FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Visualization of a Multi-Agent System

MAGISTERARBEIT

zur Erlangung des akademischen Grades

Magister

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Matthias Rodler
Matrikelnummer 0425692

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ. Prof. Dipl.-Ing. Dr. Georg Schitter
Mitwirkung: Dipl.-Ing. Dr. Munir Merdan

Wien, 29.07.2013

(Unterschrift Matthias Rodler) (Unterschrift Betreuung)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Matthias Rodler
Oberlisstraße 133/2, 8232 Grafendorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Matthias Rodler)

Abstract

Current batch process systems have a limited capability concerning agile adaptation in a dynamic environment. Distributed intelligent control systems based on agent technologies are seen as a promising approach to handle the dynamics in large complex systems. However, although seen as a promising approach, its wide application in industry is still missing. Lack of trust in the idea of delegating tasks to autonomous agents is recognized as one of the main reasons for this. This master thesis presents a visualization tool for an agent-based batch process system able to offer the human user the complete overview in the agents activities as well as a possibility to supervise its actions through an adequate Human Machine Interface. The visualization is also compatible with the standard ISA-88. The system is currently being tested and evaluated in the Odo Struger Laboratory at the Automation and Control Institute.

Kurzfassung

Zurzeit werden Batch-Prozesse sehr häufig in der chemischen und pharmazeutischen Industrie eingesetzt. Durch die festgelegten Vorgaben im Ablauf haben sie aber nur sehr eingeschränkte Möglichkeiten um sich einem stetig ändernden Umfeld anzupassen. Der Einsatz von autonomen Agenten bietet hier eine neue Möglichkeit um Prozesse bei Veränderungen im Umfeld zu adaptieren und deren Vorgänge zu optimieren. Leider hat sich dieser neue Ansatz noch nicht durchgesetzt. Dies wird vor allem darauf zurückgeführt, dass die Steuerung komplexer Anlagen - durch autonome Agenten - für den Anwender schwer durchschaubar ist und dadurch das nötige Vertrauen in autonome Agenten noch nicht gewonnen werden konnte. Um ein Multi-Agentensystem jedoch verstehen zu können sollte der Anwender zum einen das Verhalten jedes einzelnen Agenten und zum anderen das Verhalten des Gesamtsystems überblicken können. Im Zuge dieser Magisterarbeit wurde eine Visualisierung entwickelt, welche das Verhalten und die Kommunikation eines Multi- Agentensystems für den Anwender verständlich grafisch wiedergibt. Das Institut für Automatisierungs- und Regelungstechnik (ACIN) nutzt Automationsagenten, um die physikalischen Komponenten einer Anlage zu steuern und die Batchprozesse zu optimieren. Ein Automationsagent, das Kernstück ihrer Architektur, agiert selbstständig basierend auf seinem Wissen, indem er Produktionsumfeld und -bedingungen analysiert und einen Schlussfolgerungsprozess auslöst. Diese Anlage wurde als Referenz herangezogen um die Funktionstauglichkeit der Visualisierung zu testen und soll im weiteren gemeinsam mit der Visualisierung dazu dienen, um Studenten Multi-Agentensysteme näher zu bringen.

Acknowledgement

My sincere thanks are given to Dr. Munir Merdan who was a great help during the writing of this thesis by supporting me with his technical knowledge and constructive feedback. I would like to thank Wilfried Lepuschitz who supported me with his technical competence throughout my thesis. I am also very thankful to Prof. Dr. Georg Schitter for the supervision of this thesis.

I want to thank Benjamin Grössing who does a good job at the Odo Struger Laboratory by developing autonomous agents and made my life much easier by producing great lines of code. I also want to mention my friend Rene Rieger who supported me by proofreading this thesis. Furthermore I would like to express my gratitude to Ekkehard Grübl director of the Grübl Automatisierungstechnik GmbH who gave me the opportunity and flexibility to arrange my studies with my work.

I want to thank my family specially my wife Karin who has contributed an invaluable part to this thesis and made it possible to write this master thesis and finish my studies beside a fulltime work. I also want to mention my two lovely children Iris and Atina for their patience and I want to apologize myself for so many hours which I could not spent with them cause of writing this thesis.

Matthias Rodler

Contents

List of Figures

List of Tables

Abbreviations

ACC	Agent Communication Channel
ACIN	Automation and Control Institute
AID	Agent Identifier
AJAX	Asynchronous JavaScript and XML
AMS	Agent Management System
API	Application Programming Interface
BDI	Belief, Desire, Intention
CDDL	Common Development and Distribution License
CSS	Cascading Style Sheets
DF	Directory Facilitator Agent
ECMA	European Computer Manufacturers Association
EEF	Extended Editing Framework
EMF	Entity Modeling Framework
EMF-MT	Entity Modeling Framework - Modeling Transaction
EMFT	Eclipse Modeling Framework Technology
EPL	Eclipse Public License
FIPA	Foundation for Intelligent Physical Agents
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
GPL	General Public License
Graphiti	Graphical Tooling Infrastructure
GUI	Graphical User Interface
GWT	Google Web Toolkit
HLC	High Level Control
HMI	Human Machine Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JADE	Java Agent DEvelopment Framework
LLC	Low Level Control
MAS	Multi-Agent System
MAST	Multi-Agent Simulation Tool
OLE	Object Linking and Embedding
OPC UA	OPC Unified Architecture
OSGI	Open Services Gateway Initiative

OTN	Oracle Technology Network
PLC	Programmable Logic Controller
RCP	Rich Client Platform
SCADA	Supervisory Control And Data Acquisition
SDK	Software Development Kit
SMIL	Synchronized Multimedia Integration Language
SVG	Scalar Vector Graphics
SWT	Standard Widget Toolkit
UI	User Interface
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Introduction

Automation is already well established in the industry. It has reached nearly every area of the market from production to logistics. Most automation facilities use a centralized control architecture at which computations are mainly executed in a central computing unit. However current demands of the market require a high flexibility and adaptability of production lines to be able to cope with fast changing circumstances within the industry. Continuous changes of an inflexible system can cause unexpected high costs within a life cycle of a production line. To avoid this drawback of current automation solutions in future development the introduction of artificial intelligence techniques is seen as promising trend in the process industry [SL07]. In correlation with process automation multi-agent technology is identified as a major tool for developing highly flexible, robust and reconfigurable industrial solutions in this field of application [VM10, La09]. The technology is able to handle dynamics in large complex systems, decentralizes the control, decreases the complexity, raises the flexibility and improves the fault tolerance of a system [JB03a]. Based on these facts, agent technology is suggested for usage in the process domain according to analysis of its advantages and disadvantages as presented in [MP11].

Agents can be seen as autonomous units in a defined environment. They are able to interact and communicate with other agents depending on their responsibilities and duties. Because of their capabilities agents can have a positive impact on the efficiency and reliability of production lines. This can be achieved through an agile failure monitoring and automatic recovery which furthermore shortens the reaction on upcoming disturbances and minimizes the occurrence of system shutdowns respectively extensive downtimes because of a possible faster recovery caused by resource breakdowns [VMLK11]. The integrated operational safety which is part of a well organized multi-agent system can increase the overall value of a production line. Furthermore this can lead to shorter production times and more flexibility at the variation of produced products and their needed processes.

1.1 Problem Outline

Though the first multi-agent systems appeared in the mid-1980s [Syc98b] and agent-based concepts were confirmed as a promising approach and deployed in a number of different applications throughout the last few years, their widespread adoption by industry is still missing. Lack of awareness about the potentials of agent technology [PM08b] and paradigm misunderstanding [CBJC10] due to the lack of real industrial applications, missing trust in the idea of delegating tasks to autonomous agents [Syc98a] as well as concerns regarding the stability, scalability and survivability [HTW04] are identified as the main reason for the underestimation and refusal of multi-agent systems. Besides, the specific nature of software agents, which are designed to be distributed, autonomous, and deliberative, makes it difficult to apply existing software testing, monitoring, and diagnostics techniques to them. For instance, agents operate asynchronously, in parallel and concurrently, which can lead to non-reproducible effects [NPB$^+$11]. It is not ensured that two executions of the systems will result in the same state, even if the same inputs are used. As a consequence, looking for a particular error is difficult, as it is usually impossible to reproduce in a systematic fashion [HD04]. Also the possibility to use a MAS for development of automated systems is hardly known by developers and also not well supported by the hardware manufacturer who usually deliver their own developer tools in conjunction with their devices. These facts complicate the switch from common task oriented programming to an agent based development. To overcome these barriers multi-agent systems need an appropriate representation of their communication and behaviour to represent their proper strength: distribution, collaboration and intelligence [zuk].

The introduction of tools, techniques and methodologies, which ensure easier and more abstract ways of agent system development, modification and management, will lead to a higher rate of acceptance as well as understanding of the concept. In this context it is of vital importance to offer the human user the complete overview of the agent's activities as well as a possibility to supervise their actions using an adequate Human Machine Interfaces (HMI). A user has to be able to follow and understand the current state of the process. This will on one side improve safety and risk grade and on the other side enhance trust in the idea of delegating tasks to autonomous agents. The assurance of security and trust in agents is a significant aspect to be considered in future solutions [PM08b].

Currently the market is filled with a bulk of HMI and Supervisory Control and

Data Acquisition (SCADA) systems. They commonly represent values of variables (called tags) bound into a graphic of a Programmable Logic Controller (PLC). Beside of their good capabilities to bind tags of PLCs and represent them in real time within a nicely animated graphic, they are still not prepared to visualize the communication of agents within a multi-agent system. The reason for their mean interest to develop a connection to popular MAS frameworks and to find feasible ways for visualization can be found in the marginal interest of the industry to use agents to control their production lines. The missing confidence leads to a backslide to old fashioned PLC programming which is the common procedure nowadays.

1.2 Objectives of the Thesis

Visualization is the representation of abstract facts and information in general. The main task is to increase the cognitive abilities of the mind with external mechanisms. Using visualization, software processes can be made visible, creating cognitive images for the viewer that support the understanding [Die03] and through this reduce the complexity [BE96]. The advantage of visualization is that it can enable a fast interpretation of a wide range of information by the human. Visualization systems have to fulfill the following tasks: support for the understanding of large amounts of information; perception of emergent behaviours which are not predictable; fast discovery of errors by means of the visualized data and shorter time and lower cost solutions to problems through timely and intuitive access to visual data.

1.3 Visualization Requirements

A usable and easy to handle visualization of a multi-agent system should be able to fulfill several conditions to achieve a high acceptance in the user's and developers point of view. These prerequisites should be achieved by the application and reached through the conception phase.

Following attributes should be included to fulfill its requirements:

Expandability The application should provide an interface for a simple extension to be prepared for possible future scenarios and to be able to connect to different multi-agent systems through the simple integration of new interfaces.

Integration It should be able to integrate the following approach seamlessly into the current development process or development tools to enlarge the usability and handling of the used MAS at the Odo Struger Laboratory. This requirement also depends on the chosen technology.

Usability It should offer a simple GUI which can be easily handled without the need of much knowledge about the agents' underlying technology.

Autonomy The source code of the agents should not be touched by creating a visualization which requires the ability to gather data from agents without changing the existing system and source code.

Transformation The application should provide a plain possibility to create an XML-based ontology out of a graphical representation from an existing plant. The outcome should fulfill the test plant's requirements and should be the basic ontology-information for the used MAS.

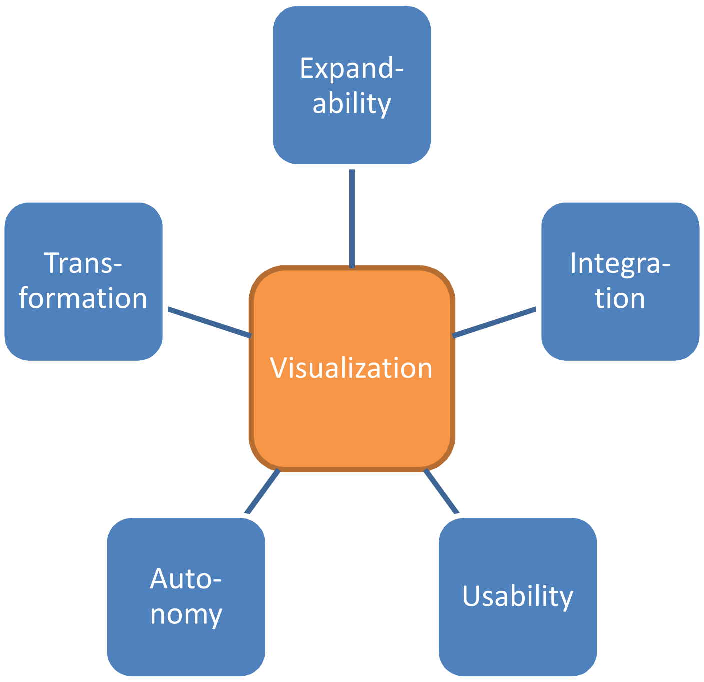


Figure 1.1: The Figure shows the main attributes of a productive and usable/reusable visualization.

By the achievement of the points mentioned above, MAS systems should be able to reach a higher acceptance, a better understanding by users and a reduced initial barrier for students respectively developers to work with multi-agent systems. An important point is the integration into tools which are already used to develop agents. This could simplify the rapid development of MAS for testing purposes.

1.3.1 Tasks

This master thesis deals with the development and implementation of a user friendly and easy to handle graphical user interface which is able to represent the state and behaviour of a multi-agent system. The presented visualization technique should enable a better supervision and understanding of an agent-based batch process with the focus on the usability and short training period for users who are not aware of MAS. The graphical representation of the agents should be based on the ISO standard 14617 and open for customized illustrations. Additionally, features for simpler creation and management of MAS are incorporated in the approach. The Automation and Control Institute (ACIN) at the Vienna University of Technology uses the Java Agent DEvelopment Framework (JADE) to control an experimental plant (see Figure 1.2). It is the aim to gather data through the agent framework itself and the OPC UA interface to visualize the information in a user friendly graphical interface and to perform the proof of concept tests on the existing test plant in the laboratory.

Following tasks will be discussed throughout this master thesis:

- Analysis of existing frameworks concerning the visualization of multi-agent systems and discussing the upcoming web technologies for visualization purposes in the automation industry.
- The design of general architecture for an MAS independent and expandable visualization.
- Analysis of the Java Agent DEvelopment Framework (JADE) [Jada] which is used to automate the test plant assembled at the Odo Struger Laboratory.
- The development or expansion of a visualization framework to accomplish the expected goals.
- The developed framework has to be tested with the existing plant and its agents at the Odo Struger Laboratory.

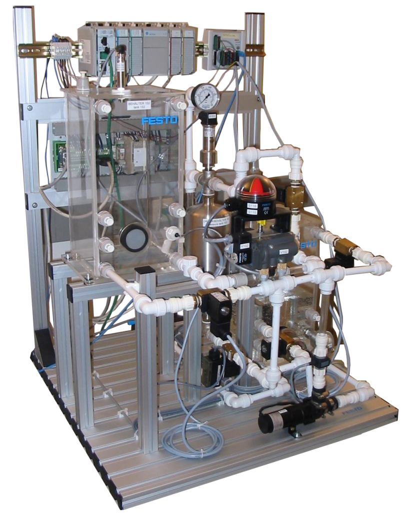


Figure 1.2: Experimental plant in the Odo Struger Laboratory at the Vienna University of Technology.

- Discussion of the test results and possible future work.

1.4 Structure of the Thesis

This thesis is structured as follows:

Chapter 2 will first discuss the term *agent* and different technologies used by multi-agent systems. Afterwards the Chapter will give an overview of current state of the art visualizations of multi-agent systems. Furthermore web based visualization and existing web technologies - their advantages and disadvantages - will be discussed. Finally current available frameworks and visualization tools which are able to offer a graphical representation of the communication of agents that already use web technologies will be mentioned.

A flexible and generic approach of a visualization for MAS is introduced in Chapter 3. Each layer of the presented tier based architecture will be described and an abstract design of a data model and its functionality will be discussed.

Chapter 4 presents the implementation of the introduced approach presented in the previous Chapter. A decision for a suitable technology will be discussed and the integration in an available IDE will be clarified throughout this Chapter.

The tests of the visualization at the plant in the Odo Struger Laboratory and its results will be discussed in Chapter 5. Furthermore a real world use case "student - tutor" will be run through and its results presented.

Chapter 6 summarizes the output of this thesis and gives a brief overview of possible future work based on the discussed results.

2 State of the Art

This Chapter will give a short introduction into agent technologies and common terms in conjunction with MAS followed by an overview of possible visualization tools for multi-agent systems and available options to visualize the exchange of information by the agents within an MAS. The second part of this Chapter will give an overview of the new upcoming HTML5 technology which is also seen as possible base for visualizations of multi-agent systems. For this reason the advantages and disadvantages of HTML5 are discussed in detail.

2.1 Agent Technologies

Ahead of detailed analysis of state of the art visualization tools a brief overview of existing agent technologies will be given throughout this Section. The term agent is widely spread in information technology literature and is not exactly associated with an intelligent software part in automation processes. Wooldridge expressed a suitable definition for autonomous agents:

“An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.” [Wei99]

Wooldridge also mentioned that this definition leaves some place for own interpretation. As this description is kept general the possibilities for the usage of agents also has a wide field of application. The broad spectrum of applications demands for different technologies of agents that will be described in this Section. In general two different approaches are used for agent design the functional and physical decomposition approach [Mad07, SN99].The functional decomposition approach defines agents depending on their tasks (e.g. order, task agent). The physical decomposition approach defines agents in correlation with its counterpart in the physical world (e.g. pump, tank). The choice of an approach always depends on the demands on agents, though functional decomposition allows a more efficient management and limited interaction by different agents, a more modular software and an easier redeployment with less code changes which leads to a drop of costs for software reconfiguration and a higher reusability of developed agents [Wei99].

In general agents can be divided into four main types of agents depending on the used architecture [FBG07, Mer09, Wei99]:

- Reactive agents observe their environment, recognize changes and react corresponding to their objectives. They are commonly used in environments which require real time behaviour. They do not save any historical information or internal states which makes further reaction based on experience impossible.
- Logic-based agents' decision making is based on logical deduction. This represents the traditional approach to create artificially intelligent systems. Its behaviour is based on a symbolic knowledge base which is manipulated by using reasoning mechanisms.
- BDI (Belief, desire, intention) based agents are able to decide which objectives they want to fulfill and how they are going to achieve them. This architecture is seen as most popular agent architecture. It still has a drawback which is the decreasing reliability with increasing complexity that might be a handicap by achieving goals under real time conditions.
- Layered (hybrid) architecture allows the usage of both reactive and deliberative agents. Two types of layering are possible: horizontal and vertical layering. At horizontal layering each layer is connected to the input devices and for each behaviour a new layer has to be implemented. In contrary vertical layering forwards the information from the bottom layer - that is connected to the sensors - to the top layer which produces the output.

2.2 Applications of Agent Technology and Ontologies in Industrial Process Control

The automation in the industrial domain is already well established and a common practice. The leading approach in industry is still the usage of centralized systems but due to the fact of increasing complexity and the restricted fault tolerance of centralized systems, decentralized systems become more appropriate [MM05]. To increase the fault tolerance and reliability decentralized techniques have been researched to reach the desired goals. Multi agent systems are seen as adequate technology to fulfill the required demands [JB03b]. [HYJY10] describes the usage of a MAS based approach to improve the production efficiency for ore grinding. In [MSD+07] an MAS is used as shipboard system to eliminate single point of failures and to achieve a higher robustness of the system. Another reason for the usage of agent technology is the assistance at fault detection and decision making in different field of application [MP11]. [FR09] describes the usage of software agents to optimize the electrical output of an Hydro-generating plant. The above mentioned examples demonstrate the broad field of application in the industry still the distribution of agent technology is at the beginning. An extensive literature review has recently been conducted by Metzger and Polakow with the focus on applications of agent technology in industrial process systems [MP11].

The above mentioned examples illustrate the successful usage of multi-agent systems in real world scenarios. To achieve a proper collaboration of deployed agents a common base of knowledge is needed. An appropriate representation of shared information can be utilized through ontologies. It is seen as a practicable way to share knowledge among agents within an MAS and as information base for decision-making and reasoning [MMH13]. Ontologies can be seen as "specification of a conceptualization" [Gru93] which means "specifying the structure of a domain of knowledge in a generic way that can be read by a computer (formal specification) and presented in a human-readable form (informal specification)" [MEP10].

2.3 Current Visualization Tools

This Section will give an overview of existing visualization applications and tools for multi-agent systems which help to debug, test or visualize an existing MAS system.

Concerning visualization tools following two points must be considered and are important for each user which are [NNLC99]:

- to understand, control and analyze the behaviour and
- debug an existing multi-agent system.

The most currently running visualizations are based on HMI applications. The main part of HMI software are client based applications but the new

HTML5 standard already gains more attention and popularity and is seen as emerging trend for HMI applications [Hec]. Nevertheless even if nowadays HMI products have the ability to visualize smooth and gentle graphical representations (see Figure 2.1) of industrial facilities they are currently not able to handle agent based automation systems.

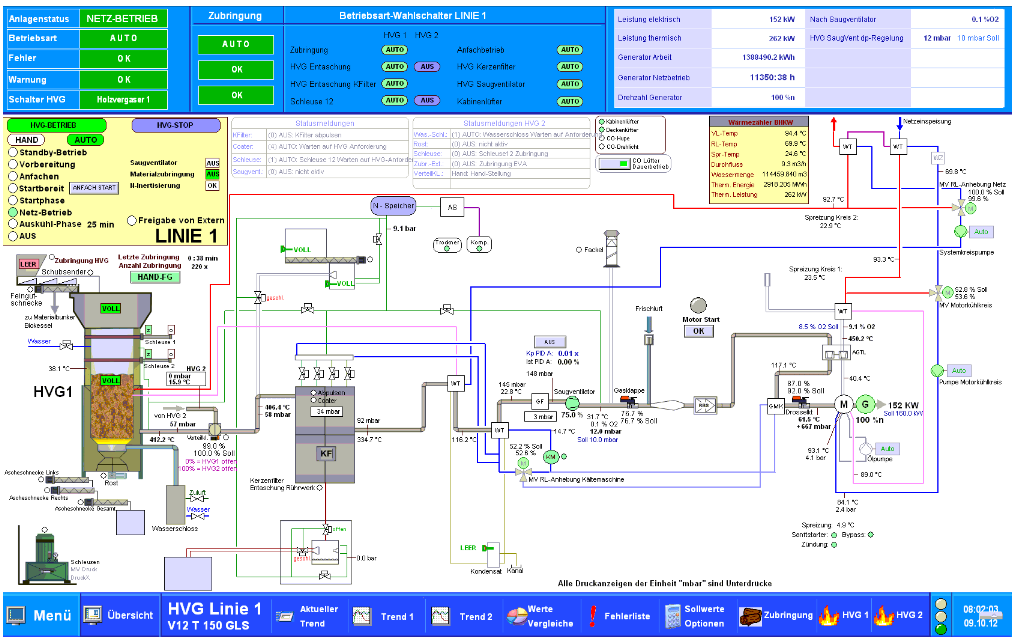


Figure 2.1: A visualization of a wood gasifier boiler. Tag values are mapped into a graphic and represent the state of the equipment.

Their structure is still inflexible and justified to represent the values of tags which are gathered from connected PLCs. This fact makes them unusable for the representation of the information which is exchanged within an MAS. The wide spread of HMI and SCADA systems and their disability to handle agent based information probably also influences the acceptance of multi-agent systems in the industry. Despite the offered graphical options HMI systems will not be discussed in detail due to the lack of functionality with MAS.

2.3.1 MAST

The Multi-Agent Simulation Tool (MAST) is a Java based and often mentioned simulation tool for multi-agent systems. It was developed by the Rockwell

Automation Research Center with the aim to show the potential of agent-based solutions. The focus lies on transportation and routing of products [Jadb] (see Figure 2.2).

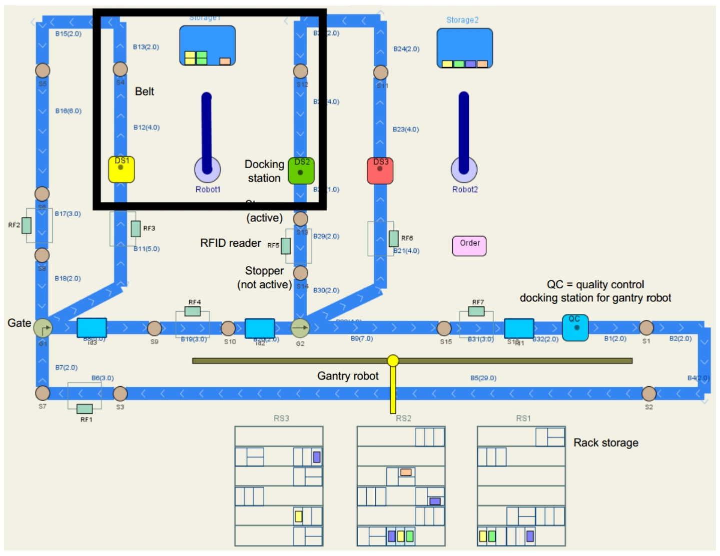


Figure 2.2: Screenshot of the Multi-Agent Simulation Tool (MAST) developed by Rockwell Automation Research Center [MOR].

A great benefit of MAST is the ability to simulate the behaviour of agents and to provide real-life physical control [MR]. The agents used by the MAST tool are based on the FIPA[1] [FIP] compliant JADE framework. The simulation environment is already used in several laboratories (e.g. Odo Struger laboratory at Vienna University of Technology, DIAL laboratory at University of Cambridge [VMK12]) for simulation, testing and the visualization of transportation systems.

The MAST consists of 4 parts [MMW+08]:

- The first part contains the library of JADE based agent classes
- The second part implies the simulation engine which emulates the behaviour of the physical system.
- The third part contains the graphical user interface which provides the visualization of the simulation.

[1]Foundation for Intelligent Physical Agents

- The fourth part contains the control interface which represents the link between the agents and their simulation objects.

Although the architecture of the application is a tier based application it does not seem to be meant for an easier extension of the application. The main purpose for MAST is still to simulate, manage and visualize transportation systems which limits its area of application.

2.3.2 VizScript

VizScript is a visualization tool developed to understand and debug multi-agent systems. The aim of the developers was to create a tool with which other MAS developers are able to create visualizations in a generic way. It allows the creation of graphs and diagrams by using a scripting language. The information which is visualized is gathered from log scripts generated by each agent itself in a specific - for VizScript usable - format. It uses the information from each agent's log to build a knowledge base and in conjunction with specific scripts written in its own scripting language it interprets the data (by pattern matching) to create the graphics for further analysis by the user [JSMS08]. VizScript is an interpreted language and allows very efficient code to create different kind of graphics (Figure 2.3 and 2.4 show two examples) in comparison to Java that needs much more coding effort for the same visualization(see Table 2.1 for detailed information). In addition VizScript supports online and offline mode to view the agents' data in its visualization.

Table 2.1: The effort to create a visualization for collected data decreases dramatically by using VizScript's efficient scripting language for data representation [JSMS08]. The original code is written in Java.

	Lines of Code		Savings
Visualization	Original	VizScript	Factor
Quality View	122	5	24.40
Agent View	190	47	4.04
Probability View	258	18	13.57
Execution View	1214	93	13.05

An existing Eclipse plugin to create and edit VizScript files in Eclipse with the support of syntax coloring, code completion and assistant support [Jin] eases the development of scripts.

Though the application is built up in tiers it is not supposed to be extended for other usage. Another main drawback is the fact that VizScript gathers its

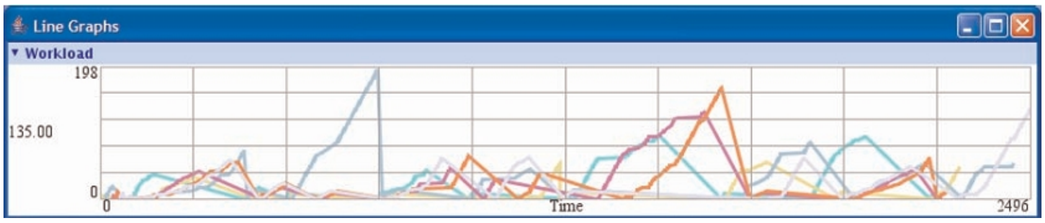
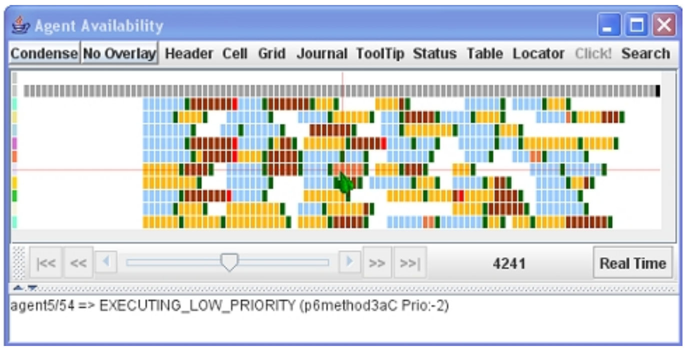


Figure 2.3: This Figure shows the amount of workload for each agent generated by VizScript [JSMS08].



agent5/54 => EXECUTING_LOW_PRIORITY (p6method3aC Prio:-2)

Figure 2.4: A visualized graph created by VizScript. It represents the availability of agents within an MAS [JMSS07].

information from log files which must be generated by every agent itself. This implies a modification of the existing source code which can be a major effort to gather the information that is used by VizScript for interpretation. The implementation of VizScript support should already be considered at design and implementation time of an MAS. This could minimize the effort for involving VizScript for debugging and testing purposes. Although the creation of views is very simple with VizScript scripting language, the provided views are mainly for developers and advanced users who want to debug and analyze an existing system that they are already aware of and which is already equipped with the needed logging required by VizScript.

2.3.3 AgentFly

AgentFly is a simulation system for air traffic developed at the agent technology center at the Czech Technical University. Its main subjects are to be able to run large scale simulations of civilian and unmanned air traffic. It supports advanced flight path planning, decentralized collision avoidance and a graphical representation of the air planes and the environment [age]. The simulation is based on the A-Globe agent platform whose focus lies on real world simulations with a high number of fully autonomous agents. Although this platform supports most features of the FIPA standard it is not fully compliant to the specification. To reach a better system performance features like interoperability where left behind [PvPU06]. The visualization is based on the CrystalSpace open source 3D engine and provides a 2D (see Figure 2.5 (a)) and 3D view of the simulation. Due to performance issues the visualizer is written in C++. It also provides a remote web client (see Figure 2.5 (b)) which is created in Java and allows remote access to view the flight simulation. Nevertheless the application scope is limited to flight simulations and is not applicable to industrial applications.

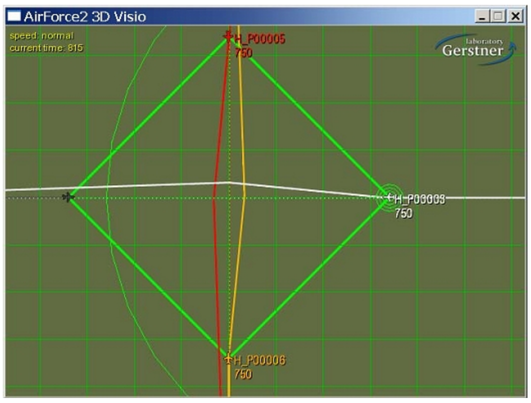
2.4 Web Based Visualization

The current main players at the software market who develop applications for the mass market try to force the development of online software products (e.g. Office365 [off], Google Docs [goo],...). New upcoming technologies as HTML5[2], CSS3[3] and AJAX[4] based frameworks accelerate the development of web based applications. Upcoming trends as cloud services enforce the arising of new supportive tools and frameworks. Global players as Google boost the development of web applications by providing development tools as the Google Web Toolkit (GWT) for developing web applications. Nevertheless HTML5 is not yet approved by the W3C[5] and the implementations of the new standard (also of the older ones) differ in their implementation by commonly used web browsers (Figure 2.6) and complicate the development of new web applications.

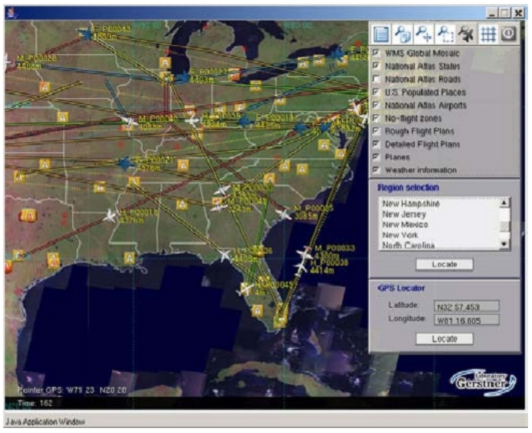
Furthermore the used web browsers are (see Figure 2.7):

- from different vendors mainly from Google (Chrome), Mozilla (Firefox), Microsoft (Internet Explorer), Opera (Opera) and Safari (Safari)
- and by each vendor fragmented in different versions.

[2]Hypertext Markup Language
[3]Cascaded Style Sheet
[4]Asynchronous Javascript and XML
[5]World Wide Web Consortium



(a) AgentFly 2D View



(b) AgentFly remote web access

Figure 2.5: AgentFly provides several ways to visualize a simulation. One way is through local access which allows a 2D (a) and 3D representation of a simulation. The second possibility is an available remote access (b) to the visualization [PvPU06].

Also other proprietary web frameworks as Flash and ActiveX are either bound to a specific platform or not supported by other vendors (e.g. Flash is not supported on IOS[6] [fla]).

Another promising technology in interactive web animations are scalar vector graphics (SVG). It provides its own declarative scripting language named

[6]Operating system of IPhone and IPad

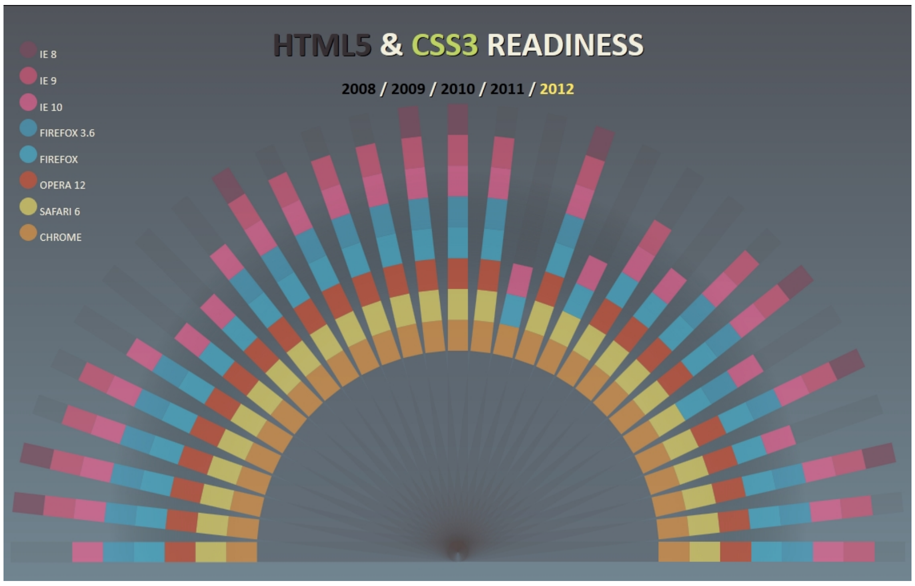


Figure 2.6: Implementation progress of HTML5 and CSS3 standard in the 5 most popular web browsers [Yak].

Synchronized Multimedia Integration Language (SMIL) to perform animations within an SVG graphic. Furthermore it offers the ability to create an interactive drawing. The scripting itself is directly integrated into the SVG XML sheet. In addition Javascript can also be used in combination with SVG graphics. This opportunity enlarges the possible field of application for SVG graphics. Nevertheless SVG is a declarative description of what should be drawn and all commands have to be supported by the web browsers. Also this language - even if the SVG 1.1 specification is approved by the W3C - has to battle with the same problems as the HTML standard (see Figure 2.8 for a graphical representation of implementation lacks concerning the official recommendation of W3C).

However in some papers web based technologies especially HTML5 are seen as possible technology for visualization applications for multi-agent systems in the future [VKJ^{+}11]. Games often act as pioneer for new technologies. Possible opportunities can already be seen in HTML5 games which use a great part of the HTML5 functionality and Javascript. The games are already intended for single player and multiplayer [HTM].

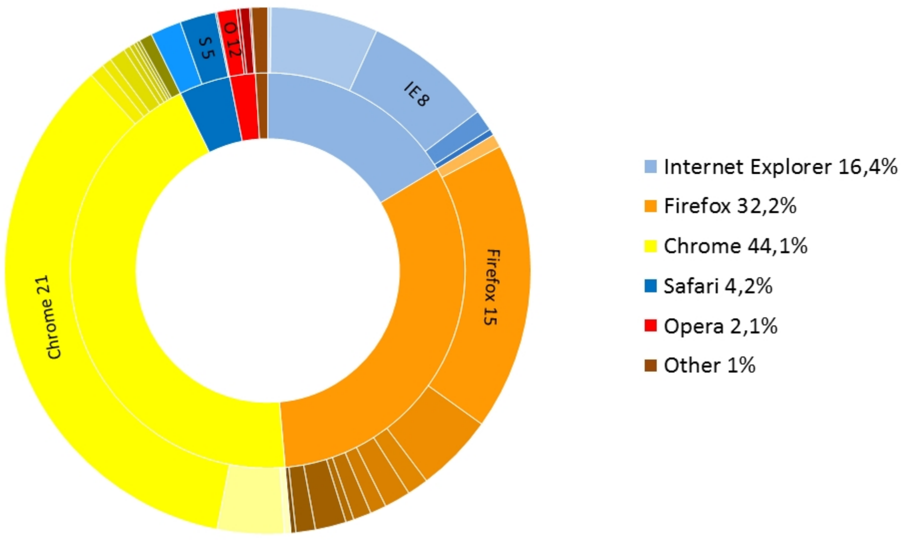


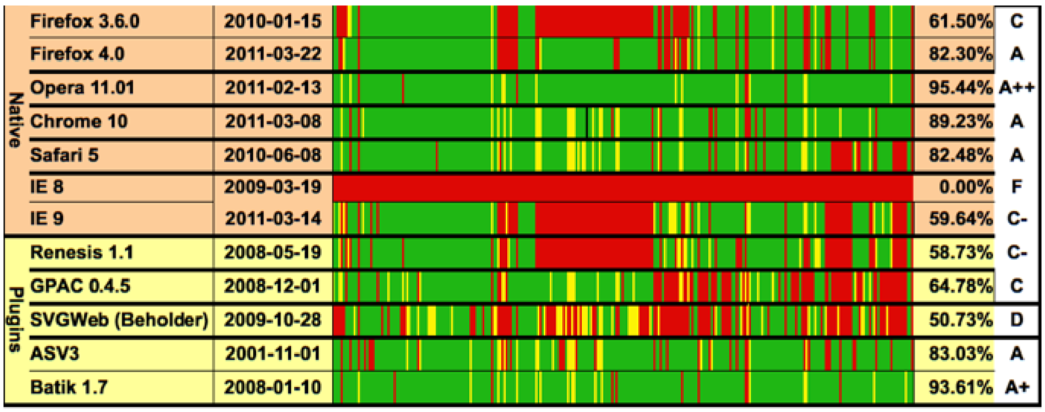
Figure 2.7: The diagram represents the segmentation of the browser usage for Internet consumption at September 2012. The inner circle gives an overview of the different kinds of browsers used, the outer circle represents the fragmentation of different browser versions within one trademark. Darker colors represent older and lighter colors newer versions of a browser. The more often used version is labeled (data from [bro]).

The gaming industry is not the only business in which these technologies are already involved. A few prototypes already exist for the visualization of MAS (or at least their ontologies) which use web technologies:

- [PM08a] introduces an application for monitoring a multi-agent system. The communication is based on web services which offer XML data through the HTTP[7] protocol. It uses AJAX for communication and SVG documents in combination with an embedded ECMAScript[8] for the graphical representation (see Figure 2.9 for a screenshot of the application).

- OntoVis is a web based tool which visualizes ontologies based on HTML5. It's built on Google's GWT and uses HTML5 canvas and SVG for graph-

[7]Hyper Text Transfer Protocol
[8]Client side scripting language for web



Native	Firefox 3.6.0	2010-01-15	61.50%	C
	Firefox 4.0	2011-03-22	82.30%	A
	Opera 11.01	2011-02-13	95.44%	A++
	Chrome 10	2011-03-08	89.23%	A
	Safari 5	2010-06-08	82.48%	A
	IE 8	2009-03-19	0.00%	F
	IE 9	2011-03-14	59.64%	C-
Plugins	Renesis 1.1	2008-05-19	58.73%	C-
	GPAC 0.4.5	2008-12-01	64.78%	C
	SVGWeb (Beholder)	2009-10-28	50.73%	D
	ASV3	2001-11-01	83.03%	A
	Batik 1.7	2008-01-10	93.61%	A+

Figure 2.8: This graphic represents the implementation status dated on the 24 of March 2011 of the SVG 1.1 specification. The first column represents the names and versions of different web browsers. The second column contains the release date of the particular version. The last three columns represent the implementation status of the SVG specification. Even if this standard became an official recommendation of W3C in January 2003 it is still not fully implemented by any web browser. The performed tests are part of an official test suite provided by the W3C [svg].

ical representation [VKJ+11] (Figure 2.10 shows an example of a visualized ontology).

The basement for most dynamic and interactive web applications represent Javascript frameworks. There is already a wide range of frameworks present on the market. They take care about the differences between web browser versions and vendors [KZ12]. Another supportive tool for web development is GWT. It was introduced in 2006 and is currently available in version 2.5 (released on 26th of October 2012) [gwt]. It contains a Cross-Compiler which allows the development of web applications with Java and the compiler translates the code into HTML and Javascript which eases the development of web applications. Furthermore parts of HTML5 are already supported. HTML5 (by using the canvas element) combined with GWT already allows sophisticated drawing applications with less effort [Sta12, Vog].

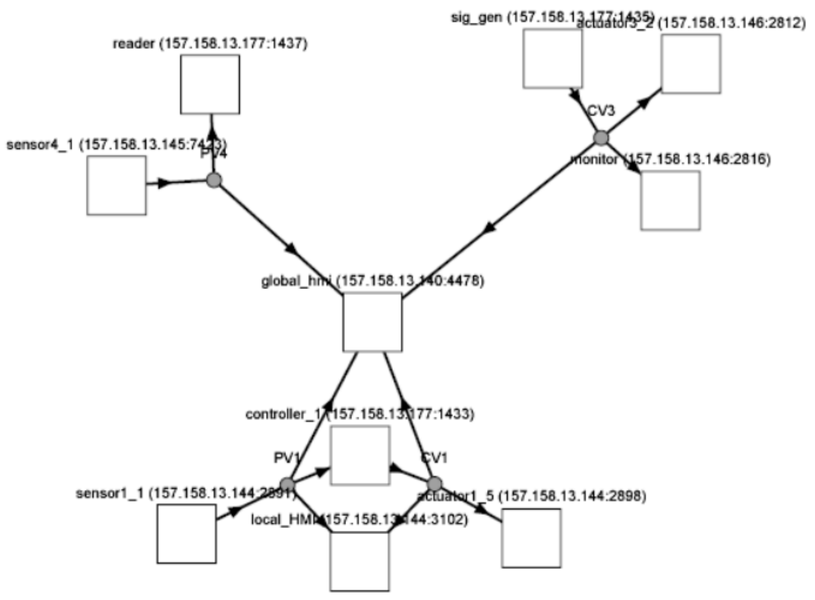

sig_gen (157.158.13.177:1435)
actuator3_2 (157.158.13.146:2812)
reader (157.158.13.177:1437)
CV3
sensor4_1 (157.158.13.145:7423)
PV4
monitor (157.158.13.146:2816)
global_hmi (157.158.13.140:4478)
controller_1 (157.158.13.177:1433)
PV1
CV1
sensor1_1 (157.158.13.144:2891)
actuator1_5 (157.158.13.144:2898)
local_HMI (157.158.13.144:3102)

Figure 2.9: The web browser visualizes the created SVG based interactive dynamic block diagram. The logic is implemented with ECMAScript and its data exchange with the backend is done through AJAX [PM08a].

2.5 Conclusion

The first part of this Chapter has listed used agent technologies and has given an extraction of existing visualization tools for multi-agent systems. The MAST visualization has great abilities to support the development of transportation systems supported by the Jade framework but its area of application is limited due to the fact that its visualization is fitted to this special scope. VizScript with its scripting language provides a wide range of options to analyze and visualize information provided by the agents' log files with less coding effort but more time consumption by execution than a native implementation in Java. Furthermore it provides online and offline visualization of the provided information but the information must be provided by each agent itself which implies a manipulation of the source code to be able to collect the needed information for visualization. This can be a handicap by analyzing existing multi-agent systems due to the fact that subsequently executed code manipulation can be a time intensive process. The second part of this Chapter has given a brief overview of the possibilities of the new HTML5 standard

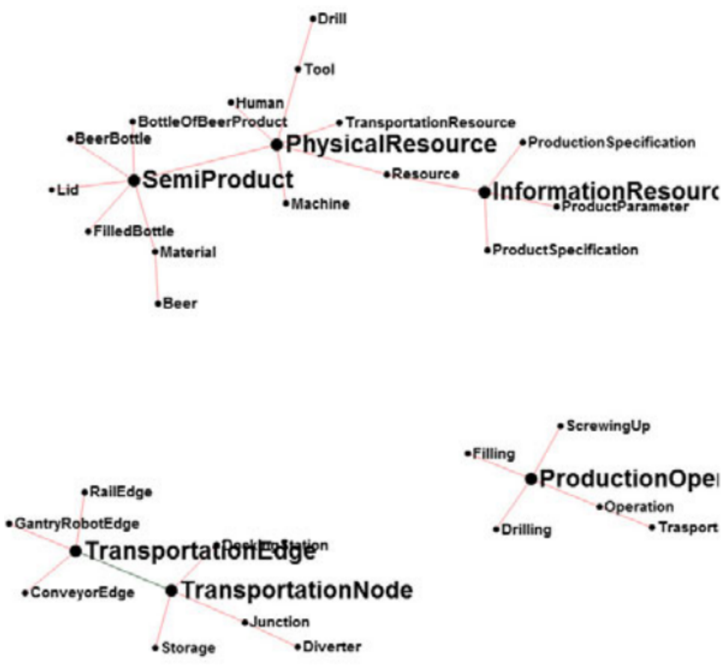


Figure 2.10: The Google Web Toolkit is used to implement Ontovis. The graphical representation of the Ontology is done by using HTML5's canvas element [VKJ+11].

recommendation. Furthermore it discussed the support of HTML5 and SVG elements by the most popular web browsers and mentioned already existing visualizations based on this technology. Over the years the support for the mentioned standards has been increased by every vendor. Nevertheless there is still a huge gap between the supported features of the main players of the browser market. Furthermore there exists a fragmentation of different browser versions within one trademark. These facts complicate the creation of a web application for a wide range of customers. However there exist already a few frameworks who take care about the distinctions between different vendors and browser versions which takes a lot of workload from the application developer. Also the type of usage should by kept in mind by choosing a technology which refers mainly to the difference between stand alone applications and server based applications (e.g. web applications).

3 Design

This Chapter describes the general approach for a visualization with which a user is able to create an illustration of a multi-agent system based on the ISO Standard 14617. At first the requirements - that should be fulfilled by the concept - will be presented. Afterwards the application design approach will be described in detail starting with the introduction of the architecture followed by the description of the *Data Model* and each layer contained by the tier architecture. This includes the free definition of the illustration of each agent as well as the creation of an ontology stored in an XML file. The information should be gathered out of the created diagram that should be the knowledge base for the agents' behaviour and decisions. Furthermore this Chapter includes the description of the general approach for almost real-time state representation of the illustrated MAS created with the visualization. It is the intention that the design of the architecture should ease the implementation and integration of interfaces to other multi-agent systems.

3.1 General Design Approach

This Section will present the general design approach of a multi-agent system visualization which should be able to fulfill the requirements mentioned in Section 1.3. At first an overview of the architecture from the application design will be given. Afterwards the structure of each tier will be described in detail.

3.1.1 Architecture

This Section defines the architecture of the visualization application. A general approach for each layer will be presented. The application structure itself is divided into following layers (see Figure 3.1):

Data Collection Layer - collects the information from the multi-agent system respectively its agents and implements the Interfaces provided by the upper layer. The implementation represents the gateway for notifications

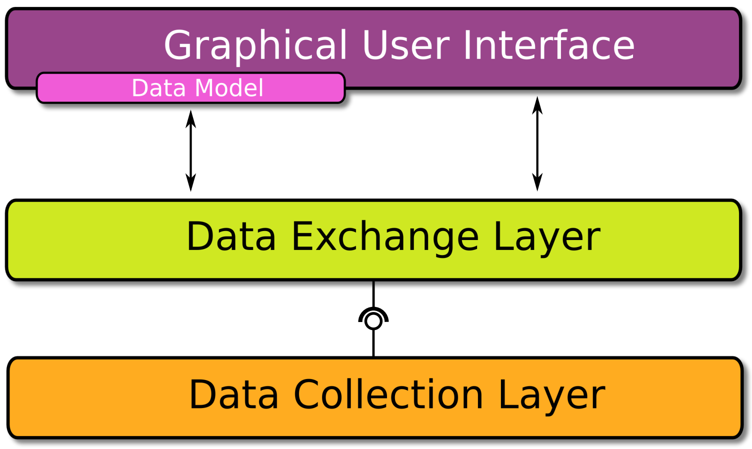


Figure 3.1: This Figure shows the general architecture from the design approach of the visualization.

by upcoming events generated of gathered information from the observed MAS.

Data Exchange Layer - provides the gateway definition for MAS link layers, converts the information from the *Data Collection Layer* - provided through the implemented gateway - to the internal format and processes these events. The information gathered through the notifications of the lower layer is transmitted to the *Data Model*.

Data Model - contains the logical definition of the available agents and their connections and can be seen as key turning point for all MAS relevant data. Its design is also based on the observer model which implies property change support for other classes. It can be seen as part of the *Visualization Layer* because this layer has to create the model entities in conjunction with the visualization itself (depending on the user input).

Graphical User Interface - is responsible for the direct interactions with the user. In offline mode it provides an editor which allows the creation of diagrams that should reflect the real world relationships of the agents. It furthermore provides the creation of the ontology description and the modification of the object attributes. In online mode it illustrates the communication and state transitions within the visualized MAS. In addition attributes with a dynamic value (OPC UA variables) should be displayed in almost realtime.

It is intended, that the *Data Exchange Layer* does not have any references to the *Data Collection Layer*. This allows a simple change of the *Data Collection Layer* to a different implementation of the provided gateway. This approach allows the usage of the same visualization to connect to different multi-agent systems with varying programming APIs by keeping the same user interface.

3.1.2 Data Model

This Section will describe a possible structure of the *Data Model*. The model contains all objects that are required to map a multi-agent system. This includes both physical and functional agents and physical connections. Attributes that are contained by the model's objects are divided into internal attributes that are used for internal data exchange and public attributes that should also be noticed and available for modification by the user. The prefix "_" is used to mark each attribute for its purpose and - according to its usage - will be added to the private attributes' name to distinguish both types from each other. This allows a simple identification of internal and public attributes. An automatic code generation could further be achieved through UML - Tools which are able to generate source code out of UML diagrams. The usage of reflections (that should be available in modern third generation programming languages) allows the distinct identification of the attribute's context.

In the following the base classes that represent the visualized objects will be described (Figure 3.2 illustrates the class diagram):

- The `IConnectableObject` Interface has to be implemented by all objects which should be able to be connected through an `AgentConnection`.

- The abstract class `InspectableObject` implements all necessary attributes and methods to support all derived classes with functionality needed to support the observer pattern which means the ability for other objects to register itself and to be notified if any changes occur at the observed object.

- Basic functionality (respectively attributes) that every agent should be able to provide is bundled in the `Agent` class. This class contains the storage of incoming and outgoing messages, the current state of an agent and a unique name that identifies the agent within the visualization and the agent platform which allows a mapping between both through the agent's name. Furthermore the class includes enumeration attributes that can be used for the distinction between one and several incoming or outgoing messages and the actual state of an agent (online, offline, . . .). The

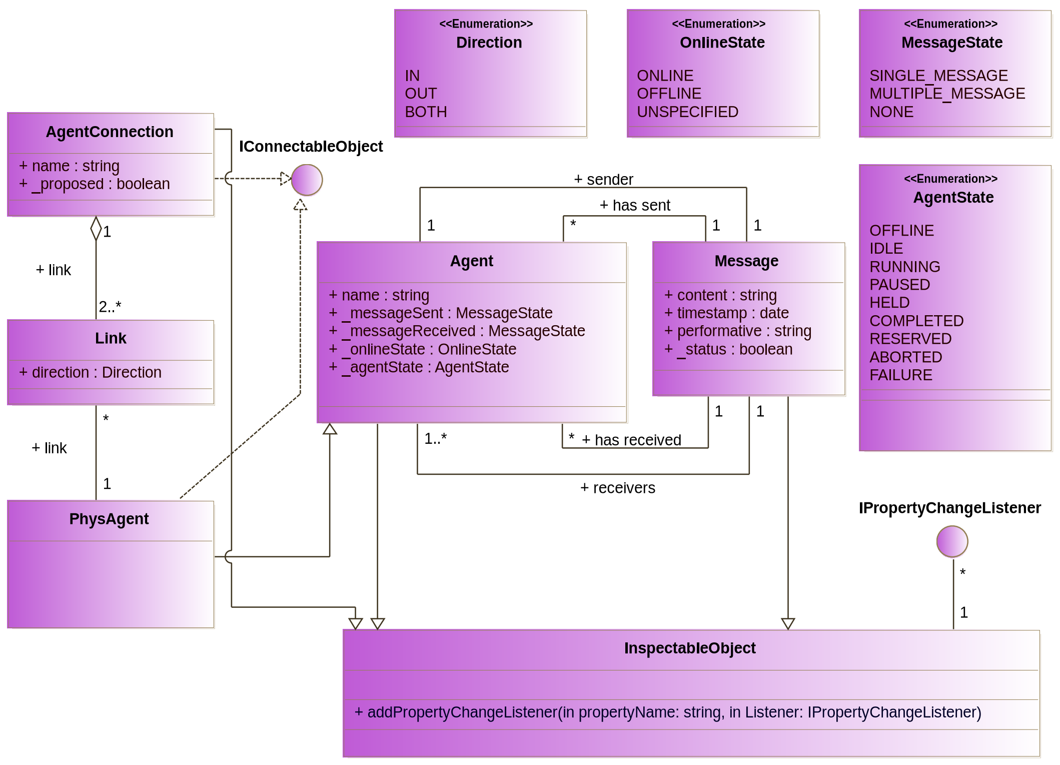


Figure 3.2: This Figure illustrates the conceptional class diagram of the visualized objects.

detailed usage of these attributes (`_messageSend`, `_messageReceived`, `_onlineState`) is described in Section 3.1.4 and 3.1.5. Classes directly derived from `Agent` represent functional agents that do not have any physical part and can not be connected to other agents. This ability is reserved for the `PhysAgents`.

- An agent with a physical component is represented through a class derived from `PhysAgent` and can contain specific implementations or additional attributes depending on the characteristic of the agent. This can either be static information or a connection path for an OPC UA variable whose value should be visualized (see Section 3.1.5 for further details). Attributes which are part of all `PhysAgents` could be added to the `PhysAgent` base class.

- The `Message` class represents a message that can be sent from one agent to at least one other agent. It contains the information of a message and a timestamp. Furthermore the sender and all its receivers are stored in the message.

- A physical connection between at least two `PhysAgents` is represented through an `AgentConnection` class. Every connection is identified by its unique name and can be connected to one or more agents with a physical part. A route can also be suggested for a possible route of a medium. For this proposal the `_proposed` attribute is used (for details see Sec. 3.1.4 and 3.1.5) to notify the visualization that a specific connection is part of a route proposal. The "_" prefix marks the attribute as internal and will not be shown to the user.

- A `Link` symbolizes the endpoint of an `AgentConnection` in conjunction with a `PhysAgent`. It stores the information concerning the possible flow direction at a specific endpoint of a connection. This can be one of the available literals in the `Direction` enumeration.

- The `Direction` enumeration offers the possible flow direction options for an `AgentConnection` endpoint (`Link`). In this case the possible options are IN,OUT and BOTH.

- The `OnlineState` enumeration offers the possible state options for an agent. Available states are `ONLINE`, `OFFLINE` and `UNSPECIFIED`.

- Three different message states can occur at every agent. This can either be no message (`NONE`), one message (`SINGLE_MESSAGE`) or several messages (`MULTIPLE_MESSAGE`) available for both corresponding attributes (`_messageSent`, `_messageReceived`). This applies for incoming and outgoing messages and can be set with one of the available literals of the `MessageState` enumeration.

`Agent` and `PhysAgent` represent the base classes of the visualization and can not be mapped to a physical or functional agent. The mapped classes have to be derived from `Agent` or `PhysAgent` to be able to be recognized by the visualization as available agent definition (in conjunction with the graphical definition - see Listing 3.1).

Somehow a reference must be kept to all objects of the model to be able to access them from other parts of the application. This can be achieved by a `ModelManager` singleton which holds a connection to each created object.

The objects' references can be kept in a simple list. Depending on the demands a separation by class type can be implemented. The mapping between *Data Model* and visualization is done by the manager. Also the saving of data could be realized through this manager because it has access to all entities that belong to the `Data Model`.

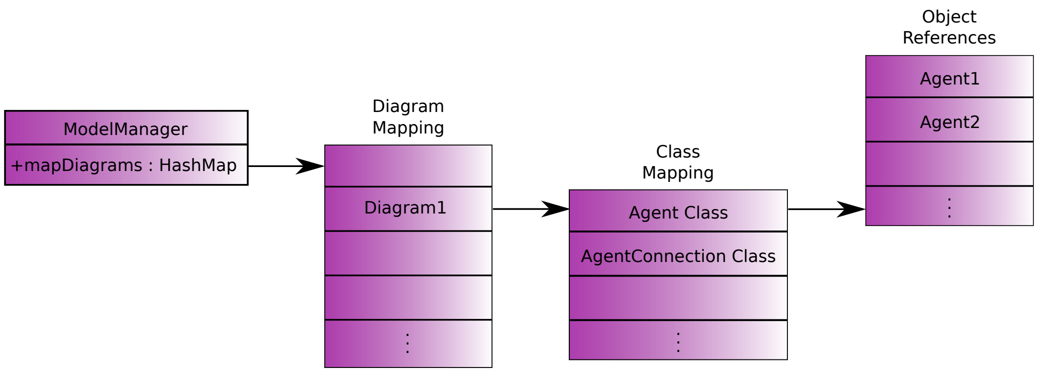


Figure 3.3: The `DataManager` handles the Mapping between the visualization and the included *Data Model* objects. It keeps the diagrams as key of a hash map and uses as value an other map of class definitions (as key). The value in turn is a list of corresponding objects.

A possible structure of the mapping can be seen in Figure 3.3. The references of the objects are hold as weak references. This saves the work from deregistration of objects. The memory of objects which are no longer in use will be collected by the garbage collector automatically and the next time at accessing the list the null references can be deleted.

3.1.3 Data Collection Layer

This Layer is responsible for acquisition of relevant data occurring within a multi-agent system. Ideally existing agents respectively the MAS itself do not need to be modified to gather information of concern. In the worst case each agent's source code has to be extended with additional functionality to provide the necessary data as by the usage of VizScript (see Section 2.3.2 for further details) that collects its information directly from each agent respectively each agent has to provide the logging information on its own. This can cause extensive code changes if the multi-agent system is already implemented and could also influence the temporal behaviour at detailed logging. Nevertheless the realization of the data collection always depends on the underlying MAS and its provided functionality.

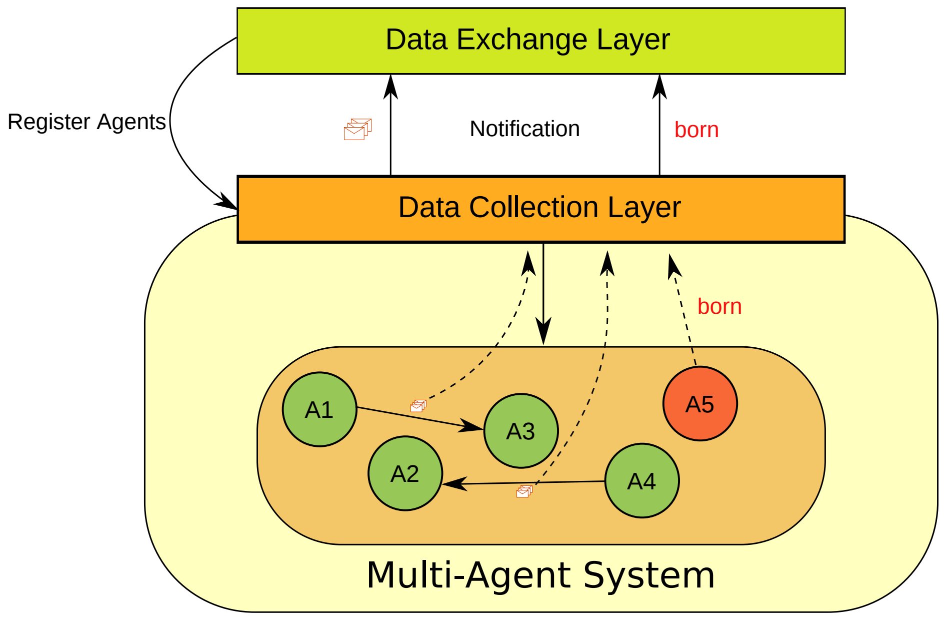


Figure 3.4: The *Data Collection Layer* represents the interface between a multi-agent system and the *Data Exchange Layer*. All agents which are relevant for the visualization will be registered at the lowest layer. This allows a reduction of information at the *Data Collection Layer*. The kind of data acquisition of the layer always depends on the supported features of the MAS.

In general the *Data Collection Layer* has to collect information from agent events (e.g. born, dead, . . .) and messages exchanged by the agents (see Figure 3.4). To minimize the amount of data traffic and to keep a high performance the layer should only collect data that is in matter of concern for the visualization. To achieve this behaviour the *Data Collection Layer* receives the names of the agents for which information should be collected from the upper layer. This offers the possibility to downsize the amount of data at the lowest layer. During data acquisition the *Data Collection Layer* forwards - via notifications - the gathered information to the upper layer. The connection between both layers is done through a predefined interface which is described in the next Section.

3.1.4 Data Exchange Layer

This Layer is responsible for the data exchange between the *Data Collection Layer* and the *Data Model*. The provided architecture should allow a simple replacement of the *Data Collection Layer* without changing any other parts of the application. This makes the application more flexible to be used with different multi-agent systems. But to achieve a loose coupling between the lower two layers an interface has to be provided. This allows the communication with the *Data Collection Layer* without referencing any classes of the lower layer.

In the following the classes and their responsibilities of the provided gateway definition(contained in Figure 3.5) will be described in detail:

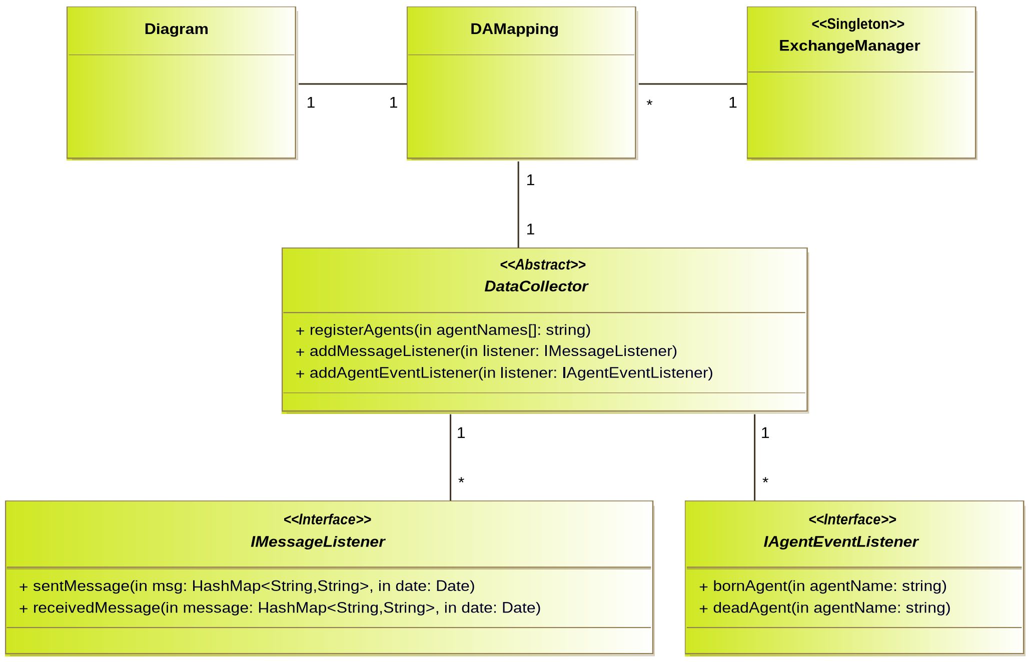


Diagram
DAMapping
<<Singleton>>
ExchangeManager
1
1
*
1
1
1
<<Abstract>>
DataCollector
+ registerAgents(in agentNames[]: string)
+ addMessageListener(in listener: IMessageListener)
+ addAgentEventListener(in listener: IAgentEventListener)
1
*
1
*
<<Interface>>
IMessageListener
+ sentMessage(in msg: HashMap<String,String>, in date: Date)
+ receivedMessage(in message: HashMap<String,String>, in date: Date)
<<Interface>>
IAgentEventListener
+ bornAgent(in agentName: string)
+ deadAgent(in agentName: string)

Figure 3.5: The class diagram represents the base interfaces and classes which are involved into the data acquisition and data model feeding.

- The `DataCollector` is responsible for the data acquisition from the corresponding multi-agent system. The implementation varies depending on the used MAS. Through the method `registerAgents` agents can be indexed with their names. This allows data filtering at the lowest layer and minimizes the appearing data. Furthermore the interface provides methods to register listeners for message and agent events. The listeners will be notified at the occurrence of relevant data referring to the indexed agents. Through the `start` and `stop` methods the data collection from the underlying multi-agent system can be started or stopped.

- An `IMessageListener` implementation is registered at the `DataCollector` and should be notified at any message sent by one of the inspected agents. The listener itself forwards the information to the *Data Model* and sets the `messageReceived` and `messageSend` flags at the affected agents (see Figure 3.2). The appearance of multiple incoming or outgoing messages is also recognized by the listener because an occurrence of multiple messages sent or received by the same agent has to be reflected in the visualization. Nevertheless the message flag has to be cleared after a specified amount of time which can be done by creating a `MessageTimer` which is responsible for resetting the flags. The notification of the *Visualization Layer* for any changes is done by the *Data Model* itself. This allows the visualization to stay passive and only react on changes occurring at the *Data Model* to avoid the need of direct impact from the *Data Exchange Layer* to the *Visualization Layer.* The usage of a `Hashmap` as parameter of both notification methods allows a flexible transfer of message content from a `DataCollector` to the listener. Furthermore an enhancement to support new message content can be easily reached through new key-value pairs in the hash map. Due to the fact that messages sent by an agent and received by another agent will be the same a recognition of identical messages will be necessary. This could also be fulfilled by a `Hashmap` which stores a generated hash code out of the message content. Both methods `sentMessage` and `receivedMessage` will use this map to recognize "duplicate" messages. If a sent message event has already occurred for a specific message and an entry in the map has been done then the processing of the received event can use the same stored message and just needs to set the `messageReceived` flag of the effected agents.

- An `IAgentEventListener` is responsible for all state changes of the agents within the MAS. It is registered at the `DataCollector` and handles the notifications about born and terminated agents. The information is stored in the *Data Model* which triggers a notification for the *Visualization Layer.*

- For the possibility to visualize multiple diagrams within the same application a mapping has to be provided between the diagram and the agent who listens to the MAS. This is achieved by the usage of an `ExchangeManager` singleton that maps the diagram and the corresponding agent by holding its references within the `DAMapping` class. The mapping could also be done through an ordinary `Hashmap` but the storage within a separate class allows the possibility to add additional functionality and the storage of supplemental information. This for example could be graphical information that is changed between offline and online mode (e.g. background color, zoom factor, ...). To reset the proper values the information has to be stored temporary related to the diagram and can be retrieved at switching back to offline state to restore previous settings.

The registration of listeners at the `DataCollector` is not restricted to one listener which offers the possibility to register several `IMessageListener` with varying responsibilities. An additional listener could manage route proposal messages that contain the information of the included connections and forward this information to the *Data Model*. The usage of a timer to highlight and reset the proposed route (using the `_proposed` attribute of a connection) would be suitable. Furthermore this approach allows a better separation of concern, an easier replacement of existing implementations and for further development a better readability and maintainability.

3.1.5 Graphical User Interface

The graphical user interface is the main item which has the most influence on the acceptance by users. A smooth handling of the user interface eases the creation and configuration of a visualization diagram. It is important that a provided editor supports common functionality by working with graphical objects.

This implies:

- standard keyboard shortcuts to create, delete and move items
- standard mouse functionality as drag-and-drop, resizing and rotating of objects
- a context menu for each item which offers additional functionality

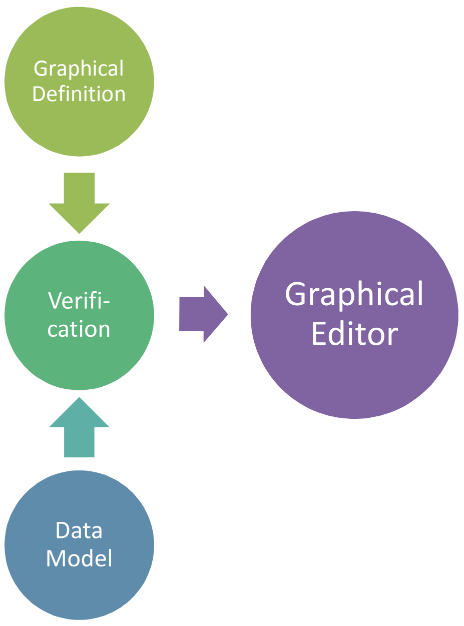


Figure 3.6: A graphical editor gathers its information for available agents - that are available within a visualization - from the *Data Model* combined with the graphical definition stored in an XML file. The mapping between both is done through the agent's class name.

For a higher flexibility of a user interface and to minimize source code changes, the information for graphical representation named graphical definition is separated from the *Data Model* information (see Figure 3.6). The structure of a graphical definition, the selection of needed data from the *Data Model* and the handling of graphical key features will be described in detail in this Section. The implementation of an editor itself and an achievement of a smooth handling for users always depends on the used underlying technology and will be discussed in Section 4.8. Still it has to be considered that the same visualization has to be used for the creation of the diagram and for the visualization of the online state within an MAS and the logical state of an agent.

At the developers side the re-usability also has an important impact on the acceptance of a visualization. For that reason an editor has to collect its information for the agents' representation from external sources which can be modified without any needed modification of source code.
The visual part - which means the presentation of an agent - is stored in an

external source (e.g. XML file - see Listing 3.1) This includes the definitions of all agents that are available within the editor. A graphical definition can be modified without any intervention into other parts of the application. An agent - identified with its `name` attribute - can contain different types of shapes and corresponding attributes. Nested objects can either be additional information for the defined shape (e.g. defined points of a polygon) or another shape.

```
<agent name="Tank">
        <shape name="polygon" filled="false" lineWidth="2">
                <point x="0" y="0"/>
                <point x="0" y="100"/>
                <point x="100" y="100"/>
                <point x="100" y="0"/>
        </shape>
</agent>
```

Listing 3.1: An example of the definition for the graphical representation of an agent within the visualization.

The agent *Tank* defined in Listing 3.1 is illustrated through a polygon that contains multiple attributes. The polygon's sub-objects define the points within the polygon. The defined points' coordinates are given as absolute positive values with the initial point of 0/0 and the maximum of 100/100. This allows simple resizing and rotation of objects. Further nesting of other shapes (e.g. line, circle, ...) could be an option for sophisticated agent graphics. The provided information should be sufficient to draw, rotate and resize every graphical agent object. This can be simple objects as a tank up to more complex illustrations (e.g. valve, pump,...). Notifications about incoming or outgoing messages can be realized through decorators at the illustration (see Figure 3.7 for an example). The affected messages can be visualized through tooltips of the decorator. A supplemental view for the message history has to be provided which lists a specific number of messages of the past. At least all objects from the ISO standard 14617 should be able to be defined by supported shapes to represent all possible agents within an MAS.

Additional information concerning the logical part (e.g. agent types, attributes,...) can be gathered from the *Data Model* (see Section 3.1.2 for further information). The combination of both parts (see Figure 3.6) provide the required information for an editor to represent each agent and its attributes. Basically all changes of information concerning agents can be recognized through the *Data Model*. The visualization tier will register at all attributes of importance an `AttributeChangeListener` to recognize every relevant data modification. That way any attribute change within the *Data Model*

Figure 3.7: This Figure illustrates a Tank defined in Listing 3.1. The mail symbol appears at incoming messages for this agent. Another color can represent outgoing messages. Furthermore other events could be visualized by other symbols (e.g. critical state of an agent, specific state changes, ...). The color of an agent can give an information about the agent's state (e.g. online, offline, waiting, ...).

can cause an impact on the graphical illustration. It must be distinguished between the manipulation of data at configuration time that should allow all types of values whether static or OPC UA path information. A string recognition will be necessary for online illustration to decide whether the attribute's value or the content of an OPC UA variable has to be expressed. The necessary access to variables could be realized through reflections.

The creation of an agent should follow a common editor behaviour which implies a free definition of the size of an object. With the values stored in the configuration the real size of an object can be calculated.

An important part of drawing a visualization diagram is the creation of connections between agents. The connections represent pipes that are available for route proposals from one object to another and that can be physically used for transportation of fluids between different physical objects (e.g. tanks). Beside the fundamental functionality of connecting to `PhysAgents` with each other following additional features should be included to enlarge the scope of graphical representations (see Figure 3.8 for a possible illustration of the features):

a.) The creation of vertices which can be freely added and customized depending on the user's need to create a better overview and to refurbish the graphical representation.

b.) The possibility to create connections with several endpoints through defined branches.

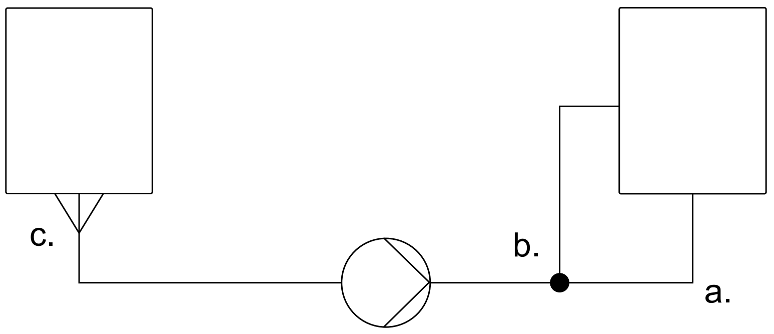
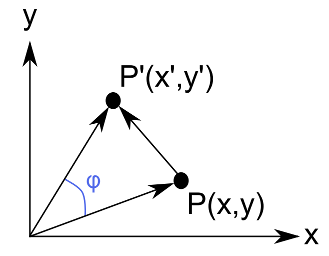


Figure 3.8: In order to create attractive illustrations routes should not just be straight lines but the editor should allow the creation of individual paths (a.) (e.g. with corners) and the possibility to produce branches (b.). Another important aspect is the definition and graphical representation of flow directions (c.) (e.g. for the creation of routing tables).

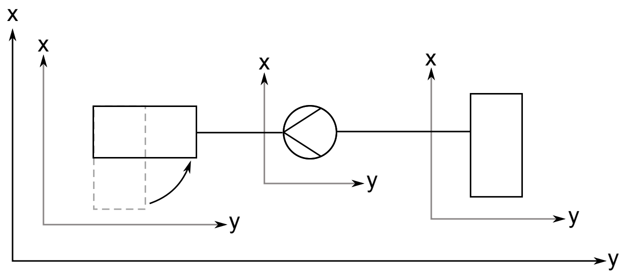
c.) A graphical representation of flow directions at endpoints of a connection.

The recognition of a route proposal happens in the middle layer (see Sec. 3.1.4 for further details) but the illustration has to be done by this tier. As already mentioned connections between agents are also mapped in the *Data Model* and a change of data can be induced by the *Data Exchange Layer*. The graphical change as reaction on attribute changes happens in the same way as for agents. For route proposals the visualization has to react on changes at the `_proposed` attribute. A boolean value of true means a proposal for a route - that includes the specific connection - has been generated.

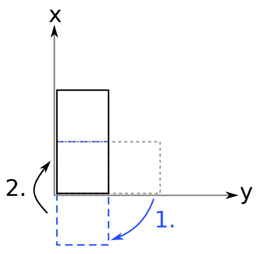
A simple placement of objects within an editor is not enough to draw a usable schema of a multi-agent system. Rotations of drawings are needed to justify the right direction of placed objects. The rotation is realized through the rotation of each defined point of one figure around a fixed point in a coordinate system. The assumption that every object is drawn within its own coordinate system eases the handling of transformations of a drawing. Figure 3.9(a.) illustrates the rotation of a point. The result of a calculation can also lead to negative coordinates which must be corrected for a proper graphical illustration of an object. This is reached through a translation of the object depending on its minimum extrema on the x and y axis and to add the extrema



(a) Rotation of a single point



(b) Rotation of an object within the entire drawing



(c) Translation of a rotated object

Figure 3.9: Figure (a.) illustrates a rotation of a point around a fixed point with the angle of φ in the coordinate system. The calculation is done with the equation shown at 3.1. (b.) every object in the diagram has its own coordinate system in which it is drawn and that is used to calculate transformations. (c.) If one point obtains a negative coordinate the whole object has to fulfill a translation into the positive quadrant.

as offset to all included points in a drawing (Figure 3.9 (c.)).

The new coordinates of the rotated point result out of:

$$\begin{aligned} x' &= cos(\varphi) * x - sin(\varphi) * y \\ y' &= sin(\varphi) * x + cos(\varphi) * y \end{aligned} \tag{3.1}$$

In that way it is possible to rotate every drawn figure in a visualization diagram independent of the size of a drawn agent.

As illustrated in Figure 3.9(b.) each object has its own coordinate system that is used to calculate transformations of an object. The transformation is calculated with those values defined in the graphical XML definition (see Listing 3.1). The real size of an object is calculated with width and height of an object in a diagram in conjunction with the defined size.

3.2 Summary

Chapter 3 presented a new approach of a possible multi-agent system visualization. It should ease the creation of new visualizations independent from the underlying multi-agent system. Three autonomous tiers constitute a flexible architecture that allows a simple change and extension from layers of the visualization without influencing other parts of the application.

Section 1.3 listed the requirements which should be fulfilled by the presented approach. A multi tier architecture with three tiers has been chosen for designing the visualization. This guarantees a higher reusability with different MAS. Dependencies to other layers are avoided for a better replaceability. The definite implementation of the *Data Collection Layer* always depends on the underlying multi-agent system with which this layer has to interact. The *Data Model* presented in Section 3.1.2 is fed with information by the *Data Exchange Layer* introduced in 3.1.4. The *Data Model* provides other layers with notifications of data changes.

The top most layer is responsible for the representation of a graphical user interface. This implies an editor for the creation of a diagram, the configuration of all producible objects and the generation of the ontology in an XML format to feed the physical agents with the needed information to operate in the right manner. In addition the same visualization acts as online view for a almost real-time representation of the connected multi-agent system. This should allow a rapid prototyping for visualizations and furthermore for multi-agent systems without much knowledge of the underlying framework of the visualization and MAS. The optimal procedure of the visualization creation and its usage is presented in Figure 3.10.

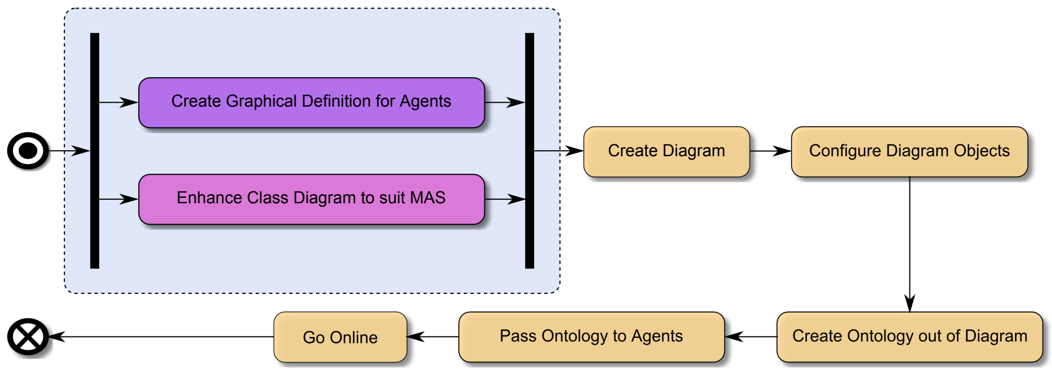

Create Graphical Definition for Agents
Enhance Class Diagram to suit MAS
Create Diagram
Configure Diagram Objects
Create Ontology out of Diagram
Pass Ontology to Agents
Go Online

Figure 3.10: The activity diagram illustrates the workflow of the creation from a visualization. The graphical and logical definition of an agent are separated from each other. The creation of a diagram is based on the information defined before. Each agent and connection can be configured. The information is the base for the agents' ontology. After the configuration of the agents the visualization can go online and visualize the condition of the MAS. The dashed line surrounds those actions that can be done by a developer (or for example tutor at the university). The user (or student) can get familiar with the MAS by performing the remaining steps.

4 Implementation

This Chapter describes the implementation of the represented approach in Chapter 3. At first the framework decision for the implementation will be discussed. After that the chosen technologies will be described and the implementation of the application will be illustrated.

Due to the fact that the multi-agent system developed by the ACIN Institute at the Vienna University of Technology is based on the JADE framework a short introduction to this specific MAS will be given.

According to the desired goals the integration in an already existing development platform is aspired to reach a seamless cooperation with already used development tools.

4.1 JADE Framework

This Section will give an overview of the JADE framework. As base of the MAS which is used at the Odo Struger Laboratory JADE also plays an important part for a visualization. It's entire implementation is compliant to the FIPA standard for multi-agent systems. The information which is exchanged by the agents must be somehow displayed to give the user the opportunity to follow the information flow within a multi-agent system. Also the current state of each agent (online, offline,...) is an important information which should be reported to the user. For better understanding the main parts of the JADE architecture will be introduced (see Figure 4.1):

- A **Platform** comprises all virtual components which belong to an MAS. This implies the containers, Agent Management System (AMS), Directory Facilitator Agent (DF) and all agents. The platform is not limited to one physical device. It can also be spread over several machines.
- Every platform has a single **Main Container**. The unique agents AMS and DF are contained in this container.
- Beside the Main Container several other **Containers** can exist. Normally the location of additional containers is on other physical machines. They register itself at the Main Container.

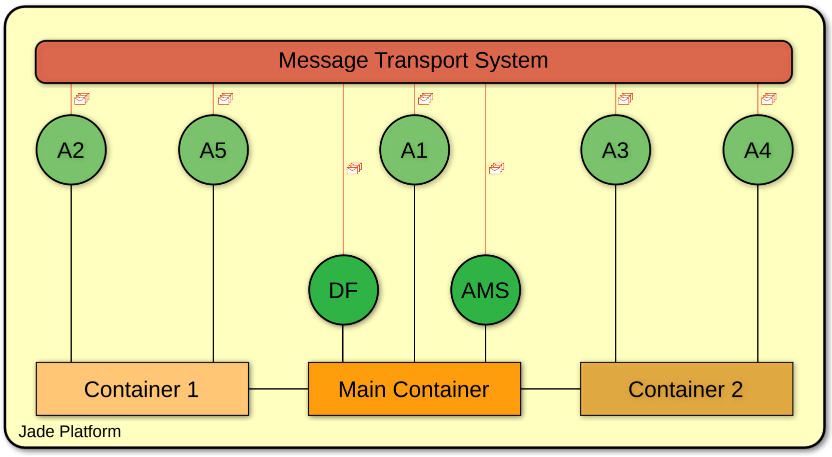


Figure 4.1: This Figure illustrates the architecture of the JADE framework. The platform consists of at least one Main Container which is in charge of an Agent Management System (AMS) and a Directory Facilitator (DF). Additional containers can be registered at the Main Container. Every container can hold multiple agents. The communication is done through the Message Transport System.

- The **Agent Management System (AMS)** is also an implementation of an agent but with special purposes. This agent is unique in the entire platform and every agent has to register itself at the AMS to gather a unique Agent Identifier (AID). Furthermore the AMS agent provides a white-page and life-cycle service from all agents. At startup it is always immediately created and lives in the Main Container of the platform.
- The **Directory Facilitator (DF)** is also kept in the Main Container and acts as yellow-pages service for other agents. The DF is aware of all available services provided by other agents. Together with the AMS it is directly started after the platform startup.
- **Agent** is the overall name of all actors within a platform. Of the programmers point of view the `Agent` class represents the base of all user defined agents. The agents life-cycle is coherent to the FIPA specification. This implies (see Figure 4.2 for possible states and their transitions) [BCTR]:

Initiated The agent is created but not registered at the AMS. It has no address and is unable to communicate with other agents.

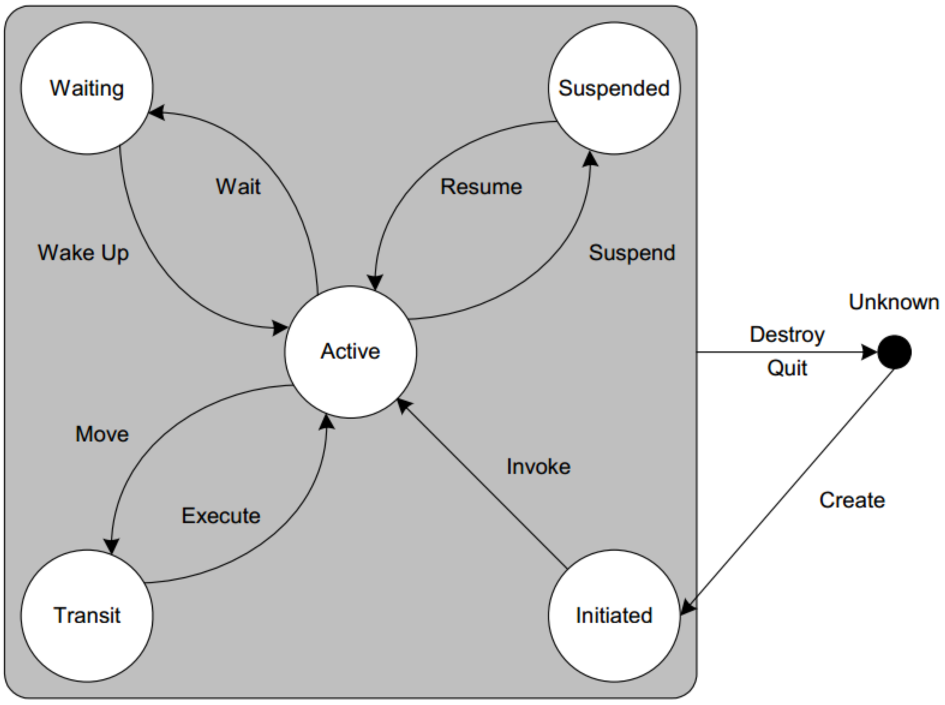


Figure 4.2: This state diagram represents the life-cycle of an agent within the JADE multi-agent system [BCTR].

Active	The agent is registered at the AMS, has a corresponding AID and full access to all JADE features.
Waiting	The agent is sleeping and waiting for events to execute further operations. This can also be messages from other agents.
Deleted	The agent has been stopped and its thread is terminated. The registration at the AMS has been removed.
Transit	An agent enters this state while it is moved to another location. All messages directed to this agent will be buffered while the agent is transferred.

The agents communicate through the Message Transport System also called Agent Communication Channel (ACC). It is responsible for the exchange of all messages within the platform and to remote platforms. The structure of a message is compliant to the FIPA 2000 "FIPA ACL Message Structure Specification" specifications [acl] but the complexity of the creation and access of the content is encapsulated through the `ACLMessage` class.

An agent executes its tasks within behaviours. The framework provides several different standard behaviours which fulfill most of the requirements of tasks an agent should perform. An abstract of this behaviours includes the three primary behaviour types that are available in JADE [FBG07]:

- An `OneShotBehaviour` can be used for tasks that should be executed only one time.
- The `CyclicBehaviour` is used for tasks that should be executed continuously and which will never complete. Each time it is called the same operation will be executed.
- Generic behaviours are classes derived from the base class `jade.core.behaviours.Behaviour`, contain a status trigger and fulfill different operations depending on the value of the trigger. After a specific status value is reached the behaviour is completed.

The package `jade.core.behaviours` contains several other predefined behaviours which allow among others scheduled operations (`WakerBehaviour`, `TickerBehaviour`). The combination of different behaviours allow the execution of complex tasks and conversations with other agents.

4.2 Test Equipment at the Odo Struger Laboratory

It is the aim of this master thesis to provide an application which eases introduction and development of multi-agent systems. The focus lies on the MAS technology used for the test plant at the Odo Struger Laboratory. This Section will give an overview of the architecture used for the agents and afterwards an introduction to agent's communication among each other.

A multi-agent system consists of at least one but mostly of multiple agents which can represent physical (e.g. tank, pump, ...) and non physical components (e.g. order agent, task agent). Primarily the agent's software components are divided into two parts (see Figure 4.3) [MLA11]:

- High Level Control (HLC) - this part is responsible for the interaction with other agents. It is able to perform high level diagnostics and uses an ontology based world model which offers the agent an overview of its immediate neighborhood and subsequent the ability to coordinate itself with agents near by. As a result of possible longer calculation times and interactions with other agents the HLC does not act in real-time.

- Low Level Control (LLC) - this layer executes its tasks in realtime and is responsible for controlling the hardware. It is able to collect sensor data and to send commands to the actuators. The design of the LLC is based on the IEC61499 standard. It is normally directly executed on a programmable logic controller (PLC) which is wired to the hardware controlled by the agent.

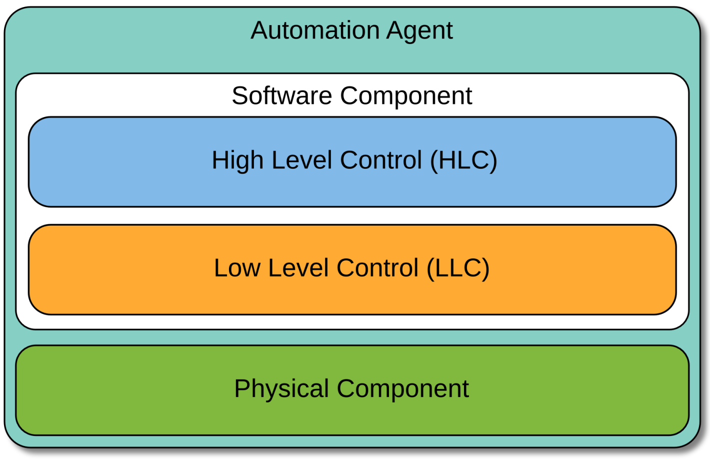


Figure 4.3: This Figure illustrates the architecture of an agent. The software component of the agent is divided into two layers - the high level and low level control [MLA11].

Agents who are in charge of a physical hardware (e.g. valve) are named automation agents. The automation agents itself can be responsible for one entity or either a group of entities. The agent registers itself and its available services at the Directory Facilitator Agent (DF Agent). The DF agent acts as registry of all agents and their corresponding services which they offer to other agents. The functionality and its behaviour of the DF agent is defined in the FIPA standard which is the base for the implementation of JADE.

The second type of agents is called functional agent (e.g. task or order agents) which have the same design but miss the LLC Layer which is not required for this type of agents. The ontology of the physical agents for the test plant is represented in Figure 4.4.

It consists of following automation agents:

- two tanks T101 and T102 which can be filled with some liquid respectively can offer liquid which will be transported to the other tank
- one pump P101 which is able to pump the liquid (depending on the condition of the valves) from one tank to another
- six valves V101-V106 which are able to impact the used direction of the liquid depending on their condition
- several pipes L101-L108 which connect the devices with each other.

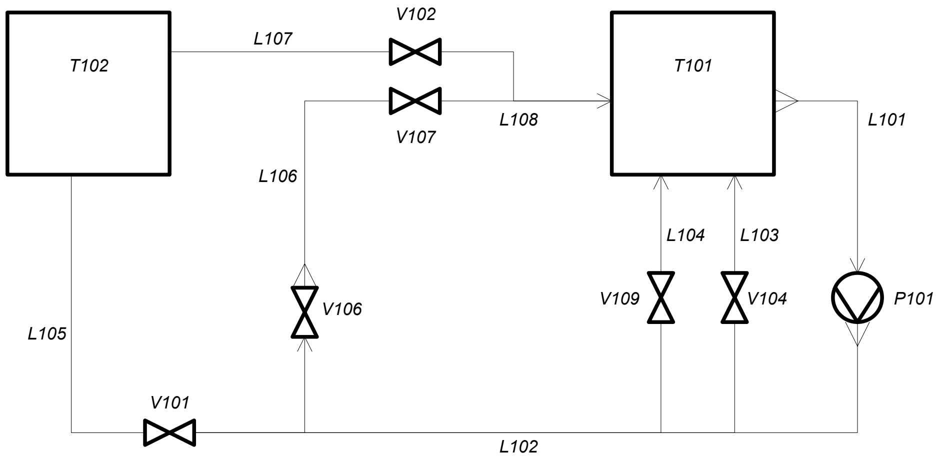


Figure 4.4: This Figure illustrates the test topology of the equipment at the Odo Struger Laboratory. It represents the hardware which is used by the test plant in Figure 1.2. The symbols are based on the IEC Standard 14617 that describes the graphical representation of devices which are used for a representation of batch oriented plants.

The graphical representation of the ontology is based on the IEC Standard 14617 which describes the graphical representation of the devices used in the ontology. The implementation of a visualization should follow the same standard to be able to provide consistent illustration of the equipment within a plant.

Furthermore following functional agents (without a physical part) are used to manage the physical agents included in the ontology [MLSV10]:

- The Order Agent represents the interface to an external source. The agent offers the ability to generate new orders either by a human or an external application. It identifies the corresponding recipe to generate the ordered product, creates a new job and notifies the Task Agent to process the new order.
- The Task Agent is responsible for the execution of new jobs. It evaluates the agents at the Directory Facilitator that provide the accurate equipment to perform the requested job. The recipe is splitted in tasks and passed through to the Work Agents which will perform the tasks. After all tasks are finished the agent notifies the Order Agent about the completion of the batch process.
- The Work Agent is responsible for the execution of the tasks. It makes decisions for possible routing paths and sends commands to the automation agents which control the needed equipment (or indirectly through Recipe Agents). Calculated route proposals are made public through a predefined message sent by the work agent (the content is described in table 4.1). After finishing a task the Work Agent informs the Task Agent and waits for new tasks to perform.
- A full set of commands (recipe) is executed by the specific Recipe Agent. It receives its commands from the Work Agent to perform its underling process.
- A Failure Handling Agent monitors the system - by receiving failure notifications - for a possible appearance of anomalies - and is able to execute supervisory functions to bring the system back into a nearby optimal state.

Those agents are not included in the ontology because they do not have any physical part which could have a possible impact on routing path decisions of the medium that has to be transported. Nevertheless these agents are essential for a successful interaction and cooperation of all agents.

Beside the multi-agent system's point of view an agent can also represent (see Section 4.1 for details) different logical states that reflect the current condition

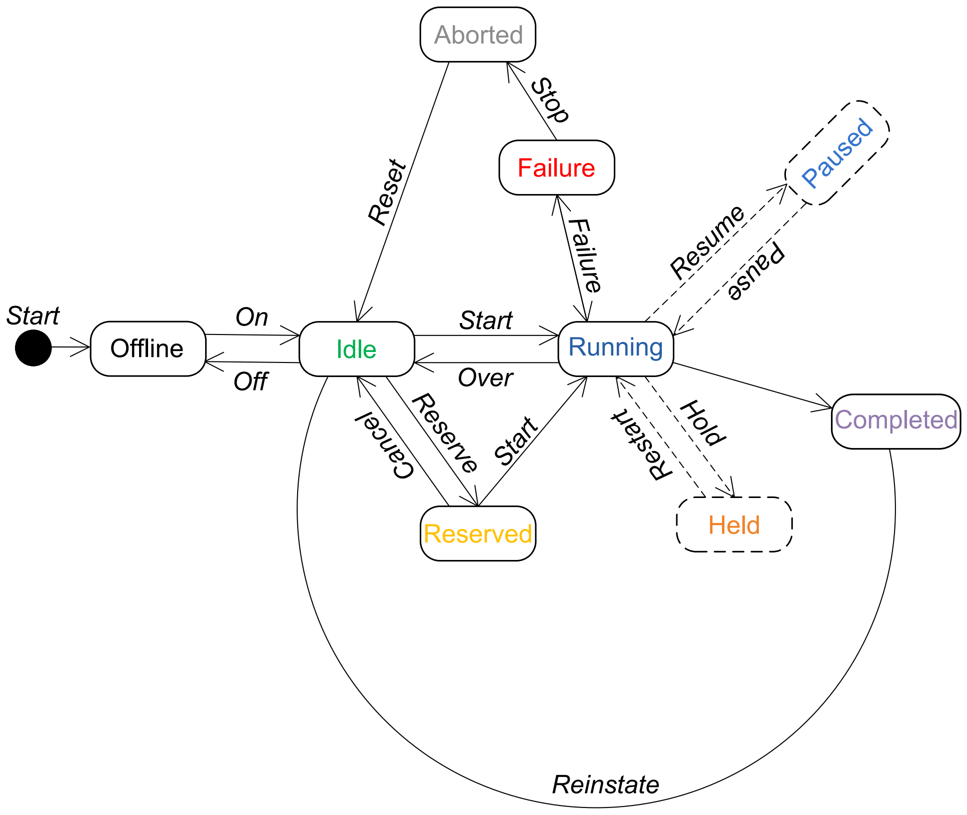


Figure 4.5: The state diagram figures possible states of an automation agent and the relations between the states.

of an agent. Possible states and their transitions are visualized in Figure 4.5. The notification for changes of states is done through messages sent by the effected agent. The determining message content is illustrated in table 4.1.

4.3 Framework Decision

This Section will describe the decision process for the underlying technology of the implementation. In the choice of technology, great importance has been attached to platform and manufacturer independence. Platform independence is mostly provided by Java based IDEs but to avoid licensing conflicts in the future the underlying licenses have also been included by making the decision.

Message Property	Values for Route Proposal	Values for Agent State Change
Sender	any mapped agent	affected agent
Receiver	any mapped agent	affected agent
Performative	'inform'	'inform'
Content	names of the proposed routes separated by a comma	number of the new state
Ontology	'route-proposal'	'status-update'

Table 4.1: This table lists affected attributes that are used by two specialized messages provided by the MAS implementation.

The following list roughly compares the five most popular IDEs[1] (see Table 4.2 for details):

- Eclipse[2] is a product of the Eclipse Foundation and licensed under the terms of the Eclipse Public License. The Eclipse Foundation is a not-for-Profit organization which provides services to the community but does not employ any developers. Source Code for eclipse projects is developed by volunteers and organizations for no charge [ecla]. The Eclipse IDE roughly consists of a base and many plugins which provide the functionality. Furthermore it can be used to develop own applications on top of it (or by using the Rich Client Platform (RCP) for standalone applications without any need of IDE functionality).

- IntelliJ IDEA[3] is a product of JetBrains and available under two different licenses. The community edition is open source and licensed under the Apache 2.0 license. The second edition is commercial and can be purchased under different licenses depending on the field of application. The two versions differ in their range of functions [int] and in its covered support. IntelliJ IDEA is based on Swing, also supports the usage of plugins to enhance the functionality of the IDE and can be used as base for own applications.

- NetBeans[4] is completely open source and developed by Oracle. It is currently available under two different licenses, the GPL[5] and the CDDL[6].

[1]Integrated Development Environment
[2]http://www.eclipse.org/
[3]http://www.jetbrains.com
[4]http://netbeans.org/
[5]Gnu Public License
[6]Common Development and Distribution License

NetBeans also supports - as Eclipse and IntelliJ - the usage of Plugins and can be used as base for own applications. NetBeans itself is based on Swing.

- MyEclipse[7] is a product of genuitec and is based on the Eclipse Platform. MyEclipse is - in contrast to Eclipse - a commercial product and offered in different versions which vary in their range of supported functionality. Technically MyEclipse offers the same features as the Eclipse IDE regarding the usage of Plugins but the standard coverage of functionality varies depending on the edition.
- JDeveloper[8] is - as NetBeans - a product developed by Oracle but covered by a proprietary license (OTN[9] JDeveloper License). The software can freely be used to develop applications for non commercial usage. Any commercial usage of products developed with JDeveloper presume another license [jde]. The development platform targets the development of applications which are in conjunction with other Oracle products and is not suited for the production of own products on top of it.

	Eclipse	IntelliJ IDEA	NetBeans	MyEclipse	JDeveloper
Technology	SWT	Swing	Swing	Eclipse based	Swing
Developer	The Eclipse Foundation	JetBrains	Oracle Corporation	genuitec	Oracle Corporation
License	EPL	Apache 2.0 License/ Commercial	GPL	Commercial	OTN JDeveloper License (Proprietary)

Table 4.2: Comparison of the most popular IDEs

Note that through the acquisition from Sun through Oracle the company became responsible for NetBeans. Since this purchase Oracle has to develop and maintain two development platforms (JDeveloper and NetBeans). Beside these two IDEs - as Eclipse Foundation Member - Oracle has committed support of 6 fulltime developers to the Eclipse Foundation and JDeveloper is still heavily integrated into Oracle's tool chain which is seen as strategic development tool for Oracle's applications. Nevertheless Oracle has assured that the support of all three development platforms will be continued but the focus of supported figures and technologies in JDeveloper and NetBeans will differ [net]. Nevertheless the further development progress of NetBeans has to

[7]http://www.myeclipseide.com/
[8]http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.htm
[9]Oracle Technology Network

be followed to be able to decide if it could be used as platform for possible future development.

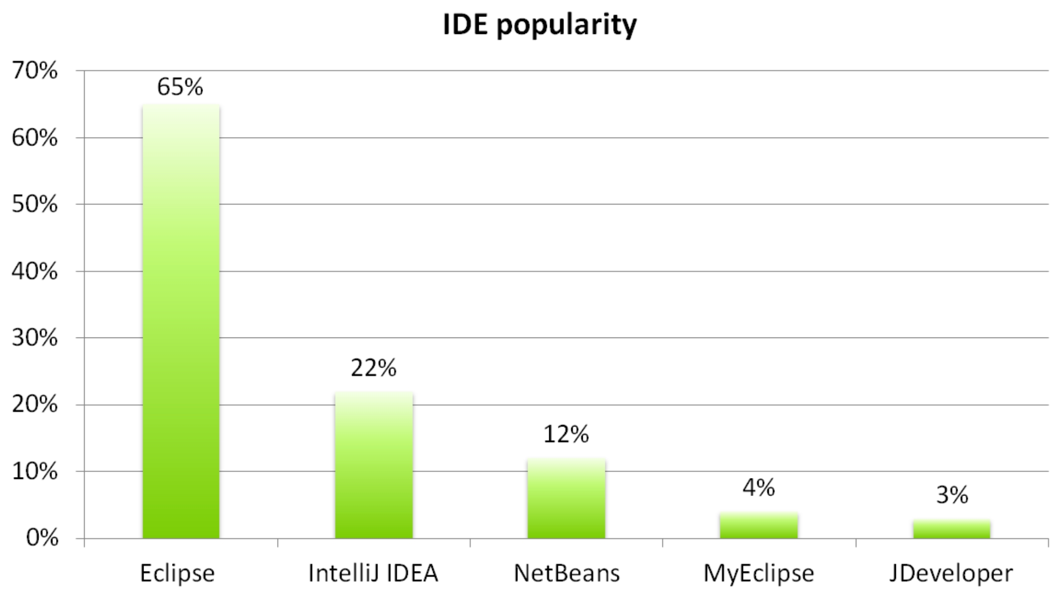


Figure 4.6: Comparison of IDE popularity from a survey of over thousand Java developers [ide].

Based on a survey among more than thousand developers Eclipse is by far the most popular development platform (see Figure 4.6) used by developers (not only Java) and furthermore the only platform which is by whole developed by the community. It is seen as first choice for developing a new visualization with a considered integration into the agent development process.

4.4 Eclipse based Technologies

Due to the fact that Eclipse is the most popular development platform it is desirable to integrate the proposed approach in Chapter 3 into the development process of the agents to achieve a shortened time of development.

Eclipse (respectively the Eclipse SDK) that is built on top of the Rich Client Platform (which is based on the OSGI[10] framework) framework offers a feature rich fundament for an expandable platform. This fact can be utilized

[10]Open Services Gateway initiative

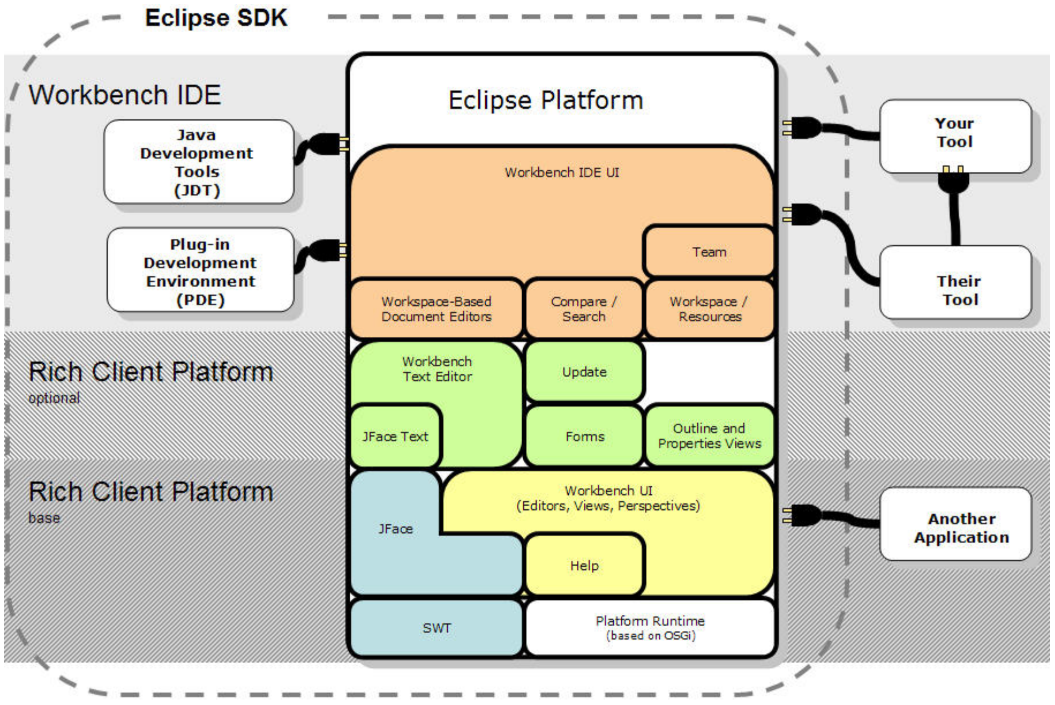


Figure 4.7: Architecture of the Eclipse IDE on top of the RCP framework. Furthermore the possible enhancement through plugins is illustrated [Eclc].

to integrate the presented approach into existing Eclipse IDEs. Already implemented features as the update feature can be very helpful to provide the following implementation as extension via an Eclipse update site (Figure 4.7 gives an overview of the Eclipse SDK and its comprised features). An important fact is the usage of the plugin pattern from the Eclipse architecture:

- A **plugin** represents a component that can provide a certain type of service to others. It defines among others it's classes' visibility, its extensions, extension points and dependencies through a manifest file (XML document) which is interpreted by the Eclipse runtime.
- A plugin can provide **extension points** which can be used by other plugins to hook an **extension** into an existing plugin. This can be seen as a kind of enhancement of an existing plugin or as interface for other plugins to achieve a higher replaceability and flexibility through independent software parts.

- Plugins which belong to the same type of context can be combined to a feature (e.g. visualization feature) that can be offered - through an update site - to other developers for a simple integration into their existing IDE.

The clarified characteristic already fulfills parts of the requirements from the presented approach. The proposed design principals of Eclipse enhancements are used to define the design of the implementation of the approach presented in Chapter 3 and is structured as follows:

- **at.rodler.graphiti** contains the graphical editor which provides all functionality to create a diagram of a multi-agent system. Furthermore it represents the reflection of the data within the *Data Model* in offline and online mode and offers the possibility to view the data gathered through the OPC UA interface in online mode.

- **at.rodler.emf.model** involves the definition and the corresponding classes of the data model.

- The plugin **at.rodler.emf.model.edit** adds additional functionality to display and modify the data within the model through generated property sheets.

- **at.rodler.dataexchange** handles incoming data provided by the *Data Collection Layer* and writes the data to the *Data Model.* Furthermore it informs the *Visualization Layer* to refresh the content on demand.

- **at.rodler.datacollection.jade** is responsible to gather data from an existing MAS (in case of the implementation - from JADE).

Figure 4.8 illustrates the tiers of the architecture and their corresponding plugins mentioned above.
It is pursued to use as much standardized software components as it is possible. This practice should minimize the code maintenance and should produce a better code quality with less bugs to ease further development. Furthermore it minimizes the need for developer resources to enhance the existing functionality of underlying frameworks. The following Sections include a short overview of the used external features to achieve the desired goals beside the usage and implementation details of the presented approach in Chapter 3. The presented implementation is based on the Eclipse Version 3.8.1 with the corresponding build id: M20120914-1540.

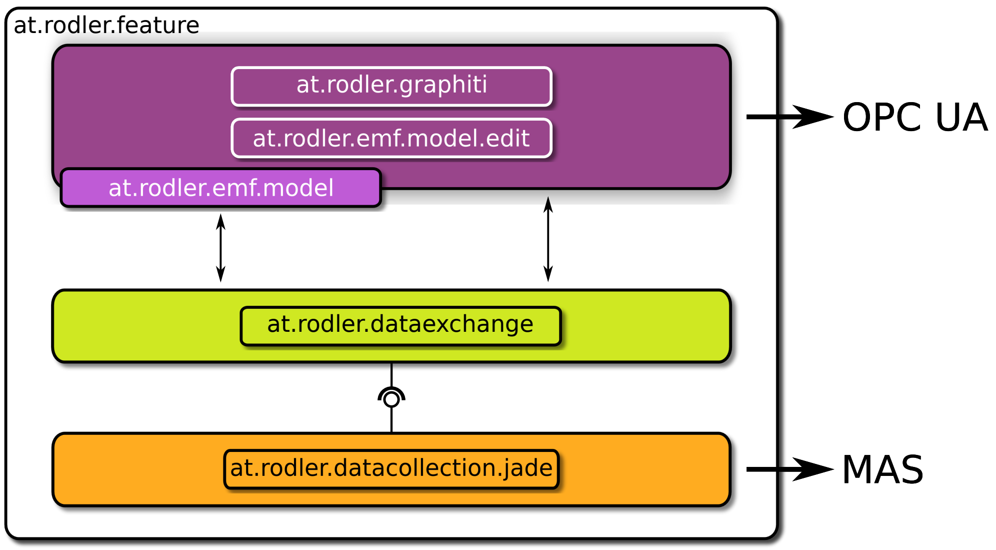


Figure 4.8: The application is splitted into several plugins which are allocated to a specific layer depending on its responsibilities. All plugins are combined to a feature. The middle layer offers an extension point for data collection interfaces.

4.5 Data Model

An important component of the design approach is the generation of the *Data Model*. This should be achieved without much effort by the developer. Eclipse provides amongst its projects (a full list of all projects is available under [eclb]) the Entity Modeling Framework (EMF) that offers - in combination with other EMF based frameworks - a well developed base for the needed features of data handling and visualization tasks. The EMF framework is an Eclipse based modeling framework which automates code generation based on information that can be provided through several different sources (see Figure 4.9). Furthermore the Eclipse Modeling Framework Technology (EMFT) project provides a rich set of components to create, edit and maintain EMF models within a graphical environment. The EMF Core part of the EMF projects consists of three major parts [emf]:

EMF core features contain a meta model for describing EMF models. One of

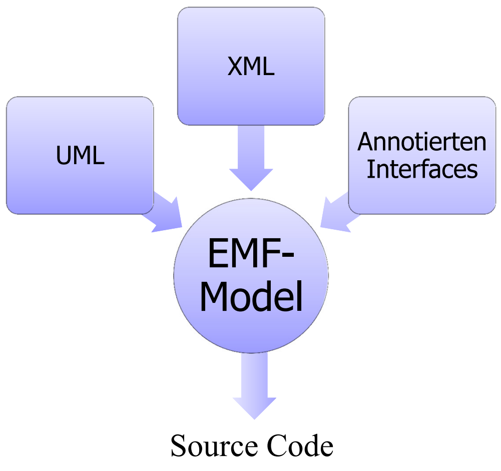


Figure 4.9: This Figure illustrates possible sources which can be used to generate code with the EMF framework.

the main components are (Figure 4.10 illustrates the component mapping) `EClass`, `EAttribute`, `EReference` and `EDataType`. They already cover a great part of the needed meta data components to create an EMF meta model.

EMF.edit contains several features which eases the manipulation of EMF models. This includes predefined JFace[11] components and property sheets, label providers and property source support for the generation of a generic UI that allows the modification of a generated EMF model. A further part is a command framework that automatically supports undo and redo of data manipulations.

EMF.codegen compromises all features that support code generation of:

- the models itself which includes interfaces, class implementations,...
- Adapters which adapt the model classes for editing and illustration
- Editors which support the modification of model elements

Write access on entities is executed through the EMF Transaction Model[12] which prohibits access violations through concurrent write operations at attributes. Additionally the Extended Editing Framework (EEF) - currently

[11]Eclipse UI Toolkit that supports the model view controller pattern
[12]http://eclipse.org/modeling/emf/?project=transaction

part of the EMFT project - is used to generate visual editing components and property sheets based on SWT[13] and JFace to manipulate the data contained in the *Data Model*.

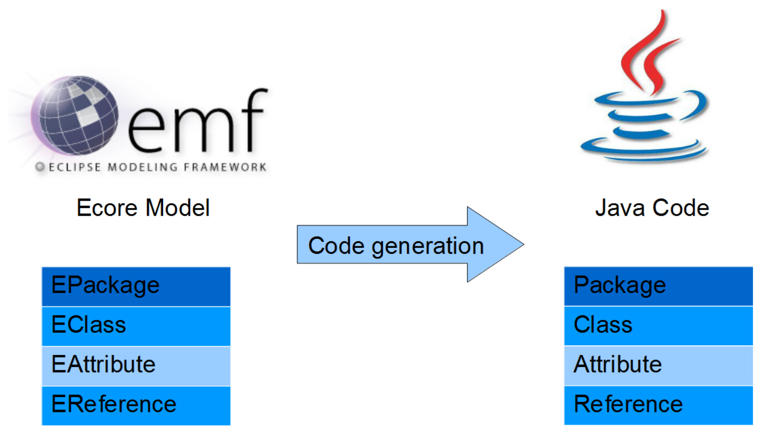


Figure 4.10: The diagram shows the mapping between EMF classes end the equivalent java classes [Hel].

Figure 4.11 illustrates the class diagram that is implemented with the EMF framework and based on the concept presented in Section 3.1.2. It is extended with following objects:

- `Valve`, `Tank` and `Pump` are application specific agent implementations that have a physical component and can be connected within a diagram. They are derived from the `PhysAgent` class and their graphical representation is defined in the XML configuration file.
- `TaskAgent`, `OrderAgent`, `WorkAgent` and `FailureHandlingAgent` are functional agents which means that their counterpart in an MAS does not have a physical component and they do not have the ability to be connected with other agents.

The abstract class `InspectableObject` used in the design approach (see Sec. 3.1.2) has not been implemented because the needed features to support the observer pattern are implemented automatically by EMF at the generation of the source code. Furthermore all instances of an EMF domain model are kept together by an `EResource` object that keeps references of all created objects.

[13]Standard Widget Toolkit

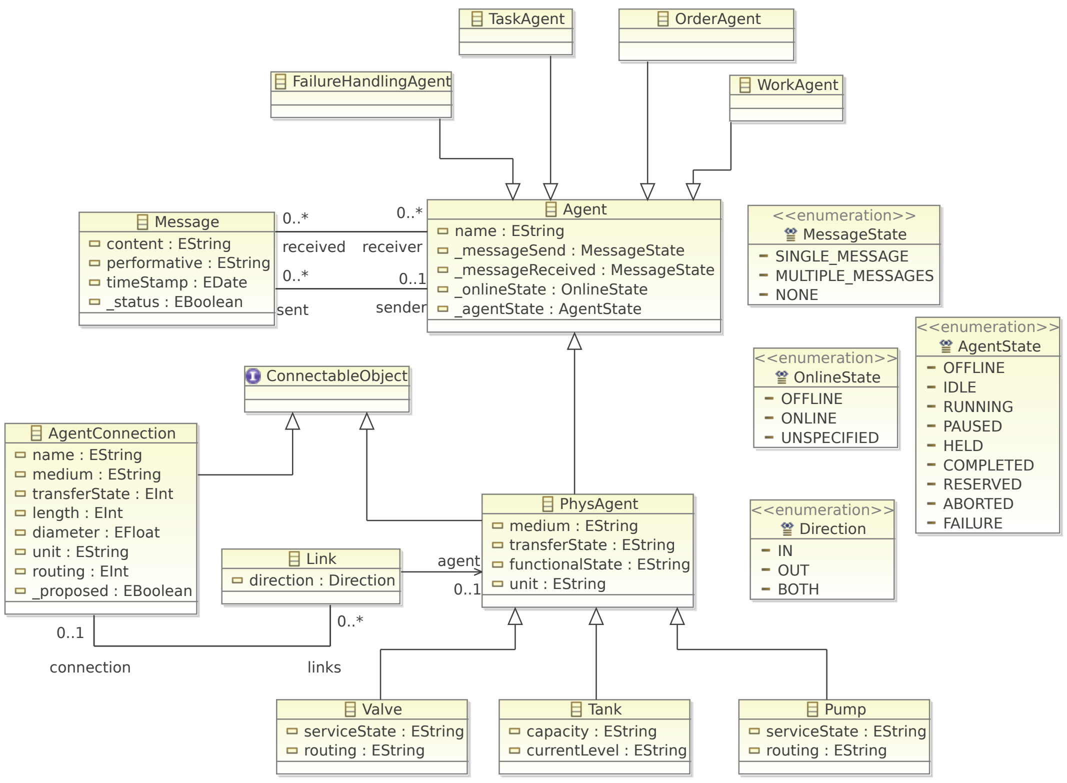


Figure 4.11: The diagram represents the data model created with the EMF-Editor in Eclipse.

This allows a simple access to all objects in different parts of the application (an example of gathering objects from a resource can be seen in Listing 4.1) and makes the creation of a `ModelManager` (presented in the design approach at Sec. 3.1.2) obsolete.

```
/**
 * Searches for objects in a diagram with the given EClass.
 *
 * @param diagram Diagram
 * @param eclass EClass of the objects which should be
    searched for
 * @return List of objects with the corresponding EClass
 */
public static List getObjectList(Diagram diagram, EClass
    eclass) {
```

```
 9     Collection<Object> listObj;
10     //get Resource
11     Resource r = diagram.eResource();
12     //retreive objects
13     listObj = EcoreUtil.getObjectsByType(r.getContents(),
           eclass);
14
15     return new ArrayList(listObj);
16 }
```

Listing 4.1: Any kind of objects derived from defined EClass (e.g. Agent, AgentConnection, ...) can be gathered through a simple command from a resource (line 13) The whole functionality - to gather objects from a diagram - is encapsulated into a helper method.

The following workflow generates the source code for the entire *Data Model*:

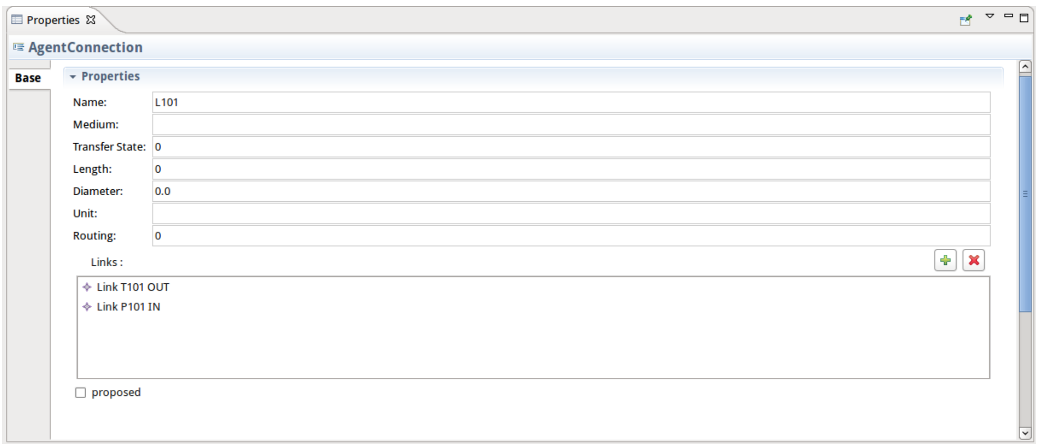
- Generation of the UML diagram within the provided EMF editor.
- Generation of the model source files through EMF features.
- Generation of the edit code through EMF features.
- Additional generation of the properties pages through EEF. For development purposes all features have been left within the properties pages. In practice at least internal features would be removed.

The *Data Model* respectively the instances of the contained classes can be seen as key turning point of all agent relevant data. The only source code modifications that have been performed by hand - at the generated source code by EMF - concerns the `toString` method (see Listing 4.2) which is used as central access point to gather the textual content representation of a class. The method is marked as modified by removing the `@generated` entry in the comment of a method (see line 5). Further modifications of the class diagram and new generation of modified source code through EMF to not override manual code modifications. This can be seen as important feature and allows the extension of generated source code and further administration of the class diagram through the provided graphical EMF diagram editor and the usage of source code generation utilities provided by the framework.

```
/**
 * <!-- begin-user-doc -->
 * Customized toString method
 * to represent the content of a message in the right
 * manner.
 * <!-- end-user-doc -->
 *
 * @generated
 */
@Override
public String toString() {
    if (eIsProxy())
        return super.toString();

    StringBuffer result = new StringBuffer();
    // add sender
    if (sender != null) {
        result.append(sender.getName());
    }
    // Add all receivers
    if (receiver != null) {
        if (!receiver.isEmpty()) {
            result.append("->");
            for (Agent agent : receiver) {
                result.append(agent.getName());
                result.append(",");
            }

            result.deleteCharAt(result.length() - 1);
        }
    }
    result.append("(");
    //add performative
    result.append(performative);
    //add content
    result.append(", ");
    result.append("content: ");
    result.append(content);
    result.append(')');

    return result.toString();
42 }
```

Listing 4.2: The listing illustrates the modified `toString` method of the `MessageImpl` class generated by EMF. The `@generated` entry in line 5 has to be removed to prevent the method of overriding through EMF source code generation.

As already mentioned above all data content can be modified through property sheets generated by the EEF framework (see Figure 4.12 for an example).



Name:	L101
Medium:	
Transfer State:	0
Length:	0
Diameter:	0.0
Unit:	
Routing:	0

Links :
- Link T101 OUT
- Link P101 IN

☐ proposed

Figure 4.12: This illustration represents the property sheet for a selected `AgentConnection` generated through the EEF framework. The headline *AgentConnection* at the top is provided through a customized implementation of the `PropertiesEditionSelection` class.

But to be able to use their features with the Graphiti framework (see Section 4.8 for details) several modifications had to be fulfilled:

- The creation of a customized property sheet label provider that represents the correct name of the corresponding domain model member of an entity (e.g. Tank for a tank object).
- Adaption of the `contributorId` to the corresponding id used in combination with Graphiti (see Section 4.8).
- The implementation of a customized object filter which only accepts objects of the Graphiti editor that have an underlying `EObject` as business object behind a pictogram element and

- a customized `PropertiesEditionSelection` class which returns the corresponding domain model entity to the selected object in the editor. The two last points are based on the work done at [SeB].
- The generated property sheet class (`LinkPropertiesEditionComponent`) of the `Link` model had to be modified due to concurrent write transaction violations. Furthermore a manual call of the update feature was required to update the content of the visual editor.

The first four described modifications have also influenced the extensions (which are defined in the plugin.xml file) that have been automatically generated by the EEF framework.

4.6 Data Collection Layer

This layer implements the provided interfaces defined in the *Data Exchange Layer*. Furthermore it uses the provided extension point of the upper layer to register the implementation of the `DataCollector` as extension (see Listing 4.3 for details) that provides a connection to the JADE multi-agent system.

```
<extension
  point="at.rodler.dataexchange.dataCollector">
          <dataCollector
                class="at.rodler.jade.datacollection.
                    datacollector.JadeDataCollector"
               masName="JADE">
          </dataCollector>
</extension>
```

Listing 4.3: The extension for the defined extension point is defined in the plugin.xml file of a plugin. The definition contains the fully qualified class name (attribute `class`) and the name of the supported MAS (attribute `masName`).

The JADE framework already provides additional plugins that ease the integration and connection of an agent within an OSGI[14] framework (Eclipse is based on OSGI) to the JADE platform (see [QC] for details). A sample implementation of a Sniffer [sni] is used as base for a customized Sniffer implementation. It has been modified according to the requirements of this layer.

[14]Open Services Gateway Initiative

```
@Override
public void start() throws DataCollectorException {
    //throw exception if sniffer already exists
    if (sniffer == null) {
        BundleContext context = Activator.getBundleContext();
        // Retrieve the JRS service
        String jrsName = JadeRuntimeService.class.getName();
        ServiceReference<JadeRuntimeService> jrsRef = (
            ServiceReference<JadeRuntimeService>) context
                .getServiceReference(jrsName);
        JadeRuntimeService jrs = (JadeRuntimeService) context
                .getService(jrsRef);

        try {
            //create sniffer agent and register listeners
            sniffer = new Sniffer();
            sniffer.addAgentEventListener(new
                IJadeAgentEventListener() {
                .
                .
                .

            });
            sniffer.addMessageListener(new IJadeMessageListener
                () {
                .
                .
                .
            });

            // Make JADE accept the new agent
            jade.wrapper.AgentController ac;
            ac = jrs.acceptNewAgent(GRAPHITI_CONTAINER, sniffer
                );
            ac.start();
        } catch (Exception e) {
            //exception occurred during creation and
                registration of agent
            throw new DataCollectorException(e);
        }
    } else {
            throw new DataCollectorException(ERROR_ALREADY_STARTED
                );
        }
}
```

Listing 4.4: This Listing contains the source code from the start method of the implemented `DataCollector` interface provided by the upper layer.

The `start` method of the `JadeDataCollector` is responsible for the creation and registration of a sniffer agent at the JADE platform. The `JadeRuntimeService` takes care about the creation of a container and the registration of the agent (line 30 at Listing 4.4) after the creation of a `Sniffer` and the corresponding listeners for notifications (line 15 to 26). The methods of the listeners simply forward the notification to an internal method that translates the JADE `ACLMessage` respectively the `AID` of an agent into a defined format that can be managed by the upper layer. This procedure releases all other parts of the application form dependencies concerning the JADE framework. The message content of a JADE message is transfered into a `HashMap` to reach a higher flexibility for the implementation of new features and supported message content (as proposed in 3.1.4). The dataflow from an upcoming event within JADE for a registered agent (the `Sniffer` has to be registered for all agents for which notifications should be received) until the information arrives at the *Data Exchange Layer* is described in Figure 4.13.

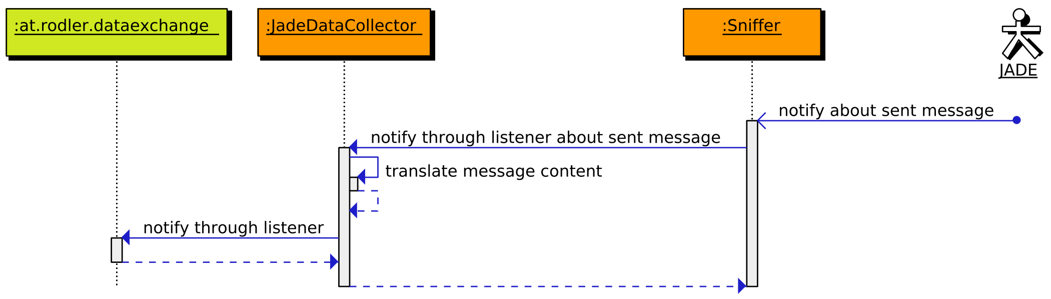

:at.rodler.dataexchange
:JadeDataCollector
:Sniffer
JADE
notify about sent message
notify through listener about sent message
translate message content
notify through listener

Figure 4.13: The sequence diagram illustrates the notification path from JADE to the *Data Exchange Layer* through the implemented `JadeDataCollector`. The implemented `DataCollector` can be seen as gateway and encapsulates the JADE framework from the rest of the application.

4.7 Data Exchange Layer

The plugin *at.rodler.dataexchange* contains all necessary classes that are needed to provide the interface to the *Data Collection Layer* described in Section 3.1.2. This implies a provision of the `DataCollector` and its corresponding interfaces. The implementation follows the approach proposed in Section 3.1.4. Furthermore this layer uses the extension point feature of the eclipse framework to register an extension point that allows the registration of multiple `DataCollector`s of different multi-agent system interfaces.

The Extension Point requires following information for registration by other plugins:

masName - the name of the multi-agent system to which the registered extension belongs to.

class - the full qualified class name of the implementation from the `IDataCollector` interface.

The ability to start online monitoring is also provided by this layer. It registers two buttons to connect and disconnect from an MAS at the Eclipse platform which is shown in the toolbar of the IDE (see Figure 4.14 for the graphical representation). At the activation of this button all registered ex-

(a) Connect (b) Disconnect

Figure 4.14: These buttons are provided to connect and disconnect to a multi-agent system.

tensions are gathered from the system (see Listing 4.5 at line 2) and - if there exists more than one registration - visualizes available `Datacollector`s to the user to select the corresponding one (from line 17 to line 38). The name of the MAS is used for the selection of an extension (the `masName` attribute of the extension contains the name of the MAS to which the `DataCollector` corresponds to). After choosing the MAS the corresponding `DataCollector` will be created (see line 37).

```java
//get registered datacollector implemenations
IConfigurationElement[] collectors = HelperUtil
      .getDataRegisteredDataCollectors();

//registrations exist
if (collectors.length != 0) {

   IDataCollector dc = null;

   // there exists only one registration
   if (collectors.length == 1) {
      //create datacollector
      dc = HelperUtil.getDataCollector(collectors[0]);
   } else {
      //multiple registrations exist
      //create selection dialog
      ElementListSelectionDialog dialog = new
         ElementListSelectionDialog(
            PlatformUI.getWorkbench()
                  .getActiveWorkbenchWindow().getShell(),
            new LabelProvider() {

               @Override
               public String getText(Object element) {
                  //show MAS name as label
                  IConfigurationElement config = (
                     IConfigurationElement) element;
                  return config.getAttribute("masName");
               }
            });

      dialog.setBlockOnOpen(true);
      dialog.setMultipleSelection(false);
      dialog.setElements(collectors);
      if (dialog.open() == ElementListSelectionDialog.OK) {
         Object[] result = dialog.getResult();
         if (result.length == 1) {
            //create selected datacollector
            dc = HelperUtil
                  .getDataCollector((IConfigurationElement)
                     result[0]);
            }
        }
    }
```

Listing 4.5: This is an extract of the source code from the connect button (see Figure 4.14(a)) handler. All registered `DataCollector` extensions are fetched. If there is more than one registration than the user will be able to select the corresponding MAS by its name. The instance will be created after a successful selection.

The transfer of information provided by the multi-agent system - through sniffed messages or notifications - is achieved through listeners registered at the `DataCollector`.

- `AgentEventListener` is derived from a default implementation of `IAgent-EventListener`. It is responsible to transfer agent relevant state notifications - related to the MAS - to the *Data Model*.

Following classes implement the `IMessageListener` interface and gather their information from sniffed messages:

- `AgentStateListener` - transfers state updates of agents to the corresponding entity in the *Data Model*. The identification of relevant messages containing state changes is based on the information presented in Table 4.1.

- `MessageListener` - embeds sniffed messages of mapped agents into the *Data Model* and sets the corresponding internal attributes of each agent (`_messageSent` or `_messageReceived` - depending on the demands) and sets an `_status` flag of those messages which are new. A `MessageTimer` is used to reset changed attributes after a specific amount of time. The listener uses a `HashMap` to recognize "duplicate" messages (as proposed in Sec. 3.1.4. Those messages can occur if they are sent and received by observed agents. To prevent double entries a hashcode is generated and stored in the mentioned map. If a second event with the same message occurs only the internal attributes will be set without the creation and saving of the message itself.

- `RouteProposalListener` - is responsible to handle route proposals sent by one of the mapped agents. It changes the `_proposed` attribute of all listed connections and activates an `AttributeTimer` which resets the affected attributes after a specific amount of time. This is a universal timer and can be used with any attribute of the *Data Model*. The identification of relevant messages containing route proposals is based on the information presented in table 4.1.

All above mentioned listeners use the transaction framework provided by the EMF Model Transaction (EMF-MT) project to execute their write operations at entities who are part of the domain model. This should avoid access violations through concurrent write access at attributes.

The recognition and and forwarding of a route proposal created by one of the sniffed agents and processed by the `RouteProposalListener` is used as showcase to explain the procedure processing an incoming event. Listing 4.6 illustrates the treatment of an incoming "sent message" event. First of all at line 7 the method will check if the incoming message is from interest through the equation of the ontology attribute (its expected value is defined in table 4.1). If the value matches with the expected value the method will gather all proposed agent connections from the message content (at line 8 and 9). At last the `_proposed` flag will be set at all contained connections (from line 12 to 14).

The method named `setRouteProposed` is presented in Listing 4.7 and sets the corresponding flag at the affected connection. The Transaction Model framework is used to change the value (from line 11 to 17). From line 20 to 26 a `AttributeTimer` is created which resets the flag after a specific amount of time. Only one timer will be created in this time span for each connection independently of the incoming proposals. The task will delete itself from the map after it has been executed.

```
 1  @Override
 2  public void sentMessage(Map<String, String> msg, Date date)
 3  {
 4      String ontology = msg.get(MessageProperties.ONTOLOGY);
 5
 6      // check ontology
 7      if (ontology != null) {
 8          if (ontology.equals(MessageProperties.
                ONTOLOGY_ROUTE_PROPOSAL)) {
 9              String[] proposedRoutes = msg.get(MessageProperties
                    .CONTENT)
10                      .split(MessageProperties.
                            SEPARATOR_ROUTE_PROPOSAL);
11
12              // walk trough all contained connections
13              for (String route : proposedRoutes) {
```

```
14                  setRouteProposed(route, true);
15              }
16
17              editor.refreshContent();
18          }
19      }
20  }
```

Listing 4.6: The `sentMessage` method of the `RouteProposalListener` handles incoming route proposals and forwards the information to the *Data Model*. Attribute values are modified in the `setRouteProposed` method called at line 13 for every proposed agent connection (explained in Listing 4.7).

```
private void setRouteProposed(String routeName, final
    Boolean value) {
    final AgentConnection connection = HelperUtil.
        getConnection(diagram,
            routeName);

    // check if connection exists
    if (connection != null) {
        // get editing domain
        final TransactionalEditingDomain domain = editor.
            getEditingDomain();

        // create command
        Command cmd = domain.createCommand(
                SetCommand.class,
                new CommandParameter(connection, ModelPackage.
                    eINSTANCE
                        .getAgentConnection__proposed(), value));

        // execute command
        domain.getCommandStack().execute(cmd);

        // only create timer if no one exists
        if (!mapTimer.containsKey(routeName)) {
            AttributeTimer task = new AttributeTimer(connection
                , editor,
                    ModelPackage.eINSTANCE.
                        getAgentConnection__proposed(),
                    mapTimer, new Boolean(false));
24            mapTimer.put(connection, task);
25
26            timer.schedule(task, PROPOSAL_DELAY);
27        }
28    }
29 }
```

Listing 4.7: This method is called within the `sentMessage` method of the `RouteProposalListener` and changes the `_proposed` flag of a `AgentConnection` to the given value by using the EMF Transaction Model (line 11 to 17). Furthermore a `TimerTask` is created - only if no one exists for the specific connection - which resets the flag after a specific amount of time (from line 20 to 26).

All other listeners use the same approach as illustrated with the example above and as proposed in Section 3.1.4.

4.8 Graphical User Interface

This Section will describe the implementation of the *Visualization Layer* with the aid of the Graphical Tooling Infrastructure (Graphiti) project. It will give a short introduction to the Graphiti project and furthermore detailed information about the implementation to reach the desired goals presented in Section 3.1.5.

Several frameworks have been tested to find the ideal framework for the desired demands. Particular attention was also paid to the Graphical Editing Framework (GEF) and the Graphical Modeling Framework (GMF) and were also tested through the implementation of a simple editor but Graphiti (which is based on GEF and Draw2D but hides its complexity) fits best do the desired demands which are reflected in the main goals of the project [graa]:

- to hide the platform specific technologies from the developer
- to provide a rich set of default implementations within the framework
- and to provide a defined look and feel that has been designed in close cooperation with usability specialists.

These goals can be seen as benefit for the realization of the designated objectives defined in 3.1.5.

The Graphiti project was founded by developers of the SAP AG and was at first only designated for internal use. In the year 2007 SAP decided to give the project under the Eclipse Public License and handed the source code over to the Eclipse Foundation [jax]. Nevertheless over 90 percent of the commits within the last three months are still done by employees of SAP (there have been 65 commits within this time span) [graa]. The activity of an project can also be seen as an indicator of its health and potential life time and should be observed prior of the decision for a particular open source framework.

Graphiti's basic approach is to isolate the graphical and logical information. Figure 4.15 (b) illustrates the separation of concern. The domain model based on EMF is responsible for the logical information and linked through the Link Model with the corresponding Pictogram Model that keeps the graphical information (but is still platform independent) for the visualization of an object. Due to the integrated support of EMF models Graphiti takes also care about the saving and restore of the diagram and domain model information. This could also be separated if other storage destinations (e.g. database) are required. As illustrated in Figure 4.15 (a) inputs by the user are forwarded by Graphiti to the `DiagramTypeAgent`. This class contains the `DiagramTypeProvider` and corresponding `FeatureProvider` with the customized features. Thanks to Graphiti there are already plenty of standard implementations of features that - if at all - only need to be extended. In this case the `DiagramTypeAgent` provides the customized `FeatureProvider` which is responsible to return the adequate feature for the given user input.

```
@Override
public ICreateFeature[] getCreateFeatures() {
    // automatic generation of the create feature entries
        depending on the
    // config
    try {
        // get configuration
        IMemento memento = ConfigHelper.getReadRoot();

        List<ICreateFeature> eList = new ArrayList<
            ICreateFeature>();
        // search for agents in the data model which are
            defined in the
        // graphical configuration
        for (IMemento child : memento.getChildren(
            ConfigConstants.AGENT)) {
                String clsName = child
                        .getString(ConfigConstants.
                            AGENT_ATTRIBUTE_NAME);
                EClass cls = (EClass) ModelPackage.eINSTANCE
                        .getEClassifier(clsName);
                // if the agent exists in the graphical
                    configuartion and in the
                // model an Create Feature will be created
                if (cls != null) {
                    eList.add(new CreateAgentFeature(this, cls));
                }

            }

            return eList.toArray(new ICreateFeature[0]);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;

    }
```

Listing 4.8: The `FeatureProvider` reads the graphical definition and will create a `CreateAgentFeature` for the specific agent if it exists in the definition in the *Data Model*. All agents that occur in both definitions are available for the user in the diagram.

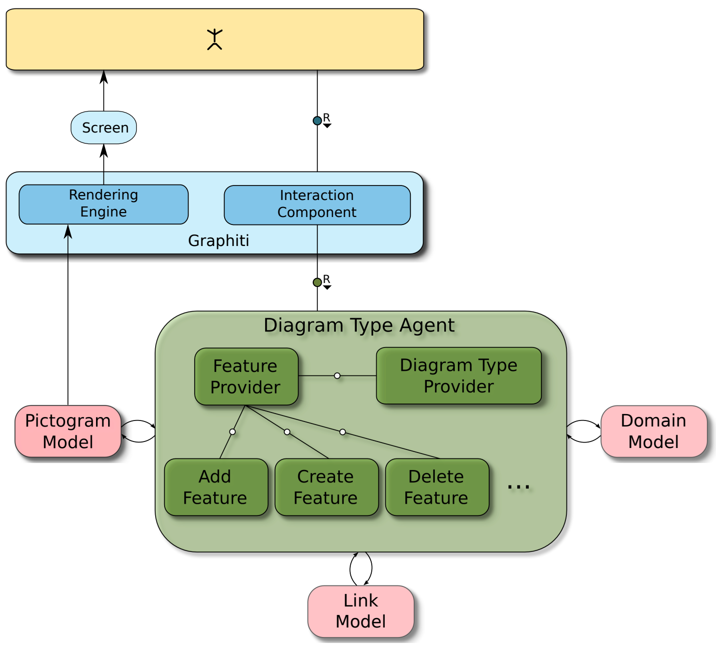
The implemented `FeatureProvider` creates Create Features dynamically by matching the names of agents defined in the graphical definition stored in an XML file (see Listing 4.9 for an example) with the *Data Model* (see Listing 4.8 for details). The graphical definition has mainly followed the approach presented in Section 3.1.5 and has been enhanced to be able to use additional graphical capabilities provided by Graphiti (as shown in Listing 4.9 the `before` and `after` attributes allow the creation of rounded edges).

```
<agents>
        <agent name="Tank">
                <shape name="polygon" filled="false"
                    lineWidth="2">
                        <point x="0" y="0" before="20" after
                            ="20" />
                        <point x="0" y="100" before="20"
                            after="20" />
                        <point x="100" y="100" before="20"
                            after="20" />
                        <point x="100" y="0" before="20"
                            after="20" />
                </shape>
        </agent>
</agents>
```

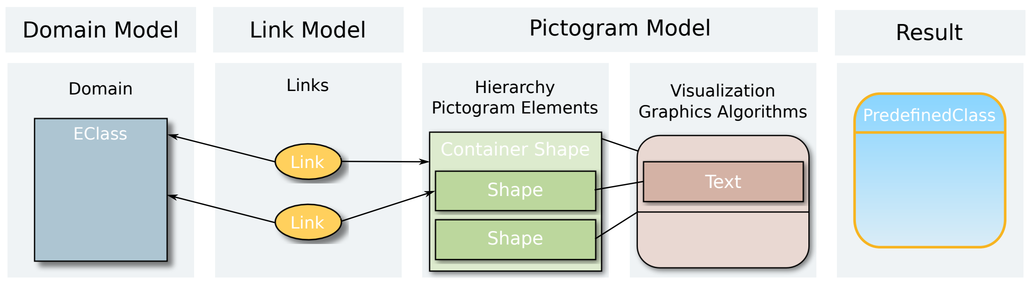
Listing 4.9: The Listing shows an example of a graphical definition from an agent in the definition file. The agent tag holds a `name` attribute which contains the agent name. The graphical definition is done with shapes. The name of the shape defines the type of the graphic (in this case a polygon). Child objects of the polygon define the edges of it (before and after can be used to form rounded edges).

In following the used features are described in their general functionality:

- Create Features are responsible for the creation of the model entity. It creates an entity and if necessary can request user interactions (to request a name for an object). After a successful creation of the business object the feature forwards the information to the Add Feature (only if the business object has a graphical representation).
- The Add Feature creates the graphical part of a new object. It initializes the so called shapes which represent different kinds of graphical objects (rectangle, ellipse, polygon, text, . . .). The initial graphical representation of a business object is created in this Feature. Furthermore it links business objects with the corresponding graphical objects.
- Update Features are responsible to transfer changes from the *Data Model* to the Pictogram Model. At first the Feature can decide whether an update is needed. If so it transfers necessary data from the business object to its graphical object. this is done automatically or by user request depending on the update strategy.
- The Layout Feature takes care about the accurate layout of the graphical representation. In most cases this Feature will be called if the graphical object is resized and the Shapes' sizes have to be recalculated.
- The Remove Feature deletes the graphical objects from a diagram but the corresponding business objects remain unchanged.



(a) Basic architecture of Graphiti



(b) Graphiti Diagram Type Agent

Figure 4.15: Figure (a) illustrates the basic architecture and information flow of the Graphiti framework. (b) represents the relations between the EMF Domain Model, the Link Model that holds the connection between Domain Model and Pictogram Model which is responsible for - an OS independent - graphical representation [grab].

- The Delete Feature deletes the business object of a corresponding graphical object. The according remove Feature is normally called in this Feature to delete the corresponding graphical part.

The default implementations of the Features (especially remove and delete) are fair enough to cover all needed standard functionality. Only by special purposes (as by agent connections) a custom implementation is needed.

The procedure to create new objects within the visualization is exemplarily explained by the creation of a new agent that is done by the execution of following steps:

- The user creates a request for creation of an agent.
- Graphiti forwards this request to the `DiagramTypeAgent` which in addition returns a customized implementation of a `FeatureProvider`.
- The `FeatureProvider` returns the corresponding Feature which is the `CreateAgentFeature` for the creation of an agent.
- The `CreateAgentFeature` creates the requested data model entity (e.g. tank, pump,...) and requests a name through an input dialog from the user. In the end the `AddAgentFeature` is called to create the graphical representation of the entity (the calling of one feature within another one is done indirectly).
- `AddAgentFeature` creates the appropriate graphical objects of the agent. The graphical definition is gathered from an XML file that contains the definition. At last the Add Feature calls the `LayoutAgentFeature`.
- The `LayoutAgentFeature` calculates the corresponding sizes of the agent graphic depending on the definition and the drawing created by the user at the agent generation. This Feature is also responsible for the correct rotation of an object according to the approach in Sec. 3.1.5. The objects current rotation status is stored as property through a provided functionality of Graphiti.

The notification about incoming or outgoing messages follows the approach presented in Sec. 3.1.5 and uses decorators - which are automatically embedded into the drawing of an agent - to notify the user about incoming and outgoing messages. The used graphics are illustrated in Figure 4.16. The `ToolBehaviorProvider` is responsible for the visibility of the notification images. Depending on the values of `_receivedMessage` respectively `_sentMessage` attribute and the number of `Messages` related to the agent with a `_status` value of true is used to identify the corresponding decorators. The appearance of multiple decorators results in a string of decorators side by side.

Additionally to the provided diagram editor following views have been added:

- A MessageView which lists all received and sent messages of an agent selected in the diagram.
- A PropertyView which is able to recognize an OPC UA path stored as value of an attribute. If the visualization is in online mode the view will list all attributes of an entity selected in the diagram editor. Static values will be illustrated as plain text and OPC UA path values will be used to create connections to an OPC UA Server to gather the corresponding values stored with the given path. This allows direct access to values used in the PLC in behind of an specific agent. The separation between static values and OPC UA path variables is done by the `pronto.comm` framework provided by the ACIN Institute at the Vienna University of Technology.

(a) Message sent

(b) Message received

(c) Multiple messages sent

(d) Multiple messages received

Figure 4.16: (a) and (b) represent a single outgoing (red) and incoming (blue) message. Multiple messages are illustrated through (c) and (d) by using multiple envelopes with the same colour as for single messages.

In the following additional functionality will be described which is provided through buttons registered as extensions. The diagram offers the possibility to show the agent state in online mode through colours but is unable to visualize the state within the MAS and the logical state of an agent at one time. This problem is bypassed through a switch (illustrated in Figure 4.17(c)) which changes the actual view type of the visualization. Through manually triggering an update event for all contained agents the colour of the agents changes to the actual view type (MAS state or logical state). The creation of a diagram is provided through a registered button with Figure 4.17(b). This eases the creation of a new diagram by the user. The registered button with the Figure 4.17(a) represents the creation and export of the ontology into an XML file. The used framework already provides additional functionality which simplifies the generation of XML files. Listing 4.10 lists an abstract from the source code which is responsible for the generation of the ontology configuration file. The `Resource` holds a reference to all objects of the *Data Model* which simplifies the generation of a list with all agents (line 6). The `XMLMemento` class provides all necessary functionality to create a well formed XML file (line 9). From line 15 to 26 the tags are created for all automation agents contained in the diagram. Internal attributes should not be included in the ontology configuration. They are sorted out by searching for the "_" prefix at each attribute (line 21). All other attributes are added as attributes to the XML tag.

```
1  //get resource from diagram
2  Resource r = diagram.eResource();
3  //get EClass instance of type PhysAgent
4  EClass cls=ModelPackage.eINSTANCE.getPhysAgent();
5  //retrieve objects with the given type (agent) from the
      content of the given resource
6  List<Agent> listA = EcoreUtil.getObjectsByType(r.getContents
      (), cls);
7
8  //create XML file with "ontology root tag
9  XMLMemento memento = XMLMemento.createWriteRoot("ontology");
10
11 //create "components root tag
12 IMemento componentsM = memento.createChild("components");
13
14 //run through all agents
15 for (Agent agent : listA) {
16     //create for each agent a new tag - named by its class
          name
17     IMemento agentM = componentsM.createChild(agent.eClass()
18             .getName());
19     //insert class attributes as tag attributes - except
          internal attributes (leading "_")
20     for (EAttribute attribute : agent.eClass().
          getEAllAttributes()) {
21         if (!attribute.getName().startsWith("_")) {
22             Object value = agent.eGet(attribute);
23             agentM.putString(attribute.getName(),
                    value != null ? value.toString() : "");
        }
    }
}
```

Listing 4.10: The Listing contains an abstract of the method responsible for the generation of the ontology configuration file. Encapsulated functionality has been embedded for better confirmability.

(a) Export topology configuration

(b) Create new diagram

(c) Switch between Agent State and MAS State view

Figure 4.17: The illustrated images are used for buttons registered at the Eclipse IDE. The framework integrates them automatically into the existing toolbar. The images are from existing and used Eclipse projects and reused for similar functionality to stay in line with the Eclipse design and for a better understanding of the meaning from the buttons.

4.9 Summary

This Chapter represented the implementation of the approach proposed in Chap. 3. At first a short introduction was made from the basics of JADE (Sec. 4.1) followed by a presentation of the test equipment at the Odo Struger Laboratory (Sec. 4.2). Out of several technologies - Eclipse SDK has been chosen as underlying technology to implement the presented approach. The decision is based on analysis of the licenses, contributors and popularity of available frameworks (see Sec. 4.3). The plugin pattern used by this platform supported the development of a tier based application. Furthermore several other frameworks (e.g. Graphiti, EMF,. . .) provided by the Eclipse Foundation have been found as suitable to maintain the development of the given approach. This should decrease the costs for code maintenance and enhance the code quality of the developed application. The *Data Model* (Sec. 4.5) of the application could be generated with minor effort by using the EMF framework. Property sheets that are available for data modification could also be generated through a subproject (EEF). The *Data Collection Layer* (Sec.

4.6) represents the link to the JADE framework used at the test plant at the Odo Struger Laboratory. Due to the designed modularity of the application this layer can be simply exchanged to connect to another multi-agent system. The middle layer (*Data Exchange Layer* - Sec. 4.7) passes the information through from the lowest layer to the *Data Model*. The Graphiti framework has been used to integrate the GUI (in Sec. 4.8). Plenty of work is already done by the framework because it has integrated support for EMF data models. Furthermore it provides a ready to use diagram editor that can be used to create topologies with the provided figures. The graphical information is gathered from an XML definition which allows the definition of graphical representations without changing any source code. The same editor is also used to visualize the online state of agents with sniffed data from the underlying MAS that is integrated into the existing *Data Model*. The generation of an ontology XML file is based on the information from the created diagram with the Graphiti editor.

5 Case Study

This Chapter discusses the usage of the implementation presented in Chapter 4 with the test plant at the Odo Struger Laboratory (see Figure 5.1). This includes a report about the processed workflow which should be used in practice by a tutor and students in a course about multi-agent systems. Furthermore the creation of the diagram with the implemented diagram editor will be illustrated and the handling (the behaviour should follow the approach described in Sec. 3.1.5) analyzed. The analysis of the visualization in online mode will follow after the creation of the graphical mapping from the physical plant. The results of the online demonstration will be described in detail. At last a comparison of the written lines of code and the generated will be explained. This also includes a discussion about the effort that would arise by implementing graphical representations pure in Java.

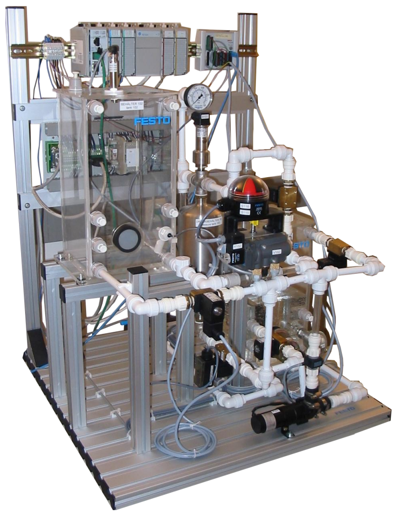


Figure 5.1: Experimental plant in the Odo Struger Laboratory at the Vienna University of Technology.

The following results of different simulation runs are made with a HP Elitebook 8540w (for detailed hardware specification please refer to Table 5.2). The

Table 5.1: System specification of the test system used in conjunction with the test plant.

Operating System:	Microsoft Windows 7 Professional Service Pack 1
System Type:	64 - Bit Operating System
Processor:	Intel Core i5 processor M 520 @ 2.40GHz Codename Arrandale
Graphics:	NVIDIA Quadro FX 880M 1024MB
Network:	Intel 82577LM Gigabit Network Connection
Ram:	DDR3 PC3-10600 SDRAM (1333 MHz) dual-channel memory
Ram Size:	8192 MB
Hard Drive:	Seagate Momentus, SATA-II 3.0Gb/s, 7200rpm
Hard Drive Size:	320 GB
Filesystem:	NTFS

Table 5.2: System specification of the linux based test system.

Operating System:	Ubuntu 12.04.2 LTS Codename precise
Processor:	Intel Pentium M processor Codename Dothan 1.70 GHz 6 Model 13 Stepping
Ram:	DDR PC2700 (166MHz)
Ram Size:	1024 MB
Hard Drive:	Hitachi Travelstar, PATA Interface, 7200 rpm
Hard Drive Size:	60 GB
Filesystem:	ext4

tests where performed with an Eclipse IDE version 3.8.1. For testing purposes JADE with the version 4.1.1 has been used. Details about the particular software framework versions can be seen in Table 5.3. The JADE main container, all agents (physical and functional) and the visualization where running on the same machine. The connection to an OPC UA-Server had been tested separately and was not included in the proof of concept with the test plant. The visualization has also been tested with a Linux based operating system (for a detailed system specification refer to Table 5.1).

But instead of the test plant dummy agents have been used to simulate the communication between agents. The behaviour has been compared with the outcome of the tests with the Windows based operating system. At this comparison no divergent behaviour could be determined and the implemented visualization operated as expected. Also the graphical representation and user

Table 5.3: Software specification of the test system.

Framework	Version	Build id
Eclipse SDK	3.8.1	M20120914-1540
Ecore Tools	1.1.0.201205150811	
Eclipse Modeling Framework - Runtime and Tools		
EMF Code Generation		
EMF Ecore Code Generator	2.8.3.v20130125-0826	R201301250826
EMF Ecore Edit		
EMF Ecore Mapping		
EMF Edit		
EMF ECore	2.8.3.v20130125-0546	R201301250546
EMF Model Transaction Core	1.6.0.v20120328-0001-377-8s734C3E7D15D6B	201205171405
Graphical Editing Framework GEF	3.9.0.201212170307	3.9.0.201212170307
Graphical Editing Framework Draw2d	3.9.0.201212170307	3.9.0.201212170307
Graphiti UI (Incubation)	0.9.2.v20130211-0913	

handling was identical on both operating systems and in spite of the weaker hardware the visualization was operating regular without much delay. To achieve this no source code changes have been required.

To install the test system probably following steps have been processed:

- The graphical representation of the agents (defined in the XML file) has been modified to confirm the ISO standard 14617. This includes a graphical representation of a Tank, Pump, Valve, Task-Agent, Order-Agent respectively Recipe-Agent, Work-Agent and Failure Handling Agent (see Figure 5.2 for the agent illustrations). The drawing of functional agents should also identify possible restrictions of the definition from agents. However the definition from the graphical representations of functional agents - which have the most complex illustrations - could be made without any limitations or workarounds.
- All necessary Plugins have been integrated into a Feature. Also the dependencies to Eclipse Project Frameworks have been included into the Feature. Furthermore an Update Site with the generated Feature has been created.
- The created Update Site with the provided Feature and its Plugins has been uploaded to the student webspace of the Vienna University of Technology.

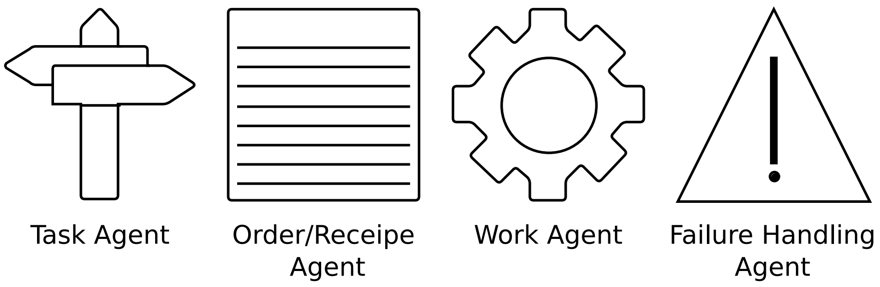


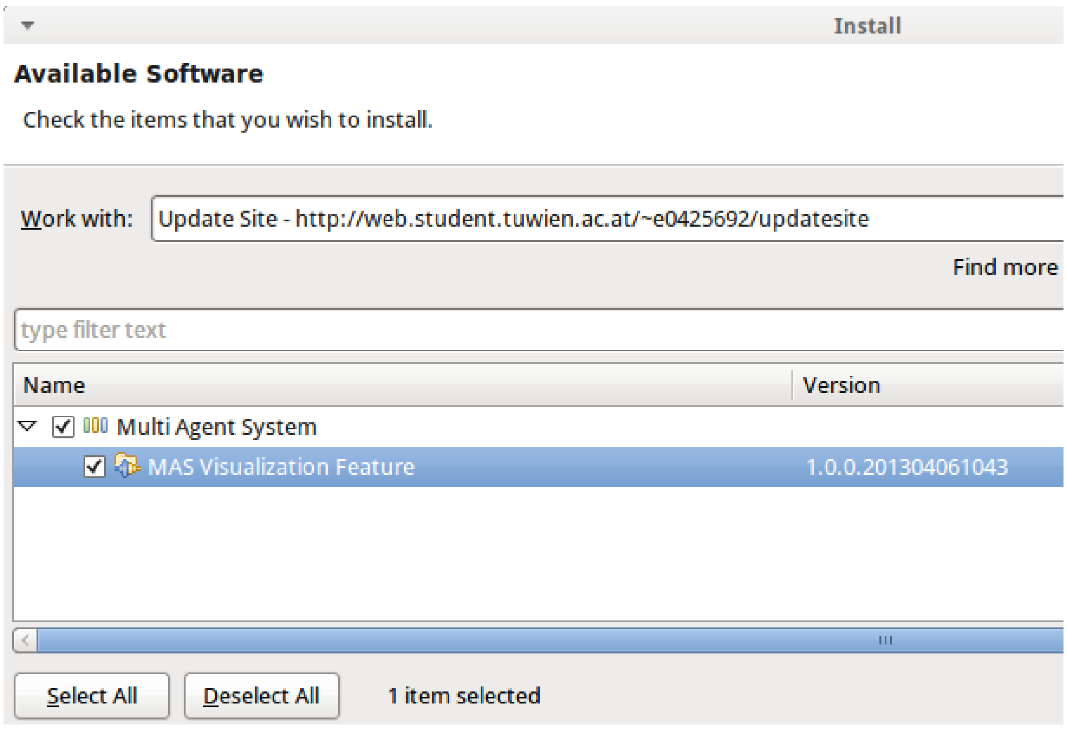
Figure 5.2: This Figure shows the illustrations of the functional agents. The graphical representations also demonstrate the possibilities to draw agents.

- Through the link to the provided Update Site - in this case `http://web.student.tuwien.ac.at/~e0425692/updatesite/` - the visualization Feature could be installed with the aid of the installation wizard of eclipse (see Figure 5.3 for the displayed Update Site information). Due to the fact that all dependencies to other Features had been defined in the Visualization Feature, the installation wizard was also able to download and install all needed dependencies without any need of a user interaction or manual configuration (this will only be successful if all defined and currently not installed dependencies can be found at the Update Sites registered within the Eclipse IDE installation).
- After a restart of the IDE all necessary plugins had been installed successfully without any conflicts and were ready to use by the developer.

The above fulfilled workflow represents the presented procedure in Section 3.2 that illustrates a possible use case from preparation of the visualization by a tutor until the usage of the provided application by a student. The presented approach has been proved as applicable for real world scenarios within university courses. Furthermore the above executed steps could be accomplished without the need of changing any source code (in spite of the graphical definition which is not seen as source code).

5.1 Visualization in Practice

After the preparation of the Eclipse IDE, the tests of the visualization have been started. The sequence diagram in Figure 5.4 illustrates the workflow of



Install

Available Software

Check the items that you wish to install.

Work with: Update Site - http://web.student.tuwien.ac.at/~e0425692/updatesite

Find more

type filter text

Name	Version
☑ Multi Agent System	
☑ MAS Visualization Feature	1.0.0.201304061043

Figure 5.3: The Update Site that has been uploaded to the student webspace is used with the installation wizard of Eclipse to install the multi-agent system visualization and its dependencies at the used IDE.

the test. The generation of the diagram could be done without much effort through the provided wizard. The creation of the illustration itself is - thanks to the Graphiti framework - straight forward. The illustration presented in Section 4.2 is used as template to create the graph in the diagram editor.

Disregarding the functional agents which are not included in the template, the editor enabled the creation of an exact copy of the topology in the sample graphic. The modification of the diagram to achieve a handsome adjustment of the created agents and connections was straight forward and could be easily achieved. Figure 5.5 illustrates the created topology with the added functional agents. The naming of the agents and connections was either done through the input dialog at creation or through the available property pages. Also the configuration of the flow direction that can be clearly identified in the diagram has been configured through the property pages. The creation of the agent diagram could be performed in less than 10 minutes just through the help of visual tools without writing any code. The diagram could be saved, closed and

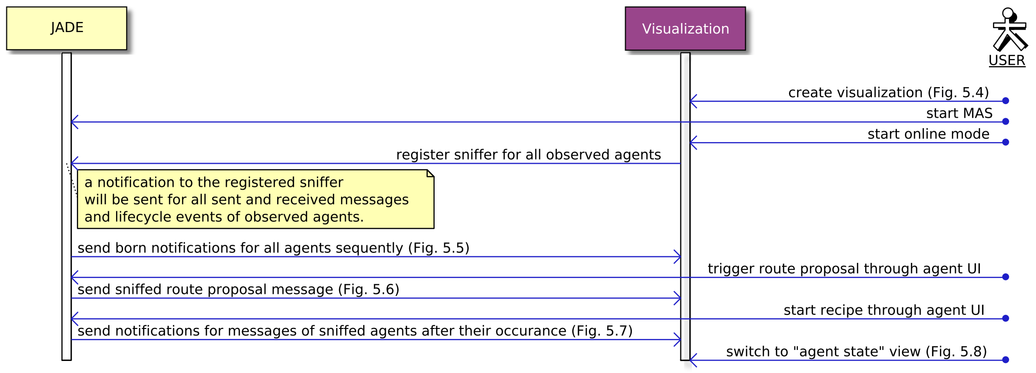

JADE
Visualization
USER
create visualization (Fig. 5.4)
start MAS
start online mode
register sniffer for all observed agents
a notification to the registered sniffer will be sent for all sent and received messages and lifecycle events of observed agents.
send born notifications for all agents sequently (Fig. 5.5)
trigger route proposal through agent UI
send sniffed route proposal message (Fig. 5.6)
start recipe through agent UI
send notifications for messages of sniffed agents after their occurance (Fig. 5.7)
switch to "agent state" view (Fig. 5.8)

Figure 5.4: The sequence diagram illustrates the process of the test case. The references to the corresponding screenshots are added to the diagram.

reopened without any complications with the same adjustment of the objects and with no data loss.

At first the JADE main container and the test plant application has been started. Afterwards the visualization was able to go online without any conflicts. The activation of the online mode has been carried out through the actuation of the *Connect* button at the top of the application (see Figure 4.14(a) for an illustration of the button). The JADE events for existing agents where received successfully after the activation of the online mode. JADE was sending the born agent event sequentially to the sniffer agent of the visualization. The result can be seen in Figure 5.6. Parts of the agents included in the topology are already online (green coloured agents) for others the online event is still missing (red coloured agents). The removed grid in the background increases the visibility of the agent diagram during the online mode. The incoming events can be seen in the Event Log view of Eclipse at the bottom of the application (also in the console window). This process had a duration about approximately 3 seconds until all agents have been displayed as online.

Figure 5.8 shows a screenshot of the visualization after the start of a recipe has been triggered. The message exchange is made visible through the decorators of the agents (red and blue envelopes). The Log View also lists the sent and received messages of each agent. The actual display illustrates the communication between the three functional agents (order, work and Evac_101to102 agent) after a recipe has been started and the communication between recipe agent (Evac_101to102) and physical agents (T101,T102 and V101) which changed their agent state to *working*.

Through switching to the agent state view (by using the button with the image of Figure 4.17(c)) the agents' states are represented as colours of the agents (all possible states are represented in the state diagram in Figure 4.2). Currently the three physical agents - T101, T102 and V101 - are in working state (see Figure 5.9). The information to switch to working state is sent by the Recipe agent (Evac_101to102) to all physical agents which concerns the state change. To switch back to the offline view the disconnect button (illustrated in Figure 4.14(b)) is used. After disconnecting the visualized diagram switched back to the state represented in Figure 5.5.

The last test of the visualization concerns the generation of the topology configuration file. The creation of the configuration file is triggered through the configuration button represented in Figure 4.17(a) and is also positioned at the toolbar at the top of the IDE. By activating the button and choosing the destination file through a standard *Save as* dialog the creation of the file is completed. An extract of the generated file can be seen in Listing 5.1 and matches with the existing configuration of the test plant. Especially the automatic creation of the topology has a high potential to save a lot of time and to prevent from configuration failures because existing failures are easier recognized in the graphical representation of the topology than in the textual format.

```xml
<Valve functionalState="" medium="" name="V104" routing=""
    serviceState="" transferState="" unit=""/>
<Valve functionalState="" medium="" name="V106" routing=""
    serviceState="" transferState="" unit=""/>
<Valve functionalState="" medium="" name="V101" routing=""
    serviceState="" transferState="" unit=""/>
</components>
<pipes>
<pipe diameter="0.0" length="0" medium="" name="L101"
    routing="0" transferState="0" unit="">
<link direction="OUT">T101</link>
<link direction="IN">P101</link>
</pipe>
<pipe diameter="0.0" length="0" medium="" name="L107"
    routing="0" transferState="0" unit="">
<link>T102</link>
```

```
<link>V102</link>
</pipe>
<pipe diameter="0.0" length="0" medium="" name="L108"
    routing="0" transferState="0" unit="">
<link>V102</link>
<link direction="IN">T101</link>
<link>V107</link>
</pipe>
```

Listing 5.1: The Listing contains an extract of a generated topology configuration file by the visualization. The output matches with the existing configuration of the test plant.

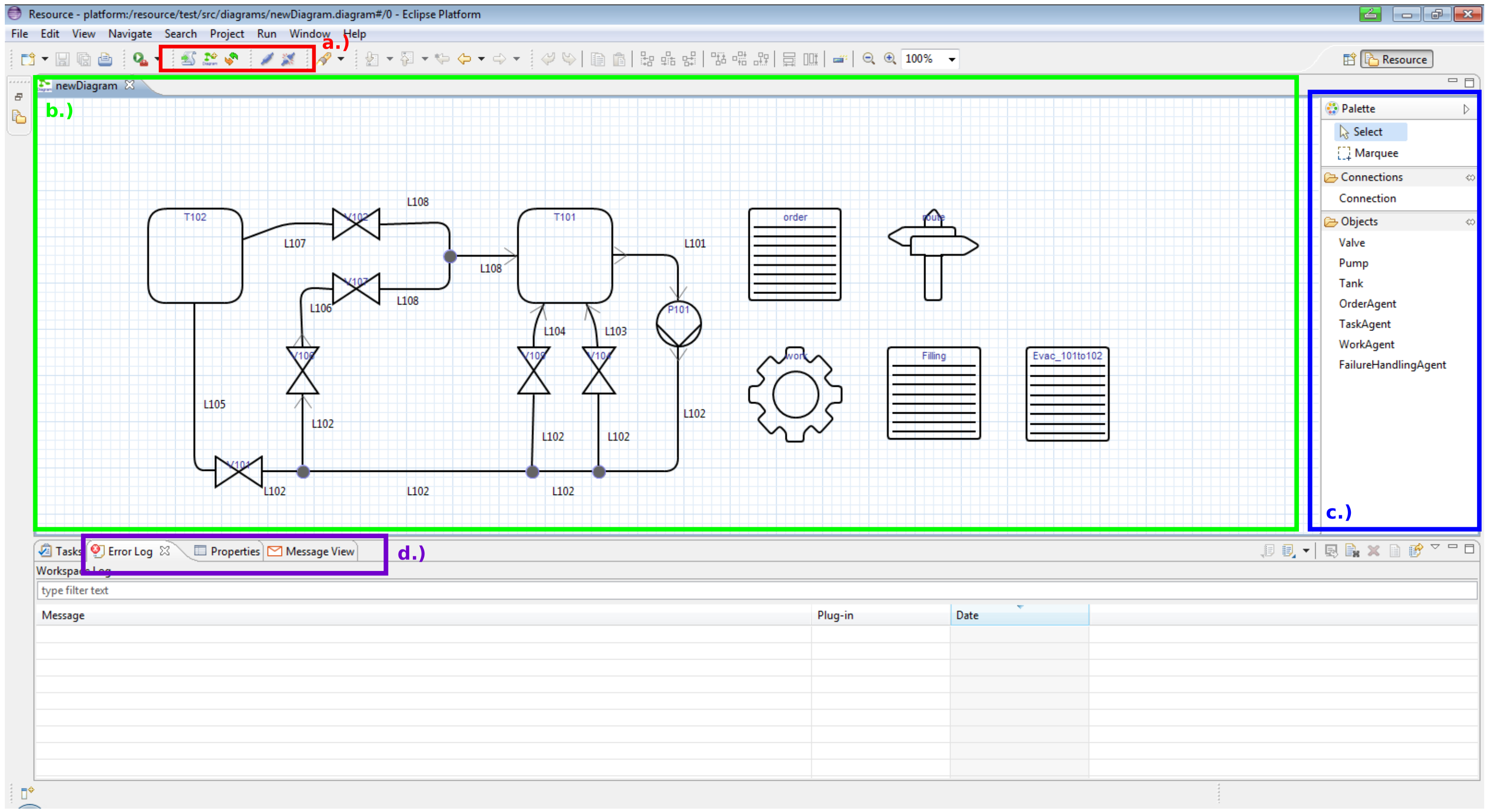
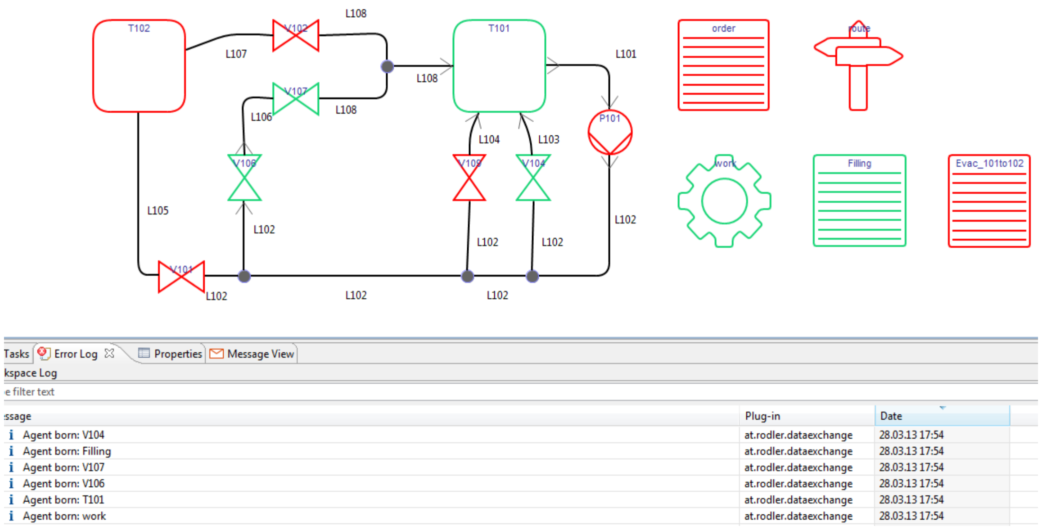


Figure 5.5: A screenshot of a created diagram from the test plant. The available buttons are integrated seamlessly into the existing toolbar of the Eclipse IDE (a.). (b.) frames the editor area in which the agent diagram is created. On the right site (c.) the palette with the available Agents and the Connection is visible (as part of the editor). At the bottom (d.) available views for data manipulation and logging are available. The usual behaviour of Eclipse would also allow a different adjustment of the available views respectively they could also be closed and opened depending on the users requirements.



ssage	Plug-in	Date
Agent born: V104	at.rodler.dataexchange	28.03.13 17:54
Agent born: Filling	at.rodler.dataexchange	28.03.13 17:54
Agent born: V107	at.rodler.dataexchange	28.03.13 17:54
Agent born: V106	at.rodler.dataexchange	28.03.13 17:54
Agent born: T101	at.rodler.dataexchange	28.03.13 17:54
Agent born: work	at.rodler.dataexchange	28.03.13 17:54

Figure 5.6: This screenshot illustrates the visualization shortly after switching to the online mode. Red coloured agents are in offline state, for green coloured agents an online event has already been received. The log entries at the bottom list the incoming events.

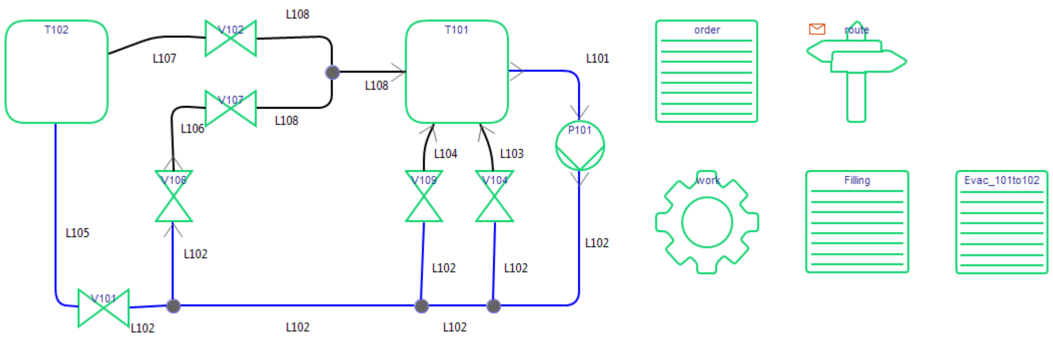
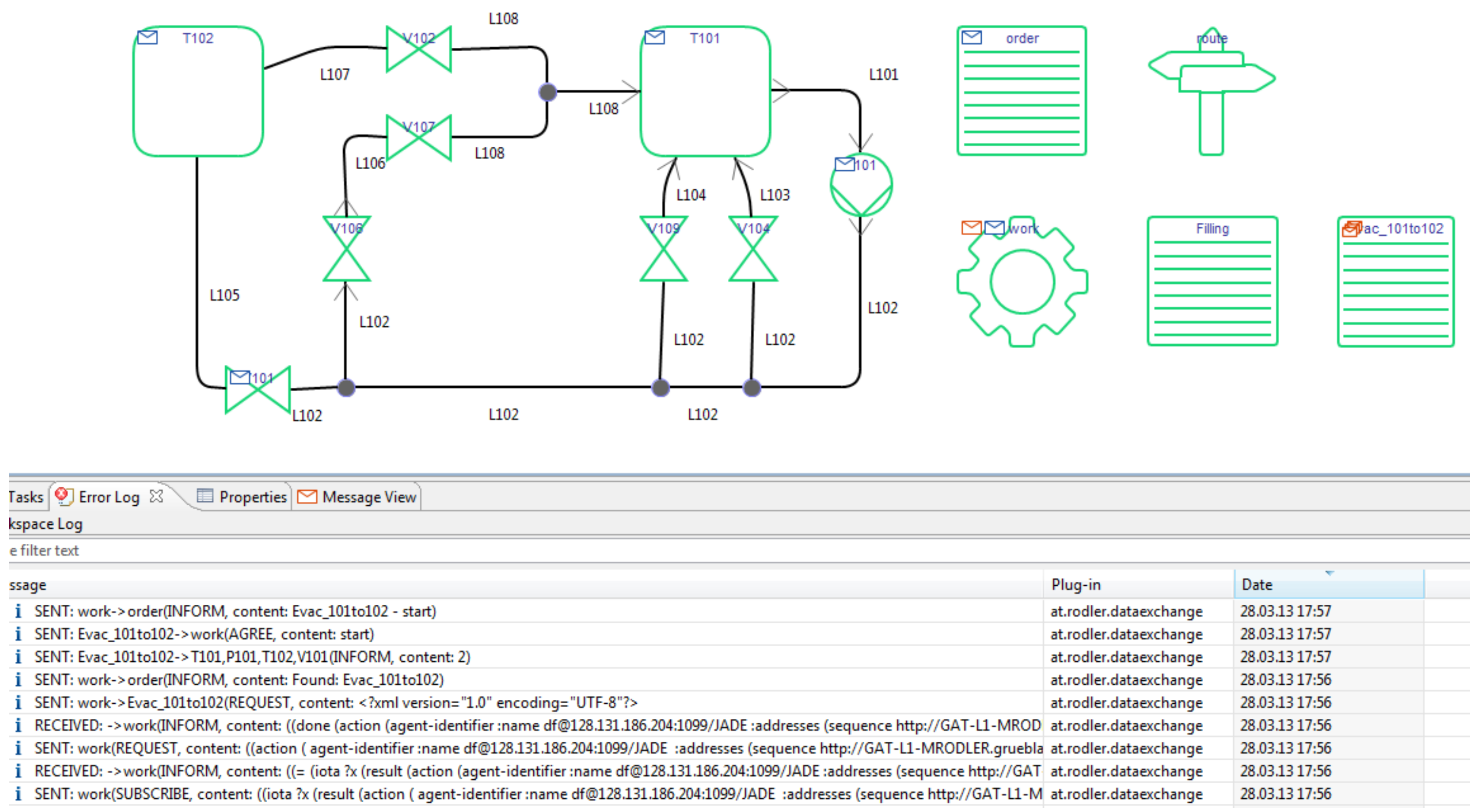
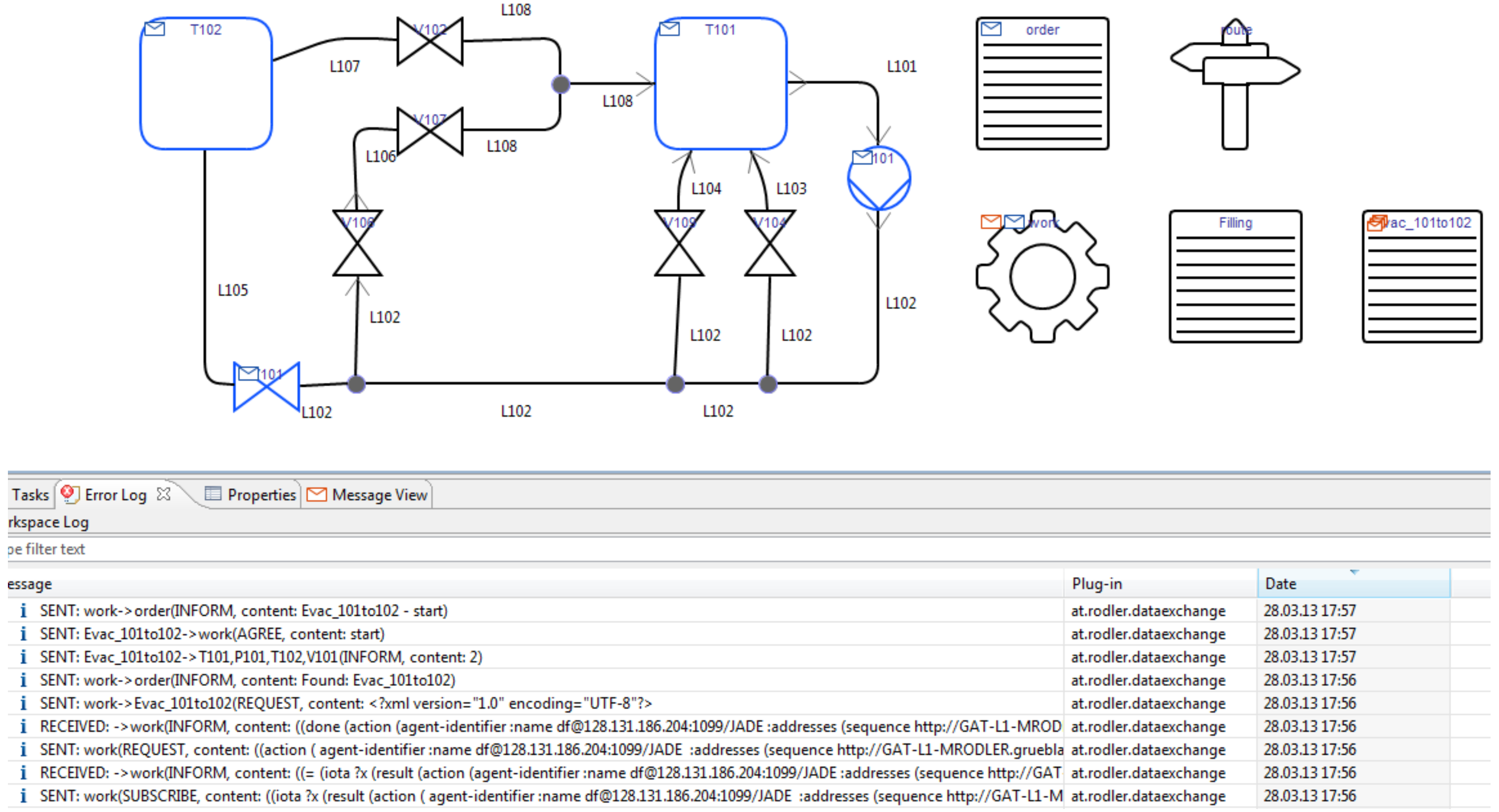


Figure 5.7: This screenshot illustrates the visualization of a proposed route by the route agent (upper right). The sent message of the route agent contains the proposed route. All pipes that are involved in the proposal are blue coloured for a defined period of time (L101,L102,L105),



```
ssage                                                                                                              Plug-in                  Date
SENT: work->order(INFORM, content: Evac_101to102 - start)                                                          at.rodler.dataexchange   28.03.13 17:57
SENT: Evac_101to102->work(AGREE, content: start)                                                                   at.rodler.dataexchange   28.03.13 17:57
SENT: Evac_101to102->T101,P101,T102,V101(INFORM, content: 2)                                                       at.rodler.dataexchange   28.03.13 17:57
SENT: work->order(INFORM, content: Found: Evac_101to102)                                                           at.rodler.dataexchange   28.03.13 17:56
SENT: work->Evac_101to102(REQUEST, content: <?xml version="1.0" encoding="UTF-8"?>                                 at.rodler.dataexchange   28.03.13 17:56
RECEIVED: ->work(INFORM, content: ((done (action (agent-identifier :name df@128.131.186.204:1099/JADE :addresses (sequence http://GAT-L1-MROD   at.rodler.dataexchange   28.03.13 17:56
SENT: work(REQUEST, content: ((action ( agent-identifier :name df@128.131.186.204:1099/JADE  :addresses (sequence http://GAT-L1-MRODLER.gruebla   at.rodler.dataexchange   28.03.13 17:56
RECEIVED: ->work(INFORM, content: ((= (iota ?x (result (action (agent-identifier :name df@128.131.186.204:1099/JADE :addresses (sequence http://GAT   at.rodler.dataexchange   28.03.13 17:56
SENT: work(SUBSCRIBE, content: ((iota ?x (result (action ( agent-identifier :name df@128.131.186.204:1099/JADE  :addresses (sequence http://GAT-L1-M   at.rodler.dataexchange   28.03.13 17:56
```

Figure 5.8: The communication between agents can be clearly identified after starting a recipe through a functional agent. The 3 physical agents T101, T102 and V101 receive information from the Recipe Agent Evac_101to102 (blue envelope). The Evac_101to102 agent notifies the work agent with an *agree* message to start the recipe. The work agent forwards the notification to the order agent. Multiple red envelopes at the recipe agent represent several outgoing messages.



```
essage                                                                                                              Plug-in                  Date
i SENT: work->order(INFORM, content: Evac_101to102 - start)                                                        at.rodler.dataexchange   28.03.13 17:57
i SENT: Evac_101to102->work(AGREE, content: start)                                                                 at.rodler.dataexchange   28.03.13 17:57
i SENT: Evac_101to102->T101,P101,T102,V101(INFORM, content: 2)                                                     at.rodler.dataexchange   28.03.13 17:57
i SENT: work->order(INFORM, content: Found: Evac_101to102)                                                         at.rodler.dataexchange   28.03.13 17:56
i SENT: work->Evac_101to102(REQUEST, content: <?xml version="1.0" encoding="UTF-8"?>                              at.rodler.dataexchange   28.03.13 17:56
i RECEIVED: ->work(INFORM, content: ((done (action (agent-identifier :name df@128.131.186.204:1099/JADE :addresses (sequence http://GAT-L1-MROD   at.rodler.dataexchange   28.03.13 17:56
i SENT: work(REQUEST, content: ((action ( agent-identifier :name df@128.131.186.204:1099/JADE  :addresses (sequence http://GAT-L1-MRODLER.gruebla   at.rodler.dataexchange   28.03.13 17:56
i RECEIVED: ->work(INFORM, content: ((= (iota ?x (result (action (agent-identifier :name df@128.131.186.204:1099/JADE :addresses (sequence http://GAT   at.rodler.dataexchange   28.03.13 17:56
i SENT: work(SUBSCRIBE, content: ((iota ?x (result (action ( agent-identifier :name df@128.131.186.204:1099/JADE  :addresses (sequence http://GAT-L1-M   at.rodler.dataexchange   28.03.13 17:56
```

Figure 5.9: This screenshot illustrates the visualization after switching to the agent state view. The current agent states are represented through colours used in the state diagram in Figure 4.2. The blue colour identifies the agents T101, T102 and V101 in working state. Referring to the illustrated message decorators, the MAS situation is the same as in Figure 5.8.

5.2 Code Analysis

This Section will analyze the source code of the developed application. Furthermore different enhancement scenarios and its influence on the generated and manually written source code will be discussed. For the comparison of lines of code only the source code lines have been counted without any blank lines or comments.

Provided that the used frameworks are mature and well tested the usage of generated code can minimize the amount of failures within the source code. The following lines will discuss the proportion between generated and manually written code due to the fact that already existing Eclipse frameworks have been used and that code generation can be seen as key functionality for a great part of the integrated frameworks.

Figure 5.10 illustrates the relation between the automatically generated code by used frameworks in comparison to redeveloped code (refer to table 5.4 for a result in numbers). The code generation primarily applies to the generated code from parts of the EMF framework which is responsible for the data model and its property sheets for data manipulation. The amount of newly developed code is less than 20 percent of the generated code. Furthermore only a fractional amount of the generated code had to be customized. A customization of generated code is always critical due to the fact that a regeneration of source code evokes an update of the code with those parts who have already been customized. Thanks to the features of the EMF framework there is no need to rewrite the generated code every time because the framework is able to recognize modified code and leaves customized code as is (as described in Sec. 4.5). This allows a repetitive change of the data diagram without the need to modify the generated code every time. This is seen as great benefit by considering the proportions of the source code in Figure 5.10. The total number of lines of code (LoC) is listed in Table 5.4.

Table 5.4: This Table compares the total lines of code contained by the project. The first three columns represent pure Java Source Code (illustrated as Pie Chart in Figure 5.10). The last column contains the LoC contained in the XML file for the graphical definition of the agents.

Generated	Customized	Redeveloped	XML Graphic Definition
22597	165	5063	154

The following comparison of the graphical definition of one of the used agents with the equal outcome written in Java should illustrate possible advantages

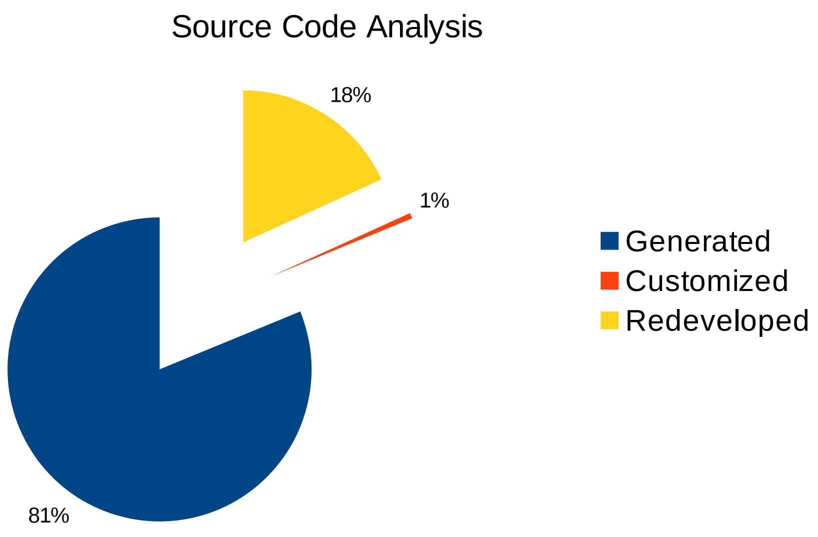


Figure 5.10: The diagram illustrates the fragmentation of source code into automatic generated, customized and redeveloped source code. Through the usage of frameworks a great part of the source code could be generated.

through the usage of the proposed XML definition. Listing 5.2 contains the definition of the Pump agent used in this use case. The whole definition covers 8 lines of code. The readability of the code is still under good condition. In comparison Listing 5.3 contains an equal definition written directly in Java Code.

```
<agent name="Pump">
        <shape name="ellipse" filled="false" width="100"
            height="100" lineWidth="2"/>
        <shape name="polyline" lineWidth="2">
                <point x="50" y="0" />
                <point x="100" y="50" />
                <point x="50" y="100" />
        </shape>
</agent>
```

Listing 5.2: The Listing contains the definition of the pump agent used in this use case. The total number of lines is 8.

```
1  Ellipse ellipse = Graphiti.getGaCreateService().
       createEllipse(container);
2  ellipse.setFilled(false);
3  ellipse.setLineWidth(2);
4  ellipse.setHeight(100);
5  ellipse.setWidth(100);
6  Polyline polyline = Graphiti.getGaCreateService().
       createPolyline(container);
7  polyline.setLineWidth(2);
8  int pointArray[]=new int[6];
9  pointArray[0]=50;
10 pointArray[1]=0;
11 pointArray[2]=100;
12 pointArray[3]=50;
13 pointArray[4]=50;
14 pointArray[5]=100;
15 EList<Point> points=polyline.getPoints();
16 List<Point> newPoints=gaServie.createPointList(pointArray);
17 points.addAll(newPoints);
```

Listing 5.3: The Listing contains the equivalent of Listing 5.2 implemented in Java Code. Some of the instructions could be combined (e.g. filling the array from line 9 to 14) but the total number of lines would not fall below 9. Furthermore the readability of the code would suffer under a combining of the code.

In practice the needed creation of pictogram objects (see Sec. 4.8 for details) would also need to be added as well as the mapping to the corresponding domain model element. But only considering the graphical definition in XML and Java allows rough conclusions about the amount of code which can be saved. The proportion of the listed code fragments is 1:2. Even if the code in Listing 5.3 is combined to achieve a lower number of lines it would still be higher in comparison to the XML definition. Furthermore this action would decrease the readability and thereby the maintainability of the code as well as changes of the source code would require a recompilation of the modified source files.

5.3 Summary

This Chapter presented the outcome of tests from the visualization combined with the test plant in the Odo Struger Laboratory. The tests demonstrated a successful usage of the visualization with the existing multi-agent system JADE. A possible scenario of a preparation from the visualization by the tutor and the installation and usage by the student has been run through during the tests. The usage of the visualization could also improve the comprehensibility of the existing agents. A possible bug in the Recipe agent could also be located with the aid of the visualization through the tests with the available plant. The link to an OPC UA Server and the representation of OPC UA tags has also been successfully tested. Furthermore a successful usage under a Linux based operating system could also be performed which proves the interoperability to other than Microsoft bases operating systems. At last the amount of generated source code has been compared with the modified and developed code. This comparison demonstrated that most of the code could be generated through used frameworks which also increases the quality of code. Only marginal parts of the generated source code had to be customized. An equation of the XML definition of the graphical representation with an implementation in Java for an equivalent representation illustrated the benefits of the implemented approach related to readability and maintainability.

6 Conclusion

A fast moving market demands for a flexible and dynamic industry. Automation aided production lines and the transportation and supervision of goods is a common practice nowadays. These requirements claim for new methods to stay competitive on the market. To reach this aim the industry has to uncouple from the traditional way of industrial automation - especially by fulfilling batch processes - by using centralized computing units and needs to forward to a more agile approach of multi-agent systems. But due to missing trust in this technology MAS could not affirm its strength on the market yet. Particularly the missing traceability and lack of understanding are main reasons for its niche existence. Chapter 2 has illustrated current possibilities of different agents and their field of application. Furthermore it examined current existing visualization tools for multi-agent systems and covered its pros and cons. In this context new emerging possibilities for visualizations based on web technologies have also been investigated. But the current state of the art offers hardly fundamentals for a flexible and easy to handle visualization of different multi-agent systems especially not for integration into an existing development process of an MAS by offering a high flexibility.

Due to this fact Chapter 3 presented an approach for a new visualization which fulfills the demands listed in Sec. 1.3. The approach consists of a 3-tier based architecture - visualization, data exchange and data collection - that should fulfill the demands on a modern and flexible visualization framework. Moreover a layered architecture should offer the possibility to use the same visualization for different multi-agent systems by exchanging the lowest layer of the framework. A generic data model and a flexible graphical definition should provide a high adaptiveness by modifying the preset data to fulfill the required demands by an illustration of an MAS.

Chapter 4 presented the proof of concept by integrating the introduced approach into the Eclipse IDE with a fundament of projects powered by the Eclipse Foundation. The implementation could be built on well developed and approved projects named Graphiti and EMF. Both offer essential features for the integration of the approach and raise the quality of the outcome because of their proper code quality. The usage of existing Open Source projects has to be well considered because discontinued projects could lead to increasing costs by maintaining the developed framework. However the chosen frameworks can lean on a living community which push the development process of the projects. This has lead to several bug fixes and improvements of the provided code during the composition of this master thesis.

A test plant at the Odo Struger Laboratory has been used as test case and the results of the tests are discussed in Chapter 5. The presented outcome describes a successful usage of the implementation from the presented approach. A "tutor - student scenario" (see Chapter 5.1 for details) has been used to run through a real world setting to test the usability beginning from the preparation of the visualization to fit the requirements of the plant onward to the distribution of the visualization by using the internet till testing the online behaviour of the visualization in conjunction with the test plant.

The successfulness of the tests have approve the design approach in Chapter 3, the technology decisions in Section 4.3 followed by the implementation afterwards. Future work could enhance the existing code base to provide more possibilities to extend the existing functionality without touching the base source code. Moreover other Eclipse frameworks are worth for an evaluation for a possible integration in the current implementation (e.g. Spray [spr] as another sophisticated way to provide a graphical representation). Also a more generic approach for the visualization (off from agents) would open other fields of application (not only agents can be sniffed). The definition of illustrations from agents is no restriction due to the generic approach of defining the graphical definition. Yet another possibility would be an even more generic approach which would allow the definition of the agent structure in configuration files. Furthermore the implementation of a web based user interface could be evaluated. As mentioned the outcome of this master thesis can provide a base for extensive future work to enhance the acceptance and usage of multi-agent systems among students and developers in the automation industry.

Bibliography

[acl] Class aclmessage. http://jade.cselt.it/doc/api/jade/lang/acl/ACLMessage.html. Accessed Mar. 16, 2013.

[age] Agentfly. http://agents.felk.cvut.cz/projects#agentfly. Accessed Jan. 5, 2013.

[BCTR] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. Jade programmer's guide. http://jade.tilab.com/doc/programmersguide.pdf. Accessed Mar. 15, 2013.

[BE96] T. Ball and S.G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[bro] Browser statistics. http://www.w3schools.com/browsers/browsers_stats.asp. Accessed Oct. 24, 2012.

[CBJC10] Gonçalo Cândido, José Barata, François Jammes, and Armando W. Colombo. Applications of dynamic deployment of services in industrial automation. In Luis M. Camarinha-Matos, Pedro Pereira, and Luis Ribeiro, editors, *DoCEIS*, volume 314 of *IFIP Advances in Information and Communication Technology*, pages 151–158. Springer, 2010.

[Die03] S. Diehl. Softwarevisualisierung. In *Informatik Spektrum*, pages 257–260, 2003.

[ecla] About the eclipse foundation. http://www.eclipse.org/org/. Accessed Nov. 28, 2012.

[eclb] Eclipse projects list. http://projects.eclipse.org/list-of-projects. Accessed Feb. 12, 2013.

[Eclc] The java developer's guide to eclipse. http://www.jdg2e.com/ch08.architecture/doc/index.html. Accessed Nov. 23, 2012.

[emf] Eclipse modeling framework project (emf). `http://www.eclipse.org/modeling/emf/?project=emf`. Accessed Feb. 13, 2013.

[FBG07] Giovanni Caire Fabio Bellifemine and Dominic Greenwood. *developing multi-agent systems with JADE*. John Wiley & Sons, Ltd, 2007. Public Available Specification.

[FIP] FIPA. Foundation for intelligent physical agents. `http://www.fipa.org/`. Accessed Nov. 10, 2012.

[fla] Thoughts on flash. `http://www.apple.com/hotnews/thoughts-on-flash/`. Accessed Oct. 04, 2012.

[FR09] C. Foreman and R.K. Ragade. Coordinated optimization at a hydro-generating plant by software agents. *Control Systems Technology, IEEE Transactions on*, 17(1):89–97, 2009.

[goo] Google docs. `http://docs.google.com`. Accessed Oct. 23, 2012.

[graa] Graphiti. `http://projects.eclipse.org/projects/modeling.gmp.graphiti`. Accessed Mar. 01, 2013.

[grab] Overview of graphiti. `http://www.eclipse.org/graphiti/documentation/overview.php`. Accessed Nov. 23, 2012.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[gwt] Google web toolkit blog. `http://googlewebtoolkit.blogspot.co.at/`. Accessed Nov. 03, 2012.

[HD04] Marc-Philippe Huget and Yves Demazeau. Evaluating multiagent systems: A record/replay approach. In *IAT*, pages 536–539. IEEE Computer Society, 2004.

[Hec] Steven A. Hechtman. Web-based hmi: An emerging trend? `http://www.automation.com/resources-tools/articles-white-papers/hmi-and-scada-software-technologies/web-based-hmi-an-emerging-trend`. Accessed Dec. 15, 2012.

[Hel] Jonas Helming. What every eclipse developer should know about emf. `http://eclipsesource.com/blogs/2011/03/22/what-every-eclipse-developer-should-know-about-emf-part-1/`. Accessed Nov. 24, 2012.

[HTM] Html5 games. `http://html5games.com`. Accessed Nov. 01, 2012.

[HTW04] A. Helsinger, M. Thome, and T. Wright. Cougaar: a scalable, distributed multi-agent architecture. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, pages 1910–1917 vol.2, 2004.

[HYJY10] Zhao Hongwei, Qi Yiming, Zhang Jiye, and Zhao Yuanheng. Based on multi-agent model for grinding process control research. In *Frontier of Computer Science and Technology (FCST), 2010 Fifth International Conference on*, pages 576–581, 2010.

[ide] Java ee productivity report 2011. `http://zeroturnaround.com/java-ee-productivity-report-2011/`. Accessed Nov. 23, 2012.

[int] Intellij idea editions comparison. `http://www.jetbrains.com/idea/features/editions_comparison_matrix.html`. Accessed Nov. 28, 2012.

[Jada] Jade java agent development framework. `http://jade.tilab.com/`. Accessed Oct. 04, 2012.

[Jadb] Who is using jade. `http://jade.tilab.com/application-who.htm`. Accessed Nov. 10, 2012.

[jax] Das modeling-framework eclipse graphiti. `http://it-republik.de/jaxenter/artikel/Das-Modeling-Framework-Eclipse-Graphiti-3551.html`. Accessed Apr. 16, 2012.

[JB03a] N.R. Jennings and S. Bussmann. Agent-based control systems: Why are they suited to engineering complex systems? *Control Systems, IEEE*, 23(3):61–73, 2003.

[JB03b] N.R. Jennings and S. Bussmann. Agent-based control systems: Why are they suited to engineering complex systems? *Control Systems, IEEE*, 23(3):61–73, 2003.

[jde] Oracle technology network developer license terms for jdeveloper. `http://www.oracle.com/technetwork/licenses/jdev-license-152012.html`. Accessed Nov. 28, 2012.

[Jin] Jing Jin. Vizscript editor plug-in for eclipse. `http://www-scf.usc.edu/~jingjin/VizScriptEditor/`. Accessed Nov. 17, 2012.

[JMSS07] Jing Jin, Rajiv T. Maheswaran, Romeo Sanchez, and Pedro Szekely. Vizscript: visualizing complex interactions in multi-agent systems. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 369–372, New York, NY, USA, 2007. ACM.

[JSMS08] Jing Jin, Romeo Sanchez, Rajiv T. Maheswaran, and Pedro Szekely. Vizscript: on the creation of efficient visualizations for understanding complex multi-agent systems. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '08, pages 40–49, New York, NY, USA, 2008. ACM.

[KZ12] Karsten Krieg and Stefan Zilch. Javascript-frameworks. *Javamagazin*, pages 96 – 102, Nov 2012.

[La09] Paulo Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Eng. Appl. Artif. Intell.*, 22(7):979–991, October 2009.

[Mad07] Janusz Madejski. Survey of the agent-based approach to intelligent manufactoring. *Journal of Achievements in Materials and Manufacturing Engineering*, 21(1):67–70, 2007.

[MEP10] E. Muñoz, A. Espuña, and L. Puigjaner. Towards an ontological infrastructure for chemical batch process management. *Computers & Chemical Engineering*, 34(5):668 – 682, 2010. <ce:title>Selected Paper of Symposium {ESCAPE} 19, June 14-17, 2009, Krakow, Poland</ce:title>.

[Mer09] Munir Merdan. *Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain.* PhD thesis, Technical University of Vienna, Vienna, 2009.

[MLA11] M. Merdan, W. Lepuschitz, and E. Axinia. Advanced process automation using automation agents. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 34 –39, Dec. 2011.

[MLSV10] M. Merdan, W. Lepuschitz, B. Săhović, and M. Vallée. Failure detection and recovery in the batch process automation domain using automation agents. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, pages 113–117, 2010.

[MM05] V. Marik and D. McFarlane. Industrial adoption of agent-based technologies. *Intelligent Systems, IEEE*, 20(1):27–35, 2005.

[MMH13] B. Grössing M. Merdan, W. Lepuschitz and M. Helbok. Process rescheduling and path planning using automation agents. *Recent Advances in Robotics and Automation, Series: Studies in Computational Intelligence*, 2013.

[MMW+08] M. Merdan, T. Moser, D. Wahyudin, S. Biffl, and P. Vrba. Simulation of workflow scheduling strategies using the mast test management system. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 1172 –1177, dec. 2008.

[MOR] Vladimir Marik Marek Obitko, Pavel Vrba and Miloslav Radakovic. Semantics in industrial distributed systems. `http://tc.ifac-control.org/5/3/activities/tc5-3-sessions-at-wc2008/c24-presentations/obitko-ifac2008-semantics-final.pdf`. Accessed Nov. 10, 2012.

[MP11] M. Metzger and G. Polakow. A survey on applications of agent technology in industrial process control. *Industrial Informatics, IEEE Transactions on*, 7(4):570–581, 2011.

[MR] Marek Obitko andPavel Vrba Miloslav Radakovic. Architecture for explicit specification of agent behavior. `http://tc.ifac-control.org/5/3/events/incom2009-folder/presentations-at-the-ein-track/session-th-a6/2-incom09d.pdf`. Accessed Nov. 10, 2012.

[MSD+07] F. Maturana, R. Staron, F. Discenzo, D. Carnahan, and K. Hall. Agent virtual machine for shipboard automation systems. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8, 2007.

[net] Kommentar: Aufatmen bei netbeans? `http://it-republik.de/jaxenter/artikel/Kommentar-Aufatmen-bei-NetBeans-2858.html`. Accessed Dec. 3, 2012.

[NNLC99] Divine T. Ndumu, Hyacinth S. Nwana, Lyndon C. Lee, and Jaron C. Collis. Visualising and debugging distributed multi-agent systems. In *Proceedings of the third annual conference on*

Autonomous Agents, AGENTS '99, pages 326–333, New York, NY, USA, 1999. ACM.

[NPB+11] Cu D. Nguyen, Anna Perini, Carole Bernon, Juan Pavón, and John Thangarajah. Testing in multi-agent systems. In Marie Pierre Gleizes and Jorge J. Gómez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 180–190. Springer, 2011.

[off] Microsoft office 365. `http://www.microsoft.com/en-us/office365/`. Accessed Oct. 23, 2012.

[PM08a] Grzegorz Polaków and Mieczyslaw Metzger. Web-based monitoring and visualization of self-organizing process control agents. In Karin Hummel and James Sterbenz, editors, *Self-Organizing Systems*, volume 5343 of *Lecture Notes in Computer Science*, pages 325–331. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-92157-8_33.

[PM08b] M. Pěchouček and V. Mařík. Industrial deployment of multi-agent technologies: review and selected case studies. In *Autonomous Agents and Multi-Agent Systems*, volume 17, pages 397–431, Dec. 2008. `http://dx.doi.org/10.1007/s10458-008-9050-0`.

[PvPU06] Michal Pěchouček, David Šišlák, Dušan Pavlíček, and Miroslav Uller. Autonomous agents for air-traffic deconfliction. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 1498–1505, New York, NY, USA, 2006. ACM.

[QC] Elena Quarantotto and Giovanni Caire. Jade osgi guide. `http://jade.tilab.com/doc/tutorials/JadeOsgiGuide.pdf`. Accessed Apr. 1, 2013.

[SeB] SeB. use eclipse eef as a property sheet for the graphiti editor. `http://eclipsercpdev.blogspot.co.at/2011/10/use-eclipse-eef-as-property-sheet-for.html`. Accessed Apr. 1, 2013.

[SL07] Jämsä-Jounela Sirkka-Liisa. Future trends in process automation. In *Annual Reviews in Control*, volume 31, pages 211 – 220, 2007.

[SN99] W. Shen and D. Norrie. 'Agent-Based systems for intelligent manufacturing: A State-of-the-Art survey. *Knowledge and Information Systems: An International Journal*, 1(2):129–156, 1999.

[sni] Jade sniffer. `http://jade.tilab.com/doc/tools/sniffer/`. Accessed Mar. 15, 2013.

[spr] Spray - a quick way of creating graphiti. `http://code.google.com/a/eclipselabs.org/p/spray/`. Accessed May. 27, 2013.

[Sta12] Stefan Starke. Gwt meets html5. *Javamagazin*, pages 60 – 66, Jan 2012.

[svg] Svg test suite results. `http://www.codedread.com/svg-support.php`. Accessed Oct. 27, 2012.

[Syc98a] K. P. Sycara. Multiagent systems. In *AI MAGAZINE*, volume 19, pages 79–92, 1998.

[Syc98b] Katia P. Sycara. Multiagent systems. *AI Magazine*, 19:79–92, 1998.

[VKJ+11] Pavel Vrba, Petr Kadera, Václav Jirkovský, Marek Obitko, and Vladimír Mařík. New trends of visualization in smart production control systems. In *Proceedings of the 5th international conference on Industrial applications of holonic and multi-agent systems for manufacturing*, HoloMAS'11, pages 72–83, Berlin, Heidelberg, 2011. Springer-Verlag.

[VM10] P. Vrba and V. Marik. Capabilities of dynamic reconfiguration of multiagent-based industrial control systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40(2):213–223, 2010.

[VMK12] P. Vrba, V. Marík, and P. Kadera. Mast: From a toy to real-life manufacturing control. In *Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pages 428 –433, aug. 2012.

[VMLK11] M. Vallee, M. Merdan, W. Lepuschitz, and G. Koppensteiner. Decentralized reconfiguration of a flexible transportation system. *Industrial Informatics, IEEE Transactions on*, 7(3):505–516, 2011.

[Vog] Lars Vogel. Gwt tutorial. `http://www.vogella.com/articles/GWT/article.html`. Accessed Nov. 03, 2012.

[Wei99] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence.* MIT Press, Cambridge, MA, USA, 1999.

[Yak] Masataka Yakura. Html5 & css3 readiness. `http://html5readiness.com/`. Accessed Oct. 24, 2012.

[zuk] Die zukunft der automatisierung. `http://www.openautomation.de/883-0-die-zukunft-der-automatisierung.html`. Accessed Oct. 06, 2012.