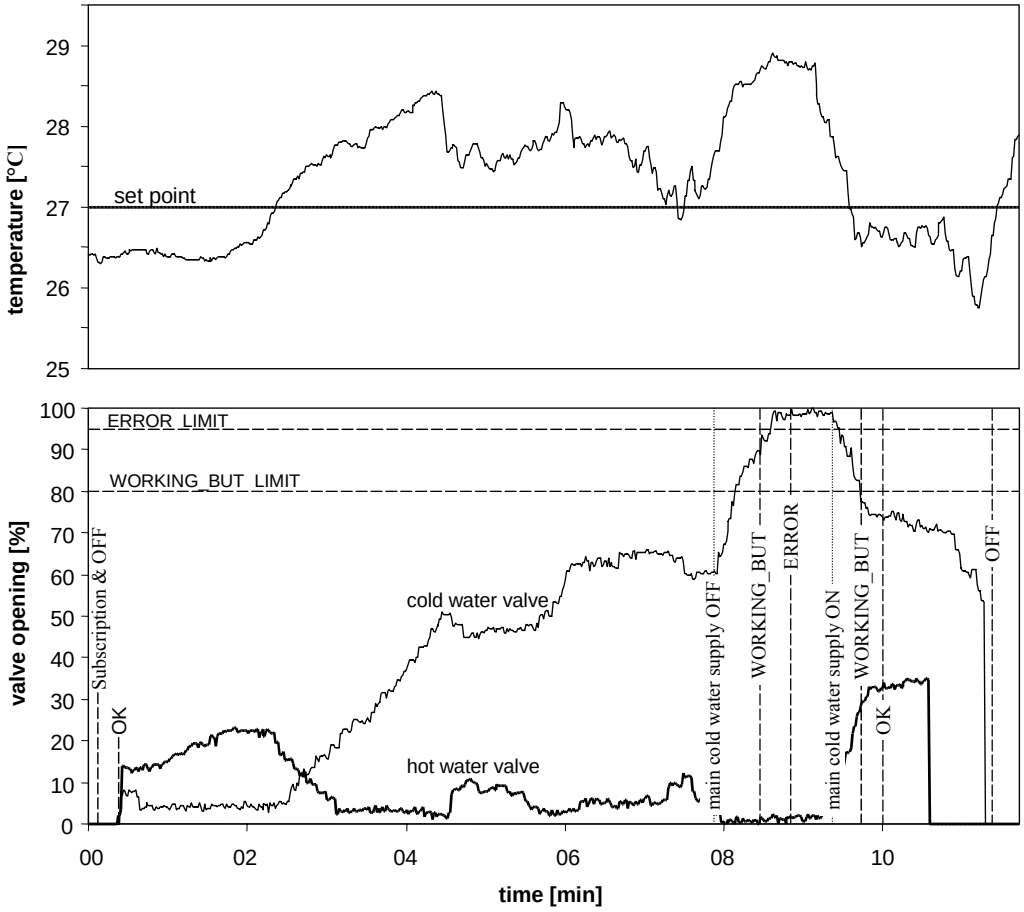

Figure 6.7. Measured data from the demonstration scenario. In the top chart, there is a development of temperature in the lower half of the water tank. In the bottom chart, there are openings of the hot and the cold-water valves with marks when the PA agent received messages from the process agent and when the main cold water supply was switched on and off.

In response to the subscription message, all information agents accepted the subscription and returned the inform reply. The `DeviceSLNode` class of the Level2

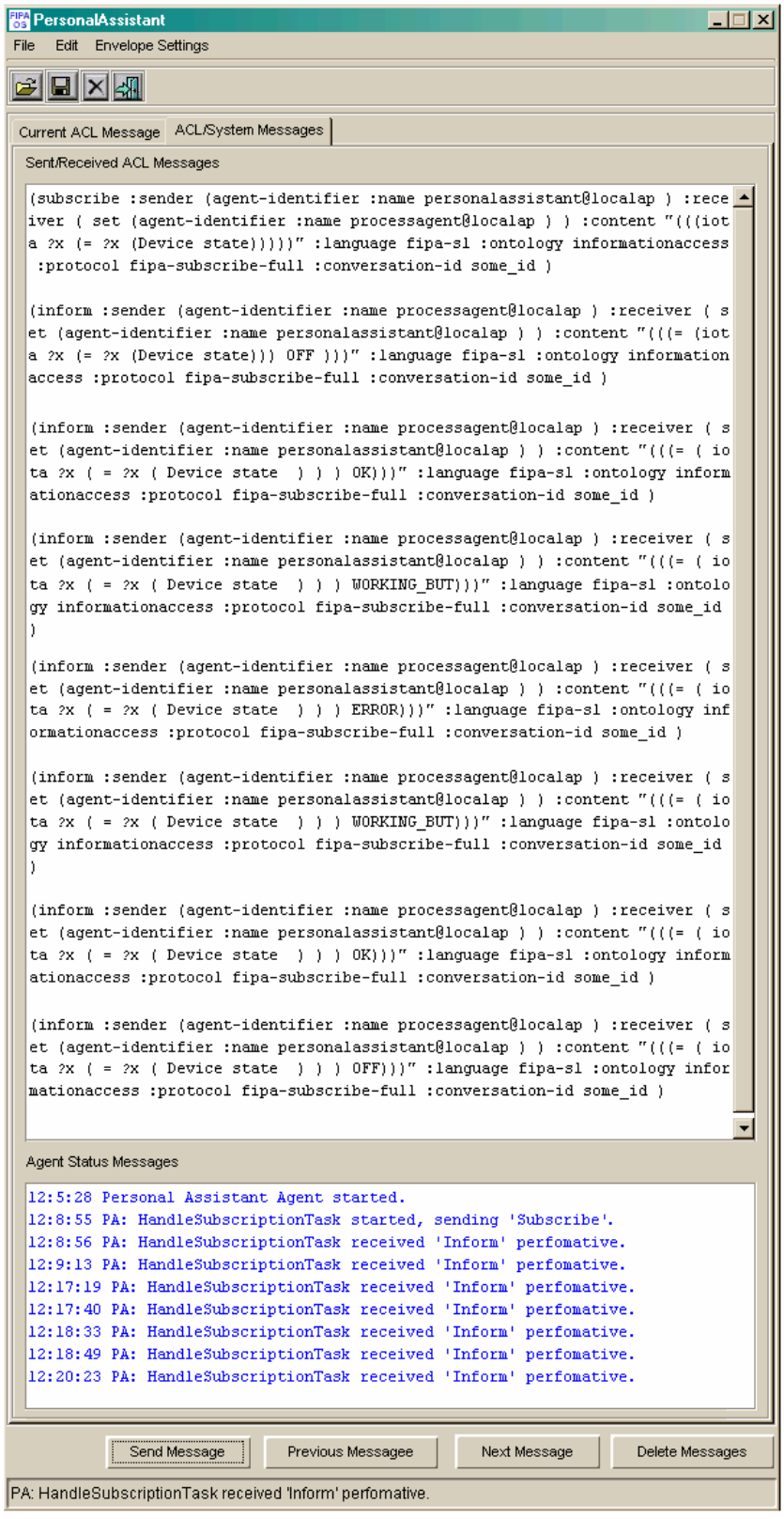

Figure 6.8. The GUI of the PersonalAssistant agent with all sent and received messages.

agent detected that the cold-water valve was in the manual regime and therefore the result of diagnosis was `OFF`, see Figure 6.7. A while later, all water valves were manually switched into the automatic regime. This change was detected by the Level2 agent and the inform message with result `OK` was sent to the PA. From this time, the temperature in the lower part of the water tank was regulated to 27°C with help of a PID controller. At marked time in Figure 6.7 the main cold water supply to the whole system was manually switched off. Incoming hot water caused increase of temperature above its set point. The PID controller started to open the cold-water valve and to close the hot-water valve. When opening of the cold-water valve reached 80% and 95%, the Level2 agent sent the inform messages with `WORKING_BUT` and `ERROR` results respectively. The main water supply of cold water was again manually switched on after while. This lead to decrease of temperature under both limit values and therefore `WORKING_BUT` and `OK` inform messages were sent. The regulation process was renewed and all water valves were manually switched back into the manual regime. The Level2 agent had noticed this change by sending the inform message with the `OFF` result.

7 Conclusions

7.1 Summary

So far, information agents were mainly studied in connection with information access in the Internet. The application of information agents in process automation is rather new, and up to now, almost no research was given to this topic. Therefore, the goal of this thesis is to give a basic analysis of this subject.

First, information access in process automation is discussed in this thesis. The concept and requirements on information access in general and then specifically in process automation along with the OPC Specifications are presented. Second, a brief introduction to agent technology and to agents representing autonomous, communicating and perceiving entities is provided. Third, the idea of agent approach is applied to information access in process automation. Last, in the practical part of this thesis, attention is given to design, implementation and testing of an information access module. The module is based on the FIPAPC agent architecture and it is utilizing an agent message interpreting mechanism developed in this thesis. The proposed architecture of message processing is independent on an agent content language and it is configurable via XML files. Functionality of this mechanism was tested in the demonstration scenario for which the FIPA SL content language was chosen. In the scenario, five information agents using the information access module were monitoring a laboratory physical system. Agents were organized in a hierarchical structure that corresponded to the structure of the physical system. During the test, information agents were sent a message from an external agent requesting a notification about changes of the system running state. Later, the operation of the system control process was disrupted by manually cutting off the main supply of cold water used for regulating the temperature in the system. This interference forced the control mechanism to increase opening of a cold valve to its critical limit. This situation was properly observed by the information agents and the external agent was notified about this event. Besides the information access module, the FIPAPCAgent class was designed. This class allows to assign different modules to an agent by configuring the agent's profile and automatic starting of these modules at the agent's start up.

7.2 Conclusions

In the theoretical part of the thesis, applicability of agent technology for information access in process automation was analyzed according to the general requirements. The results are given in the next paragraphs.

Information systems in process automation have to be very reliable especially when they are used as a part of control systems. Information agents are software units whose reliability depends on the faultless program code and on the environment where

this code is running. Traditional information access technologies are in practice for several years and during this time a lot of experience with their implementation was gained and a number of program faults were removed. In contrast, information agents represent new approach that needs to be well tested before installation in process automation. After surpassing problems connected with introducing new technology, information agents are expected to be more reliable than the legacy systems because of their independence on each other. A failure of one agent does not threaten the whole information system.

Information agents can increase accuracy of returned information by preprocessing it and they can tune information closer to the users needs. Agents can validate information and provide filtering services for example by selecting only the essential part and reducing thus the information overflow. On the other hand, agent message processing, which includes message parsing and evaluation is slower than classical network function calls with predefined parameter types. Also messages are sent asynchronously at unpredictable moments. Therefore, information agents are not suitable to be a part of the system where fast responses are crucial for trouble free running of the system.

In agent information systems, information can be expressed in any agent language that is supported by agents and it can be changed at runtime. Moreover, agent communication languages are universal and extendible the same way as human languages are. Therefore, they are ideal tool supporting automatic information system adaptability, flexibility and scalability. Agent languages are not bounded to any programming language or operating system.

In the practical part of the thesis, the information access module was implemented. The FIPA-Subscription protocol provided by the FIPA-OS agent platform was enhanced according to the FIPA standard. The original FIPA-Subscription protocol did not support sending of the cancel messages needed to abandon an establish subscriptions.

Implementing of a mechanism that is able to process and evaluate any kind of message in any kind of agent language is extremely difficult. In this thesis, implementation of ontologies through Java classes was chosen. This approach is suitable for interpreting simple agent messages but for more complicated messages also implementation becomes complex and requires changes in already written code. A general algorithm for message evaluation using the approach where knowledge is hard coded is probably impossible to design. On the other hand, this approach does not require any ontology language and is many times easier to implement for modules with limited functionality.

Running of the demonstration scenario first required a restart of the operating and control systems. Before the restart, agents were not able to connect to the physical demonstration system. After the restart, agents were monitoring the system and reacting to the changes in the system as it was expected. The result is shown in Figure 6.7 and in Figure 6.8. Agents reacted to the changing state of the laboratory system properly by sending corresponding messages. Agents were able to detect indirectly a failure in the control system. As a problem can be regarded the slow reaction of agents to the changes in the system. The reasons for this were first, the high load of the computer where agent platform along with monitoring software were running. And second, agents were periodically monitoring the system in constant intervals of 5 seconds.

Information agents have many interesting features, which are definitely demanded in process automation. Information agents enable distribution of intelligence over information systems. They can adapt to changes in system architecture and system functionality, they cooperate, learn, etc. The client application represented by the user agent does not need to know any technical details about the system. When the system structure is updated along with the corresponding information agent, the client remains the same. Nevertheless, to be fully used, information agents require continuous extensive research and mainly many tests need to be done to achieve high reliability required in process automation. The designed information module still needs a lot of improvements. To create perfect information access support for agents was not the aim of this thesis. The goal was to discuss and demonstrate agent based information access and for these purposes the information access module was implemented.

7.3 Future work

As for the information access module and other modules in the FIPAPC project, functionality for configuring performatives, which are supported by an agent, in the agent's profile would be beneficial. This would enable easier update of already written code and it would increase modularity of the whole project.

Another testing of the agent message evaluation mechanism is needed. Currently, only functionality required by the demonstration scenario is implemented and therefore other and more complex messages are necessary for studying the suitability and use of this mechanism. In the next demonstration scenario, attention could be paid to using the hierarchical structure of information agents. That is, agents should be able to process information from their sub-agents according to information from other sub-agents and their own information. This approach requires the use of deliberate agents in contrast to reactive agents used in the demonstration scenario.

In the demonstration scenario, the size of intervals in what system data are evaluated should be adjusted dynamically at runtime according to the magnitudes of changes of system variable values.

Attention should be also given to such agent architecture that is independent on ontologies and messages used in agent communication. Currently, knowledge is hard-coded and its modification or extension requires changes in code and its recompilation. To enable agent knowledge independence, standards for knowledge manipulation and storing languages and tools for working with these standards are needed.

As it was mentioned previously, agent messages are sent asynchronously in unpredictable moments. This characteristic could limit the use of information agents in process automation. Therefore a study of this characteristic is needed and research of implementation of synchronous message sending mechanism is necessary.

8 References

[AgentBuilder, 2003] *Why, When, and Where to Use Software Agents.* URL: www.agentbuilder.com/Documentation/whyAgents.html [visited 3 January 2003]

[Angelfire, 2003] *Dynamic Data Exchange (DDE) and NetDDE FAQ.* URL: http://www.angelfire.com/biz/rhaminisys/ddeinfo.html#DDECOM [visited 10 March 2003]

[Appelqvist 2000] Appelqvist P., *Mechatronics Design of A Robot Society – A Case Study of Minimalist Underwater Robot For Distributed Perception and Task Execution*, Doctoral Thesis, Automation Technology Laboratory, Helsinki University of Technology, Finland, 2000

[Appelqvist et al., 2002] Appelqvist P., Seilonen I., Vainio M., Halme A., Koskinen K., Heterogeneous Agents Cooperating: Robots, Field Devices, and Process Automation Integrated with Agent Technology, in *Distributed Autonomous Robotic Systems 5 (DARS 2002)*, Asama H., Arai T., Fukuda T., Hasegawa T. (Eds.), Springer-Verlag, Tokyo, Japan, pp. 350-359, 2002.

[Babbage simmel, 2003] *Business Intelligence -Information Agent*. URL: http://www.babsim.com/consulting/businessintelligence_infoagent.htm [visited 3 January 2003]

[Bailin and Truszkowski, 2001] Bailin S. C. and Truszkowski W., Ontology Negotiation as a Basis for Opportunistic Cooperation between Intelligent Information Agents. In *Proceedings of the 5th International Workshop, CIA 2001*, pages 223-228, Modena, Italy, 2001.

[Barbuceanu and Fox, 1994] Barbuceanu M. and Fox M. S.**,** The Information Agent: An Infrastructure Agent Supporting Collaborative Enterprise Architectures. In *Proceedings of the Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 112-117, Morgantown, West Virginia, USA, 1994.

[Ben-Ami and Shehory, 2002] Ben-Ami D. and Shehory O., Evaluation of Distributed and Centralized Agent Location Mechanisms. In *Proceedings of the 6th International Workshop, CIA 2002*, pages 264-278, Universidad de Rey Juan Carlos in Madrid, Spain, 2002.

[Castelfranchi, 2001] Castelfranchi C., Information Agents: The Social Nature of Information and the Role of Trust. In *Proceedings of the 5th International Workshop, CIA 2001*, pages 208-210, Modena, Italy, 2001.

[Chakraborty, 2003] Chakraborty S. *Agent based approach to fault recovery in a process automation system.* Master's thesis, Department of Automation and System Technology, Helsinki University of Technology, Espoo, Finland, February 2003.

[Decker et al., 1997] Decker K., Pannu A., Sycara K. and Williamson M.**,** Designing Behaviors for Information Agents. In *Proceedings of 1st International Conference on Autonomous Agents,* pages 404-413, Marina del Rey, California, USA, February 1997.

[Emorphia Ltd., 2001] Emorphia Ltd., *FIPA-OS Developers Guide*, 2 February 2001

[Fensel, 2002] Fensel D. A. *Premises and Challenges of Research and Development in Information Agent Technology in Europe*. URL: www.cs.vu.nl/~dieter/wgal/roadmap.html [visited 19 November 2002]

[Ferber, 1999] Ferber J. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, Pearson Education Limited, 1999, ISBN 0-201-36048-9.

[Finin et al., 1994] Finin T., Fritzson R., McKay D. and McEntire R., KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456-463, Gaithersburg, MD, USA, 1994.

[Finin et al., 2001] Finin T., Joshi A., Kagal L., Ratsimore O., Korolev V. and Chen H., Information Agents for Mobile and Embedded Devices. In *Proceedings of the 5th International Workshop, CIA 2001*, pages 264-286, Modena, Italy, 2001.

[FIPA, 2002] FIPA. http://www.fipa.org [2002]

[FIPA, 2002a] *FIPA Network Management and Provisioning Specification.* URL: http://www.fipa.org/specs/fipa00082/XC00082B.html [visited 10 October 2002]

[FIPA, 2002b] *FIPA ACL Message Structure Specification*. URL: http://www.fipa.org/specs/fipa00061/XC00061E.html [visited 23 October 2002]

[FIPA, 2002c] *FIPA SL Content Language Specification*. URL: http://www.fipa.org/specs/fipa00008/XC00008H.html [visited 31 October 2002]

[FIPA, 2002d] *FIPA RDF Content Language Specification*. URL: http://www.fipa.org/specs/fipa00011/XC00011B.html [visited 1 November 2002]

[FIPA, 2002e] *FIPA KIF Content Language Specification*. URL: http://www.fipa.org/specs/fipa00010/XC00010B.html [visited 2 November 2002]

[FIPA, 2002f] *FIPA CCL Content Language Specification*. URL: http://www.fipa.org/specs/fipa00009/XC00009B.html [visited 2 November 2002]

[FIPA, 2002g] *FIPA Ontology Service Specification*. URL: http://www.fipa.org/specs/fipa00086/XC00086D.html [visited 6 November 2002]

[FIPAOS, 2003] *FIPA-OS*. URL: http://fipa-os.sourceforge.net/index.htm [visited 26 May 2003]

[Fortu, 2002] Fortu T. *Enterprise Resource Planning - Integration with Automation Systems.* Master's thesis, Department of Automation and System Technology, Helsinki University of Technology, Espoo, Finland, 10 June 2002.

[Gamma et al., 1994] Gamma E., Helm R., Johnson R. and Vlissides J., *Design Patterns-Elements of Reusable Object-Oriented Software.* Addison-Wesley, Pearson Education Limited, 1994, ISBN 0-201-63361-2.

[Gomez et al., 2001] Gomez M., Abasolo C. and Plaza E., Domain-Independent Ontologies for Cooperative Information Agents. In *Proceedings of the 5th International Workshop, CIA 2001*, pages 118-129, Modena, Italy, 2001.

[Haus, 2003] Haus K.T. *OPC XML-DA Introduction.* Technosoftware Inc., February 2003.

[Hayzelden and Bourne, 2001] Hayzelden A. L. G. and Bourne R. A., *Agent technology for communication infrastructures*, John Wiley & Sons Ltd, 2001, ISBN 0-471-49815-7.

[Jennings, 1999] Jennings N. R., On agent based software engineering. *Artificial Intelligence*, 117, pages 277-296, Elsevier Science B.V, 1999.

[Karhela and Weiss, 1999] Karhela T. and Weiss R., Process Simulation Coordination Using Software Component Technology. In *PC-based automation systems and applications*, Information and computer systems in automation, Helsinki University of Technology, Espoo, Finland, 1999.

[Klusch, 2003] Klusch M., *CIA-2003 Workshop*. URL: http://www.dfki.de/~klusch/cia2003.html [visited 23 March 2003]

[Labrou et al., 1999] Labrou Y., Finin T. and Peng Y., Agent Communication Languages: The Current Landscape. In *Intelligent Systems*, 14(2), pages 45-52, IEEE Computer Society, 1999.

[Laukkanen et al., 2002] Laukkanen M., Helin H. and Laamanen H., Tourists on the Move. In *Proceedings of the 6th International Workshop, CIA 2002*, pages 36-50, Universidad de Rey Juan Carlos, Madrid, Spain, 2002.

[Luck et al., 2002] Luck M., McBurney P., Preist C. and Guilfoyle C. *Agent Technology Roadmap.* AgentLink, Southampton, United Kingdom, October 2002.

[Mangina, 2002] Mangina E. *Review of Software Products for Multi-Agent Systems.* Applied Intelligence (UK) Ltd for AgentLink, www.AgentLink.org, June 2002.

[MathWizards, 2003] *MathViews - Dynamic Data Exchange (DDE)*. URL: http://www.mathwizards.com/techinfo/man/mathv06.htm [visited 10 March 2003]

[Maturana et al., 2002] Maturana F., Staron R., Tichy P. and Slechta P. Using Dynamically Created Decision-Making Organizations (Holarchies) to Plan, Commit, and Execute Control Tasks in a Chilled Water System. In *Proceedings of the 3rd International Workshop on Industrial Applications of Holonic and Multi-Agent systems*, pages 613-622, Aix en Provence, France, 2 September 2002.

[McEntire et al., 1999] McEntire R. and collective. An Evaluation of Ontology Exchange Languages for Bioinformatics. In *Proceedings of ISMB*, pages 239-250, 1 August 1999.

[MSDN, 1992] *Supporting the Clipboard, DDE, and OLE in Applications*. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndynex/html/msdn_ddeole.asp [visited 27 March 2003]
[Nodine et al., 1999] Nodine M. et al., Active Information Gathering in InfoSleuth. In *Proceedings of the 2nd International Symposium on Cooperative Database Systems for Advanced Applications* (CODAS), pages 15-26, Wollongong, Australia, 1999.

[OPC AE, 2002] OPC Foundation, *OPC Alarms and Events Custom Interface Standard, Version 1.1*, www.opcfoundation.org, 2 October 2002.

[OPC Batch, 2001] OPC Foundation, *OPC Batch Custom Interface Specification, Version 2*, www.opcfoundation.org, 19 July 2001.

[OPC Common, 1998] OPC Foundation, *OPC Common Definitions and Interfaces Version 1.0*, www.opcfoundation.org, 27 October 1998.

[OPC DA, 2002] OPC Foundation, *OPC Data Access Custom Interface Standard, Version 2.05A*, www.opcfoundation.org, 28 July 2002.

[OPC DX Press Release, 2002] OPC Foundation, *OPC DX Press Release,* www.opcfoundation.org, 15 March 2002.

[OPC DX Vision, 2002] OPC Foundation, *OPC Data eXchange Vision 2002,* www.opcfoundation.org, 2002.

[OPC HDA, 2001] OPC Foundation, *OPC Historical Data Access Custom Interface Standard, Version 1.1,* www.opcfoundation.org, 26 January 2001.

[OPC Overview, 1998] OPC Foundation, *OPC Overview Version 1.0*, www.opcfoundation.org, 27 October 1998.

[OPC Overview, 2002] *OPC Technical Overview*. URL: http://www.opcfoundation.org/01_about/OPCOverview.pdf [visited 8 November 2002]

[OPC Security, 2000] OPC Foundation, *OPC Security Custom Interface, Version 1.0,* www.opcfoundation.org, 17 October 2000.

[Parunak, 1998] Parunak H. V. D. *Practical and Industrial Applications of Agent Based Systems.* Environmental Research Institute of Michigan (ERIM), 1998.

[Pirttioja, 2001] Pirttioja T. *FIPA-OS as an example of agent system implementation.* Automation and Systems Department, Helsinki University of Technology, Finland, 2001.

[Pirttioja, 2002] Pirttioja T. *Agent-Augmented Process Automation System,* Master's Thesis. Automation and Systems Department, Helsinki University of Technology, Finland, 2002.

[Poslad et al., 2001] Poslad S. Standardizing Agent Interoperability: The FIPA. In *Proceedings of Advance Course on Advance Intelligence*, Prague, The Czech Republic, 2001.

[Seilonen et al., 2001] Seilonen I., Nurmilaakso J. M., Jacobsson S., Kettunen J., Kuhakoski K. *Experiences from the Development of an XML/XSLT based Integration Server for a Virtual Enterprise Type Co-Operation.* VTT Automation, Finland, 2001.

[Seilonen et al., 2002a] Seilonen I., Pirttioja T., Appelqvist P. Agent technology and process automation. *10th Finnish Artificial Intelligence Conference (STeP 2002)*, Oulu, Finland, 16-17 December, 2002.

[Seilonen et al., 2002b] Seilonen I., Appelqvist P., Halme A., Koskinen K. Agent-based approach to fault-tolerance in process automation systems. Submitted to the *3rd International Symposium on Robotics and Automation (ISRA 2002)*, Toluca, Edo. de Mexico, Mexico, 1-4 September, 2002.

[Seilonen et al., 2002c] Seilonen I., Appelqvist P., Vainio M., Halme A., Koskinen K. A concept of an Agent-Augmented Process Automation System. *17th IEEE International Symposium on Intelligent Control (ISIC'02)*, Vancouver, Canada, 27-30 October, 2002.

[Seilonen et al., 2003] Seilonen I., Pirttioja T., Appelqvist P., Halme A., and Koskinen K., Distributed Planning Agents for Intelligent Process Automation. Accepted to the *5th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2003)*, Kobe, Japan, 16-20 July 2003.

[TALtech, 2003] *Understanding Dynamic Data Exchange (DDE)*. URL: http://www.taltech.com/TALtech_web/support/dde_sw/ddeunder.htm [visited 10 March 2003]

[Young et al, 1997] Young P., Johnson R. and Wolfe K.**,** *Agent Technology: Application of Agent Technology*. CSE 190 Internet Technologies, 19-21 May 1997.

9 Appendix 1 FIPA SL content language grammar

The following text is a part of the FIPA SL Content Language Specification. The entire specification with explanation of individual terms is in [FIPA, 2002c].

```
Content              = "(" ContentExpression+ ")".

ContentExpression    = IdentifyingExpression
                     | ActionExpression
                     | Proposition.

Proposition          = Wff.

Wff                  = AtomicFormula
                     | "(" UnaryLogicalOp Wff ")"
                     | "(" BinaryLogicalOp Wff Wff ")"
                     | "(" Quantifier Variable Wff ")"
                     | "(" ModalOp Agent Wff ")"
                     | "(" ActionOp ActionExpression ")"
                     | "(" ActionOp ActionExpression Wff ")".

UnaryLogicalOp       = "not".

BinaryLogicalOp      = "and"
                     | "or"
                     | "implies"
                     | "equiv".

AtomicFormula        = PropositionSymbol
                     | "(" BinaryTermOp Term Term ")"
                     | "(" PredicateSymbol Term+ ")"
                     | "true"
                     | "false".

BinaryTermOp         = "="
                     | "\="
                     | ">"
                     | ">="
                     | "<"
                     | "=<"
                     | "member"
                     | "contains"
                     | "result".

Quantifier           = "forall"
                     | "exists".

ModalOp              = "B"
                     | "U"
                     | "PG"
                     | "I".

ActionOp             = "feasible"
                     | "done".

Term                 = Variable
                     | FunctionalTerm
                     | ActionExpression
                     | IdentifyingExpression
                     | Constant
```

```
                      | Sequence
                      | Set.

IdentifyingExpression = "(" ReferentialOperator Term Wff ")".

ReferentialOperator   = "iota"
                      | "any"
                      | "all".

FunctionalTerm        = "(" "cons" Term Term ")"
                      | "(" "first" Term ")"
                      | "(" "rest" Term ")"
                      | "(" "nth" Term Term ")"
                      | "(" "append" Term Term ")"
                      | "(" "union" Term Term ")"
                      | "(" "intersection" Term Term ")"
                      | "(" "difference" Term Term ")"
                      | "(" ArithmeticOp Term Term ")"
                      | "(" FunctionSymbol Term* ")"
                      | "(" FunctionSymbol Parameter* ")".

Constant              = NumericalConstant
                      | String
                      | DateTime.

NumericalConstant     = Integer
                      | Float.

Variable              = VariableIdentifier.

ActionExpression      = "(" "action" Agent Term ")"
                      | "(" "|" ActionExpression ActionExpression ")"
                      | "(" ";" ActionExpression ActionExpression ")".

PropositionSymbol     = String.

PredicateSymbol       = String.

FunctionSymbol        = String.

Agent                 = Term.

Sequence              = "(" "sequence" Term* ")".

Set                   = "(" "set" Term* ")".

Parameter             = ParameterName ParameterValue.

ParameterValue        = Term.

ArithmeticOp          = "+"
                      | "-"
                      | "*"
                      | "/"
                      | "%".
```

10 Appendix 2 Contents of the enclosed compact disc

A part of this Master's thesis is an enclosed compact disc that contains the following files and directory:

`read_me.txt`	file	Information about the content of the CD.
`thesis.doc`	file	Electronic form of this Master's thesis text.
`documentation.doc`	file	Documentation to Java source code.
`InfoAgent`	directory	Configuration files and Java source code of the information access module and the demonstration scenario.

The printout of the `read_me.txt` file:

This CD is a part of the *Information agents in process automation* Master's thesis. This thesis was written at Helsinki University of Technology - Department of Automation and Systems Technology in Automation Technology Laboratory by Milan Fajt in 2002/2003. On the CD, there is a file `read_me.txt` containing this text. Further, there is a `thesis.doc` file with electronic version of the thesis text, `documentation.doc` file with documentation to Java source code and an `InfoAgent` directory.

The `InfoAgent` directory contains source code to Java classes that are a part of the information access module and the demonstration scenario both referred from the thesis. Further, there are configuration files and examples of agent messages in the directory. To run the demonstration scenario, installation of the FIPA-OS version 2.1.0 and Sun Microsystems's Java Virtual Machine version 1.3.1 are needed. Java classes have to be first compiled into Java byte code and installed according to FIPA-OS documentation. In the case, that agents are not going to be connected to the real physical system, code in the `DeviceSLNode` class of the lower level agent providing this connection has to be removed.

Milan Fajt, Espoo, June 18, 2003