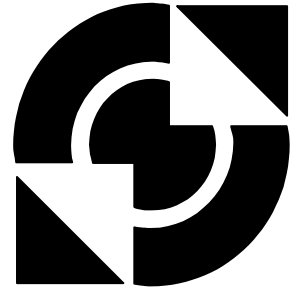


University of Twente



Faculty of Electrical Engineering,
Mathematics & Computer Science

Integrated Design and Implementation Tool for Multi-Agent Controllers [IDITmac]

G. Bajracharya
MSc Thesis

Supervisors: prof. dr. ir. J. van Amerongen
dr. ir. T.J.A. de Vries
dr. ir. J.F. Broenink
ir. M.H.A. Verwoerd

March 2003

Report 001CE2003
Control Laboratory
Faculty of Electrical Engineering,
Mathematics & Computer Science
University of Twente
P. O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Multi-Agent Controller Systems (MACS) form a new concept in the field of control engineering. In the Multi-Agent Controller System framework, a complex control problem is divided into simple control problems and the solution to each of these problems is implemented as a controller agent. Controller Agents within MACS are coordinated with each other to solve the complex problem.

A specification language, MacsML (Multi-Agent Controller Specification Markup Language), has been developed for specifying MACS, which is based on XML (Extensive Markup Language) and on MACSL (Multi-Agent Controller Specification Language), a specification language for MACS that has been developed previously. A tool, IDITmac (Integrated Design and Implementation Tool for Multi-Agent Controllers), has been developed in this thesis, which supports checking of the specification and generation of C++ code of the specified MACS. The generated code can run in real system. A dll (dynamic link library) of the MACS can also be generated by this tool, which can then be used in 20-Sim for simulation purposes.

A demonstration of MACS has been implemented on a mass-spring-mass system (DemoLin from Imotec BV), and the generated code has been tested both in simulation (20-Sim) and in real system (20-Works).

Contents

Preface	v
1 Introduction.....	1
1.1 Modern Control Solutions.....	1
1.2 Integrated Design and Implementation Tool.....	2
1.3 Multi Agent Controller.....	2
1.4 Goals of the project.....	3
1.5 IDITmac	4
1.6 Outline of this thesis.....	4
2 Background of the Project	7
2.1 Introduction.....	7
2.2 Overview of Methodologies for Controllers	7
2.2.1 Control Architecture.....	8
2.2.2 Agents Based Software Development	9
2.2.3 Code generation.....	11
2.2.4 Controller specification	11
2.3 Integrated Approach for Controllers	12
2.3.1 Competent Specification of Multi Agent Controller	13
2.3.2 Support for Design of Multi Agent Controller	13
2.3.3 Integrated Implementation of Multi Agent Controller	13
2.4 Conclusion	15
3 Multi Agent Controller System Architecture.....	17
3.1 Introduction.....	17
3.2 Agent.....	17
3.3 Multi Agent Controller System.....	18
3.3.1 Overview of Multi-Agent Controller Implementation Framework (MACIF)	18
3.4 Architecture of Multi Agent Controller System.....	22
3.4.1 Agent.....	23
3.4.2 Sensor Agent	24
3.4.3 Actuator Agent	24
3.4.4 Controller Agent.....	24
3.4.5 Elementary Agent.....	25
3.4.6 Composite Agent.....	25
3.4.7 Coordination Object	26
3.4.8 Main Agent.....	26
3.5 Conclusion	27
4 Specification of Multi Agent Controller System	29
4.1 Introduction.....	29
4.2 Multi-Agent Controller Specification Language.....	29
4.3 Multi-Agent Controller Specification Markup Language (MacSML). 30	
4.3.1 XML Schema for MacSML	32
4.3.2 Basic Structure of MacSML	33
4.3.3 Input Ports and Output Ports	34
4.3.4 Parameters Definitions and States.....	35
4.3.5 Method Specification	35
4.3.6 Controller Agents within Composite or Main Agent	36
4.3.7 Coordination within Composite or Main Agent.....	36
4.3.8 Connections within Composite or Main Agents	37

4.3.9	Including Header Files.....	37
4.3.10	Instances of C++ Classes.....	37
4.4	Conclusion.....	38
5	Design and Implementation of Multi-Agent Controller Systems.....	39
5.1	Introduction.....	39
5.2	Operating Principle of Multi Agent Controller Systems.....	39
5.2.1	Multi-Agent Controller Systems and the Environment.....	39
5.2.2	Main Agent.....	40
5.2.3	Sensor Agent.....	42
5.2.4	Actuator Agent.....	42
5.2.5	Coordination Object.....	43
5.2.6	Composite Agent.....	43
5.2.7	Elementary Agent.....	44
5.2.8	Realization of Multi-Agent Control Systems.....	45
5.3	Keywords.....	46
5.3.1	Ports, Parameters and State Variables types.....	46
5.3.2	Sensor and Actuator types for 20-Sim.....	46
5.3.3	Operating State of Elementary Agents.....	46
5.3.4	Keywords of Coordination Objects.....	46
5.4	Design of Multi-Agent Controller Systems with 20-Sim.....	48
5.5	Implementation of Multi-Agent Controller Systems with 20-Works.....	49
5.6	Conclusion.....	50
6	Case Study.....	51
6.1	Introduction.....	51
6.2	Demonstration Setup (DemoLin).....	51
6.3	Control Problem.....	52
6.4	Control Strategy.....	52
6.4.1	Sensor and Actuator Agents.....	54
6.4.2	Motions of DemoLin.....	54
6.4.3	PID-Controller and Safety.....	56
6.5	Simulation Result.....	56
6.5.1	Model of DemoLin.....	56
6.6	Implementation Result.....	57
6.7	Conclusion.....	60
7	Conclusion and Recommendation.....	61
7.1	Conclusion.....	61
7.2	Recommendation.....	63
APPENDIX A	XML and XML Schema.....	65
A.1	Background of XML (Extensive Markup Language).....	65
A.2	XML (Extensive Markup Language).....	65
A.3	W3C XML Schema.....	66
APPENDIX B	UML.....	69
B.1	Structural Diagrams.....	69
B.1.1	Class Diagram.....	69
B.2	Behavior Diagrams.....	70
B.2.1	Sequence Diagram.....	70
B.2.2	Statechart Diagram.....	71
APPENDIX C	Features of IDITmac.....	73
C.1	Support for MacsML in IDITmac.....	74
C.2	Support for Design and Implementation in IDITmac.....	75
APPENDIX D	DemoLin Setup and Model.....	77
D.1	Configuration of DemoLin.....	77
D.2	Encoders of DemoLin.....	78

D.3	Model of DemoLin.....	78
APPENDIX E	Design of PID-Controller for DemoLin.....	79
E.1	Plant Model.....	79
E.2	Design.....	80
E.3	Online Parameter Tuning.....	81
APPENDIX F	Multi-Agent Controller System for DemoLin.....	83
Bibliography	103

PREFACE

Thanks are due to a lot of people for their encouragement and help in the completion of this project.

Firstly, I would like to express my gratitude to Theo de Vries, for his valuable guidance and continuous support during execution of this project. His suggestions and critical remarks were instrumental in the success of this thesis. I would like to thank Job van Amerongen for providing a warm welcome into the Control Group and for directing me towards finding a project suited to my interests. I would also like to thank Jan Broenink for his suggestions in the UML aspects of this project, and Bas de Kruif for his guidance in this project. I am grateful to Mark Verwoerd for his feedback regarding my report. A special thanks goes to Paul Weustink and CLP for his help for preparing the movie for the presentation.

The case study used in this project is implemented in DemoLin, a setup from Imotec BV. I am grateful to Imotec for providing me the facility to work on their setup.

I would also like to thank my parents for their ever-present and unconditional support. Last but certainly not the least, a special thanks to Albert van Breemen for creating the Multi-Agent Controllers, which provides the framework for this thesis.

CHAPTER 1

INTRODUCTION

1.1 Modern Control Solutions

Needs of control engineers are growing with the advancement of technologies in the field of control engineering. Rapid advances in computational power introduce ability to solve even more complex control problems. Sophisticated systems have been successfully developed and implemented which exploit enormous computational power. Almost all present day controllers are being implemented with software realization to accomplish the increasing needs of the current problems. Hardware excellence in computer applications supplements functionality, flexibility as well as cost effectiveness of the computer based control solutions.

Soft solution approach to hard control problems of industrial environments has triggered evolution of automation in manufacturing processes. Present industrial controllers evolved from conventional loop control to hierarchical layered structures to supplement additional functionalities. Functions such as Sequence Control, Supervisory Control, Data logging and Interfaces with other plants, which used to be accomplished with human intelligence, are being implemented autonomously with the aid of computer supervision. Automation not only reduces production cost but also improves production efficiency by increasing production capacity and decreasing production time.

Ease of operation and improved productivity are accomplished through considerable attention in the design phase. Intensive simulations and rigorous tests ensure successful performance of the system in consideration. Recent developments in the field of simulation tools, particularly for mechatronic systems such as industrial applications, have considerably reduced design cost by rectifying design faults prior to its implementation, which would else wise be identified in the testing or even worse in the operational phase of the system. Numerous commercially available tools [25], [28], [29] are being utilized in the design of mechatronic systems, which facilitate the controller design and verification by simulating the designed controller on the model of the plant.

1.2 Integrated Design and Implementation Tool

In an industrial process, design of controllers involve formulation of reasonably accurate models of the plant to be controlled, designing control laws based on the derived models and simulating the designed control laws using available simulation tools. Whereas implementation is accomplished by converting the designed control laws to the native code of target systems, most commonly embedded microprocessor based architecture or personal computer with analog and digital interfaces. Controllers can be designed in the continuous, discrete or hybrid time domain whereas implementation is accomplished mostly in discrete time domain as most of the present day controllers are being implemented in digital machines. Presence of the vast difference in design and implementation of control applications is inherent due to different concepts in the field of control engineering and computer science. Thus, transformation of controller designs to implementation induces possibilities of errors and unreliable behaviors. In some cases, these errors cannot be identified by rigorous tests of the implementation thus these errors results in failure of the system causing serious and even catastrophic disaster.

Furthermore, a hierarchical layered control structure, which is essential in the industrial processes, is complicated to be simulated using present simulation tools for mechatronic systems. Features such as mode switching and state transition, which are often encountered in a control system, are difficult to capture in simulation tools and cannot be structured well. Being object oriented, these tools still cannot accommodate inheritance and polymorphism in the design, which is a key factor of successful and efficient implementation. The resulting implementation often tends to deviate from the designed controller, downgrading the performance of the controller.

The need for an integrated approach in the design and implementation of a controller for mechatronic systems is mandatory. This project is an effort to extend these well-proven tools for supporting not only the design aspect of mechatronic systems but also its implementation phases thus providing an integrated design and implementation environment, which, apart from facilitating the implementation of the design, reduces possible errors occurring in realizing the design and supplements functionality to the design itself.

1.3 Multi Agent Controller

After introduction of ‘objects’, ‘agents’ is a frequently used term in software development. In general, an agent is referred as an abstract entity that is able to solve a particular (partial) problem. A complex problem can be solved by a pool of agents, in which each agent is responsible for solving a part of the whole problem, thus providing a well-structured problem solving approach. Since

multiple agents are acting on their particular problem to solve the complete complex problem, conflicts between individual agents arise, as these partial problems are interdependent. These conflicts are resolved by coordination between the agents.

Multi agent controllers are being implemented in solving control problems also. Van Breemen (2001) [23] has successfully implemented multi agent controllers in numerous control problems. Implementations presented also featured functionalities such as a hierarchical structure of controllers with multi-layered agents and efficient state transitions with ‘initialize’ and ‘finalize’ procedures. The framework incorporates bottom-up approach of design enabling incremental design of controllers. The ability of handling generic control problems is well suited for applying the framework of multi agent control in the integrated design and implementation of mechatronic systems. The structured framework of multi agent controllers can be exploited to ensure well-organized design and error free implementation.

1.4 Goals of the project

As mentioned earlier, the need for an integrated tool for design and implementation is inevitable for control system design, which supports the current developments in the field of controller design. Multi Agent Controller System has potential to implement advanced control systems, though an integrated support tool for Multi-Agent Controller System is still lacking. The main aim of this project is to develop and implement an integrated design and implementation tool based on Multi-Agent Controller System. The tool should be based on the Multi-Agent Controller Systems Implementation Framework (MACSIF) proposed by van Breemen (2001) [23]. The main task description of the project is summarized as follows.

- ❑ Redesign of the specification tool of Multi-Agent Controller Systems to support efficient and extensible specification.
- ❑ Build a design support tool, which can be used to test any designed Multi-Agent Controller System in a simulation environment.
- ❑ Build an implementation support, which allows automated implementation of the designed Multi-Agent Controller System on real systems.
- ❑ Incorporate the tools into a single integrated tool, which facilitates design and implementation of Multi-Agent Controller Systems.

1.5 IDITmac

IDITmac (Integrated Design and Implementation Tool for multi-agent controller) is the software tool developed in this project to support an integrated approach for Multi-Agent Controller Systems. A snapshot of the graphical user interface of IDITmac is shown in Figure 1-1.

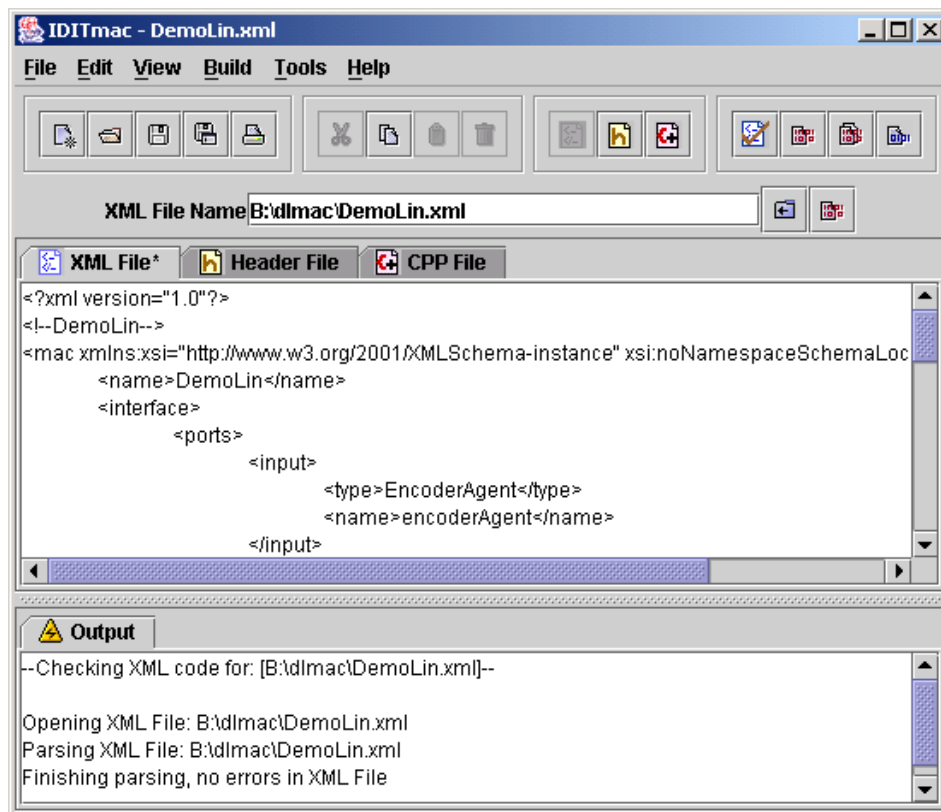


Figure 1-1: Snapshot of Graphical User Interface of IDITmac

IDITmac is developed in Java and it utilizes the Xerces 2 Java Parser [33] and the Microsoft Visual C++ Compiler. The tool supports XML editing, checking, generation of C++ code, generation of a dll (dynamic link library). The features of IDITmac are presented in Appendix C.

1.6 Outline of this thesis

Essential features of the implemented Multi-Agent Controller Systems are presented in this report. The organization of the report is as follows.

Chapter 2 presents a brief discussion of current trends in the field of control engineering and illustrates applications of Multi-Agent Controller Systems in this field. A need of an integrated approach is identified in the chapter and the proposed approach is presented, which combines design and implementation of Multi-Agent Controller Systems.

Chapter 3 describes the architecture of Multi-Agent Controller Systems. The description presented reflects the implemented system thus forms a basis for their specification. The architecture is presented in UML class diagrams.

Chapter 4 gives a brief description of MACSL (Multi-Agent Controller Specification Language), a specification language for Multi-Agent Controller Systems developed by van Breemen [23]. A guideline for the XML based specification developed in this thesis, MacsML (Multi-Agent Controller Specification Markup Language), is presented in the Chapter.

Chapter 5 introduces the operating principle and keywords of the implemented Multi-Agent Controller Systems. Frameworks for design and implementation of the integrated approach are presented in the chapter.

Chapter 6 presents a case study of Multi-Agent Controller Systems developed in this thesis to demonstrate the performance of the designed tool. The demonstration is supported by experiments performed on the case study.

Chapter 7 presents conclusion and recommendation for further improvement of the tool developed in this thesis.

CHAPTER 2

BACKGROUND OF THE PROJECT

2.1 Introduction

This chapter outlines the background of the project. The chapter describes current research trends in the field of controller architectures and agent based systems, currently available code generators and current trends of controller specification. A brief description of developments in controller architectures is presented with the focus in Mechatronic Systems. A study of agent based systems in the field of software development and control engineering is presented with examples of current applications of agents in control engineering. Current trends in automated code generation and controller specification are also studied. From the findings of current developments, an integrated approach of design and implementation of Multi-Agent Controller Systems is formulated. The integrated approach for controller is implemented in the project.

2.2 Overview of Methodologies for Controllers

This project unifies the concept of agents in software engineering and the principle of dividing a complex control problem to supposedly simple and independent control problems. By coordination between the agents, it combines the solutions of these simple control problems to solve the complex control problem. The concept of Multi-Agent Controller is based on the framework presented in “Agent-Based Multi-Controller Systems, A design framework for complex control problems”, A. J. N. van Breemen (2001) [23]. Main objective of the project is to propose a “control architecture” based on Multi-Agents, recommend specification of the agents and develop design tools (specification of the agents, validation of the specification, test of the multi-agent system by simulation) and implementation tools (software realization from the specification of the agents).

This project covers four developments in four major fields and unifies these developments and implements it to solve control problems of mechatronic systems. The four developments are listed below:

- ❑ Control Architecture
- ❑ Agents Based Software Development
- ❑ Code Generation
- ❑ Controller Specification

Current developments in above mentioned fields are narrated as follows.

2.2.1 Control Architecture

The conventional controller design framework mainly focuses upon designing a single-loop controller for a plant. In this framework, the single controller is designed based on a single dynamic model of the plant. The general structure of conventional controllers is shown in Figure 2-1.

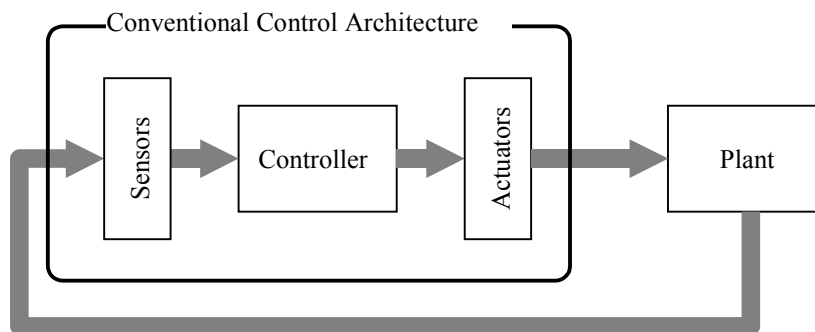


Figure 2-1: General structure for feedback controllers

However, modern mechatronic systems are becoming increasingly complex, such as manufacturing processes, power plants etc. A single dynamic model and a single loop controller solution for such a complex system is often becoming the bottleneck in the modern control application. Most of the modern plants can be described conveniently by multiple models, consisting of (simple) submodels that describe the plant's dynamic behavior in particular local regimes of the plant's overall operating regime [1] [2]. Various design tools based on operating regime decomposition are available for designing controllers, such as ORBIT (Operation Regime Based modeling and Identification Toolkit for Matlab) [1].

Recent developments in hybrid control systems [3] enable design and formal analysis of control systems for systems with changing dynamics (SCD). Different hybrid controller strategies that combine loop and logic control, such as switching controllers [4], are being studied. Various methodologies for analysis and design are presented for combining continuous time systems, described by differential/difference equations and discrete event systems, described by finite automata and Petri Nets [3].

Further, to simplify control problems, hierarchical control architectures are also proposed. In hierarchical control architectures, higher levels utilize coarser models of the system and lower levels deal with detailed models of the system. Several hierarchical models such as two-layered control hierarchy [5] [6], Open Control Platform with three-layered hierarchy [7] has been proposed.

Hierarchical structure is inherent in the controller structure of industrial processes. As illustrated in Figure 2-2, a seven-layered hierarchical structure is widely implemented in the industrial applications. Two base layers, Measurement and Actuation and Data Processing, are responsible for data acquisition and actuation. Loop controllers are generally feedback loop controllers and in special cases feedforward, adaptive controllers or other special controller algorithms. Generally, an industrial automation is complex to be controlled with a single loop controller. Coordination and sequencing of multiple loop controllers is done by sequence control, also known as logic control. Supervision of the operation of the controlled processes and data logging of the performance of the plant is also maintained. Further, a network interface is provided, which enables exchange of information between the controlled plants and Human Machine Interface (HMI).

2.2.2 Agents Based Software Development

An Agent is commonly used in software engineering and artificial intelligence as entity/program that are responsible for performing a set of tasks in coordination with its environment. Agents have been widely used in different contexts, ranging from generic autonomous agents, software agents and intelligent agents to the more specific interface agents, virtual agents, information agents, mobile agents and so on [8]. A definition of an autonomous agent by Franklin and Graesser (1997) is as follows [9].

“An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.”

In the field of control engineering also, autonomous agents are being implemented to solve various control problems [10] [11] [12] [13] [15]. Autonomous controller agents sense their environment, i.e. the plant, by sensors and react to the environment through actuators, to obtain a desirable performance. The ability of agents to execute their tasks in coordination with other agents make them highly effective for structuring distributed control architectures and hierarchical control systems.

Bianchi and Rillo (1996) [16], implemented a “purposive computer vision system” using agent based control architecture. Tasks are represented as basic agents and organized in a hierarchical structure with autonomous agents on the

top. Bussmann and Schild (2001) [17] implemented agents in a modern automotive industry using work-piece agents, machine agents and switch agents. The work-piece agents auction off their current task, while the machine agents bid for the tasks and the switch agent coordinates the work-piece and machine agents.

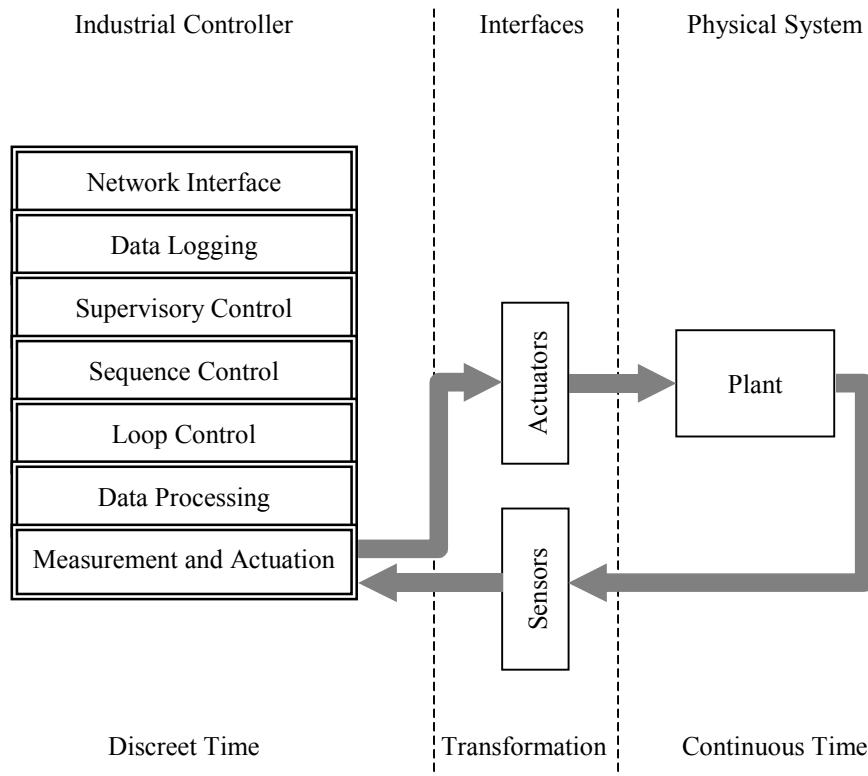


Figure 2-2: Hierarchical Structure of Industrial Controllers

Holonic Manufacturing Systems (HMS) [14] introduced a special type of physical agent called a “holons”, which consists of an information processing part and often a physical processing part, to provide an open environment for the manufacture of high-variety low-volume products. “Holons” is derived from the Greek word *holos* (signifying whole) with a suffix *on* (a particle, as in proton or neutron). Problem solving in an HMS is achieved by holarchies, or groups of autonomous and cooperative basic holons and/or recursive holons that are themselves holarchies [18].

Van Breemen (2001) [15] [19] [20] [21] [22] [23] proposed a framework for designing and implementing hierarchical structured multi-controller systems using an agent-based framework. In the presented framework, the concept of a controller-agent is introduced which contains all information of a particular control (sub-) problem [21] and the controller agency, which consists of a pool of controller-agents and a coordination object. The coordination object is responsible for resolving conflicts, sequential execution order and blending actions of the pool of controller-agents within the controller agency. Different types of coordination mechanisms such as Parallel, Fixed priority, Winner-

takes-all, Sequential, Addition have been described. Further, the external interface of a controller agency is the same as that of a controller agent, which allows multiple layered hierarchical structures of controller agencies and controller agents. The given framework is being implemented on various applications such as a Water Vessel Problem [19], Room Thermostat Control [20], Process of Corrugated Cardboard [22], Fast Component Mounter [23] and so on.

2.2.3 Code generation

Use of computer tools for design of control systems has been increasing in recent decades. Presently, various tools are available for computer aided control system design (CACSD) [24]. Some of the tools not only support analysis and design of controllers but also support automated code generation of the designed controllers for various target systems. Matlab [25] implements Real Time Workshop Embedded Coder and Stateflow Coder to generate C code directly from Real Time Workshop and Stateflow diagrams. Products such as dSpace [26] have been used for implementing code generated from Real Time Workshop and Stateflow diagrams. Software like MATX/RTMATX [27] also supports design of controllers and automated code generation from the designed controller. LabView [28] introduces hardware such as LabView Real-Time Model and RT series hardware targets, which can run applications, designed in LabView. 20-Sim [29] also generates ANSI C code of 20-Sim-models or 20-Sim-submodels that can be used in various systems. Standard templates are available to generate code for various applications. User defined templates can also be included to support customized code generation for a broad range of tasks.

20-Sim also supports linking during simulation through dll (Dynamic Link Library). The implementation of a control algorithm can be linked with 20-Sim through dll-functions to test the implementation.

2.2.4 Controller specification

Specifications of loop controllers are mostly presented by equations and graphical representations. The specification provides information of data flow and state transitions of the controller.

Since controllers are discrete event systems, they can be analyzed and synthesized by finite automata (state transitions), and Petri Nets. Petri Nets is a formal and graphical language, which is appropriate for modeling systems with concurrency. A preliminary proposal of standard of Petri Nets specification based on XML (Extensive Markup Language) is being developed [30]. PNML (Petri Nets Markup Language) is an interchangeable format for description of Petri Nets and being an extension of XML, it is portable and platform

independent. PNML has been implemented in industrial logic controller specification and design [31].

IEC 61499 Function Block is a component based open architecture, for Distributed Industrial-Process Measurement and Control Systems (IPMCS). IEC 61499 is derived from PLC Function Blocks (IEC 61131-3) and DCS Function Blocks (IEC 61804). A standard for XML based specification of IEC 61499 Function Block is also being developed [32].

For the multi agent controller framework of van Breemen (2001), an organizational diagram has been proposed for illustrating hierarchical structure of the agents[23]. However, the organizational diagram does not provide complete information of the agents, as data transfer between agents is not specified in the organizational diagram. The complete controller specification (which also includes the hierarchical structure) is defined in a specific text format.

2.3 Integrated Approach for Controllers

As illustrated in the previous section, various developments in different fields of control engineering are emerging to support the present needs of the controllers. Significant developments in the field of controller architectures provide sufficient support to simplify the design of complex control systems. The multi agent controller approach provides a backbone for implementing the available controller architectures in an effective and well-organized fashion. The framework of multi agent controller [23] accords hierarchical structuring of controllers, hybrid control systems with state transition and operating regime philosophy with mode switching. Controller specification and code generation from the specification ensure an error free and consistent implementation of the designed controllers.

Combination of these abilities in a single tool would provide a powerful approach for solving current control problems in an industrial environment. Thus a tool “Integrated Design and Implementation Tool for Multi-Agent Controller (IDITmac)” based on the framework of multi agent controller by van Breemen (2001) [23], is proposed in the project. The tool is primarily designed for combined use with 20-Sim (Design) [29] and 20-Works (Implementation). The designed controller is simulated with 20-Sim via a dynamic link library (dll) and implemented with 20-Works via object files. The structure of the tool is illustrated in Figure 2-3. Features of the integrated tool are elaborated below.

2.3.1 Competent Specification of Multi Agent Controller

In the multi agent controller framework, multi agent controllers are coordinated with each other and their environment to solve a given control problem. In order to design a good controller, the specification of the controller should be precise and well structured.

The specification language for multi agent controllers is defined in an XML (Extensible Markup Language) format. A guideline for the specification is designed, which specifies the structure of the XML specification. The guideline is implemented as W3C XML Schemas [34] and the controller specification is validated against the Schemas to check the integrity of the specification.

2.3.2 Support for Design of Multi Agent Controller

Design tools such as 20-Sim are commonly used for designing mechatronic controllers. 20-Sim is a Windows-based simulation program for dynamic systems. 20-Sim also supports run-time linking of simulation with a dynamic link library module. The specified multi agent controller is simulated in 20-Sim with a dynamic link library. 20-Sim is a powerful modeling tool for multi-domain systems as it also supports bond graphs and iconic diagrams.

A library is built in the Microsoft Visual C++ to capture the generic behavior of multi agent controllers. From the specification of the multi agent controller defined in XML, C++ code is generated. The generated code is compiled and linked to generate a dynamic link library module using the Microsoft Visual C++ compiler and linker. The generated dll (dynamic link library) can be linked with 20-Sim to simulate the defined Multi-Agent Controller. The process of translation of the specification of the controllers in XML to C++ code and generations of the dll from the C++ code is automated by a software tool developed in java.

2.3.3 Integrated Implementation of Multi Agent Controller

The implementation of Multi-Agent Controller Systems is accomplished in 20-Works. 20-Works is a C++ (GNU compiler) based platform designed for control of mechatronic systems at Control Laboratory, University of Twente. 20-Works operates in a fixed sampling period. It also provides a graphical user interface and online plotting facilities.

An integrated design and implementation approach is adopted for the implementation of Multi-Agent Controller Systems in which the code generated for design (with 20-Sim) is implemented on the real system. A framework is designed for 20-Works for implementing the code generated. The integrated approach reduces errors in translating the design to implementation.

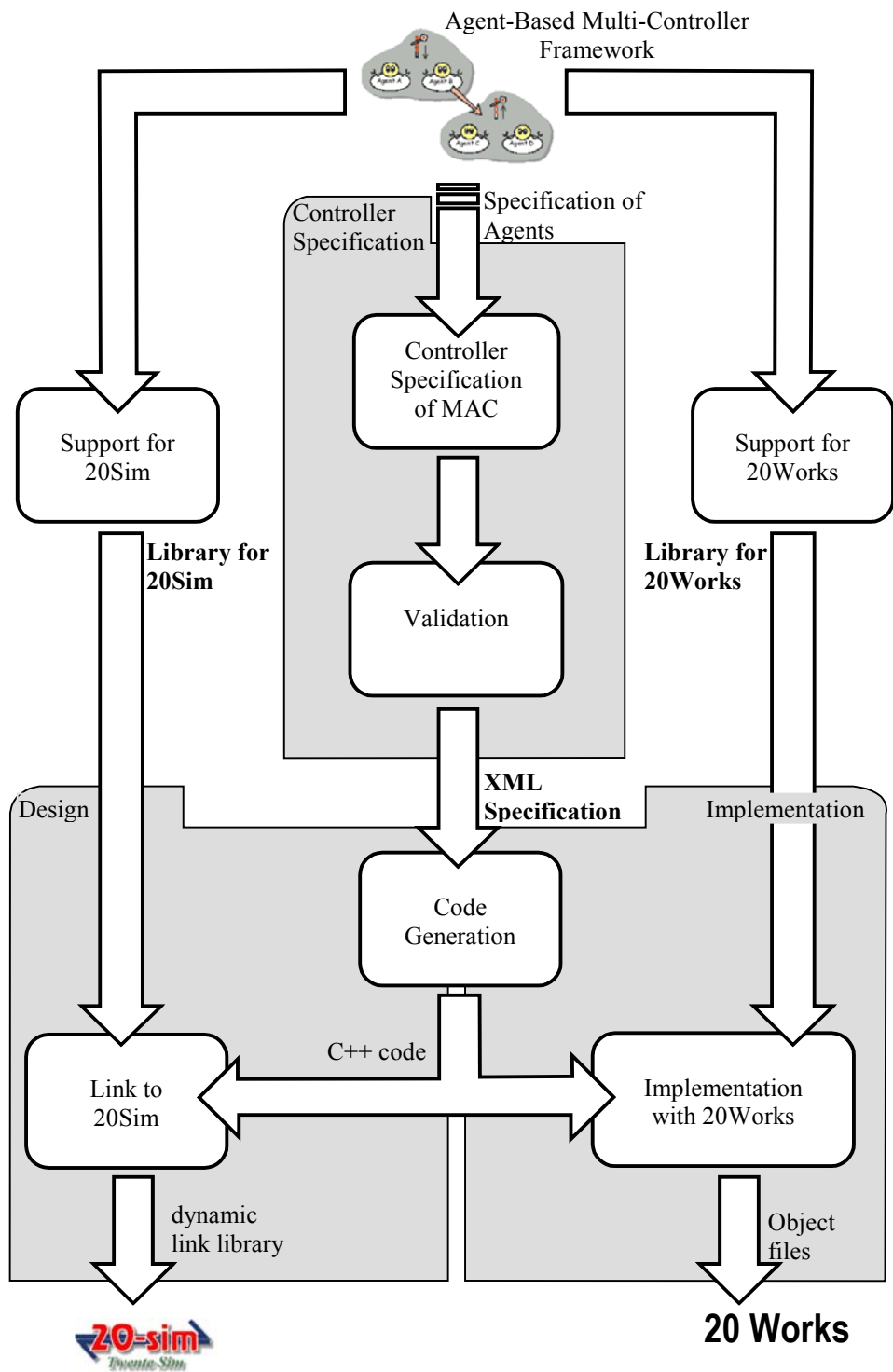


Figure 2-3: Structure of Integrated Design and Implementation Tool for Multi-Agent Controller (IDITmac)

2.4 Conclusion

In this section, current trends in controller design have been studied. Need of a tool for structuring the control strategy and providing an integrated approach in design and implementation of a control system to facilitate the current developments has been identified. Competence of Multi-Agent Controller Systems in organizing the control strategy has been demonstrated. Generic characteristics of Multi-Agent Controller Systems enable their application in almost all types of control systems. The flexible hierarchical structure of Multi-Agent Controller Systems facilitates appropriate structuring of the control solutions.

Based on Multi-Agent Controller Systems, a brief description of an integrated approach for design and implementation has been presented. The presented approach reduces potential errors that may occur during translation of the design to the implementation and provides a more realistic design approach.

CHAPTER 3

MULTI AGENT CONTROLLER SYSTEM ARCHITECTURE

3.1 Introduction

This chapter gives the basic theory underlying Multi-Agent Controller Systems. A general description of an agent is presented and an introduction of the Multi-Agent Controller Implementation Framework developed by van Breemen (2001) [23] is presented. The architecture of Multi-Agent Controller Systems implemented in the project is presented with the aid of UML (Unified Markup Language), which is also a basis for Specification of Multi-Agent Controller System (Chapter 4). The operating principle of a Multi-Agent System is presented in Chapter 5.2-Operating Principle of Multi Agent Controller System.

3.2 Agent

Recently, the term “agent” is widely used in various fields of software engineering. In general, “agent” is termed as an autonomous entity responsible for performing a certain task in coordination with its community. Since the term “agent” is used very frequently in various fields with different purposes, a unanimous definition of the term “agent” cannot be formulated. However, an agent in general can be described as a system, which exhibits the following properties.

Autonomous: an agent is an intelligent entity that can act upon its local task independently without the assistance from other systems.

Social ability: an agent communicates with its environment (which may be a community of other agents and/or the physical environment of the whole system) to achieve the global objective (of the whole system).

Reactivity: an agent perceives its environment and responds positively to the changes that occur in the environment.

The agent concept is not an absolute theory, which has a minimum requirement or a precise guideline that has to be followed to qualify for being an agent. The agent concept is a way of solving problems by dividing the solution of a complex problem into many autonomous and well-structured solutions and coordinating the well-structured solutions to achieve the goal. An agent should have a well-defined task description and should be able to perform its task by communicating with other agents and/or the physical environment.

Agents are extensively implemented in the field of artificial intelligence and distributed computing because of their autonomous and social ability. The configuration of the agents being implemented in these fields varies according to the problem to be solved and the approach taken to solve the given problem. However, the basic principles of the agents comply with the properties mentioned above.

3.3 Multi Agent Controller System

Apart from the field of artificial intelligence and the field of distributed computing, agent-based approach is being applied in control systems design also. Numerous control applications based on agents have been successfully developed and implemented [10] [11] [12] [13] [14] [15]. The architecture of the implementation of agents varies depending upon the problem description and the solution approach. However, the basic principle of autonomous agents with ability of communication is elemental in almost all the implementations.

A framework for implementing the agent-based concept in control engineering is proposed by van Breemen (2001) [23]. The “Multi-Agent Controller Implementation Framework” (MACIF) presents a generic backbone for implementing controllers as agents. The framework introduces controller agents, which can be integrated and coordinated efficiently to solve complex control problems [23]. The controller agents can be tailored to implement required controllers. The framework also presents a guideline for implementing these controller agents for solving general control problems.

The tool, “Multi-Agent Controller”, for the “Integrated Design and Implementation Tool”, developed in this project, is based on the “Multi-Agent Controller Implementation Framework” (MACIF). An overview of the Framework is presented in next section.

3.3.1 Overview of Multi-Agent Controller Implementation Framework (MACIF)

Control systems involve sensors, which measure data of the plant and actuators, which control the plant according to the given references. From the controller point of view, a controller has sensor data and references as inputs and actuation

signals as outputs. A Multi-Agent Controller consists of a single agent called “main agent” which constitutes different agents for reading input data (sensors and references), processing the input data (control algorithm) and outputting the actuation signal (actuators). The agents within the “main agent” are responsible for its local tasks and control the plant via communicating among each other. Six different primary components constitute the building blocks of the MACIF. These six agents are described briefly as follows.

Components

Sensor Agent: acquires data from the environment and dispatches the data to other agents. The environment involves the plant to be controlled and other external systems such as Human Interfaces (which gives references) or disturbances. In the discrete time implementation, Sensor Agents acquire these data from AD-converters (which acquire data from the environment).

Actuator Agent: dispatches the processed data to the environment. In the discrete time implementation, Actuator Agents dispatch data to DA-converters.

Composite Agent: consists of a group of Controller Agents (i.e. Elementary Agents and/or other Composite Agents) and a Coordination Object. Composite Agents enable layered structures of agents, as a Composite Agent may contain other Composite Agents.

Main Agent (Multi-Agent Controller): is the overall agent responsible for the whole control system. Main Agent is a kind of a Composite Agent so it consists of a group of Controller Agents and a Coordination Object. In addition, Main Agent also consists of a group of Sensor Agents and Actuator Agents, which interface with the environment. Main Agent is literally the main agent of a Multi-Agent Controller System. The Main Agent gets information of the plant to be controlled by its Sensor Agents and acts upon the plant via its Actuator Agents. A Multi-Agent Controller System has only one Main Agent, which is responsible for interfacing with the physical system (the controlled plant).

Coordination Object: is contained in all Composite Agents (and Main Agents). The task of a Coordination Object is to coordinate the behavior of the Controller Agents that exist within a Composite Agent (or a Main Agent).

Elementary Agent: is the fundamental agent of the MACIF and it implements local control solutions of the global control problem.

An example of a Multi-Agent Controller System is illustrated in Figure 3-1. The Multi-Agent Controller System has a Main Agent, “MA1”. “MA1” consists of two Sensor Agents (“SA1” and “SA2”), two Actuator Agents (“AA1” and “AA2”) and two Controller Agents (“EA1” and “CA1”). One of the Controller Agents (“EA1”) is an Elementary Agent and the other Controller Agent

(“CA1”) is a Composite Agent. “MA1” also comprises a Coordination Object “CO1”, which coordinates the Controller Agents of “MA1” (i.e. “EA1” and “CA1”).

The Composite Agent “CA1” has three Controller Agents (“EA2-1”, “EA2-2” and “EA2-3”). All the three Controller Agents of “CA1” are Elementary Agents. In addition, “CA1” has a Coordination Object “CO2” which coordinates the Controller Agents (“EA2-1”, “EA2-2” and “EA2-3”) of the Composite Agent “CA1”.

These six types of components form basic building blocks of a Multi-Agent Controller System. Composite Agents allow for a hierarchical structure of the Multi-Agent Controller System and Coordination Objects allow for proper coordination (activation) of the sub-agents within a Composite Agent (Main Agent). Within a Composite Agent, data transfer between its sub-Agents and data transfer from its sub-Agents to its external (input/output) ports takes place via connections. Connections recommended by the MACIF are as follows.

Connections

OI Connection: (Output to Input Connection). OI Connection is a channel for data transfer from an Output Port of a sub-agent to an Input Port of another sub-agent within a Composite Agent or a Main Agent.

EII Connection: (External Input to Input Connection). In a Composite Agent, an External Input Port is the Input Port of the Composite Agent. The data transfer channel from the External Input Port of the Composite Agent to an Input Port of one of the sub-agents of the Composite Agent is termed as EII Connection.

OEO Connection: (Output to External Output Connection). A connection from an Output port of one of the sub-agents of a Composite Agent to an External Output Port of the Composite Agent is termed as OEO Connection.

A connection from an External Input Port to an External Output Port of a Composite Agent is merely rerouting of data through a Composite Agent, thus is a nonfunctional connection. Thus, **EIEO Connection** is **not included** in the Framework.

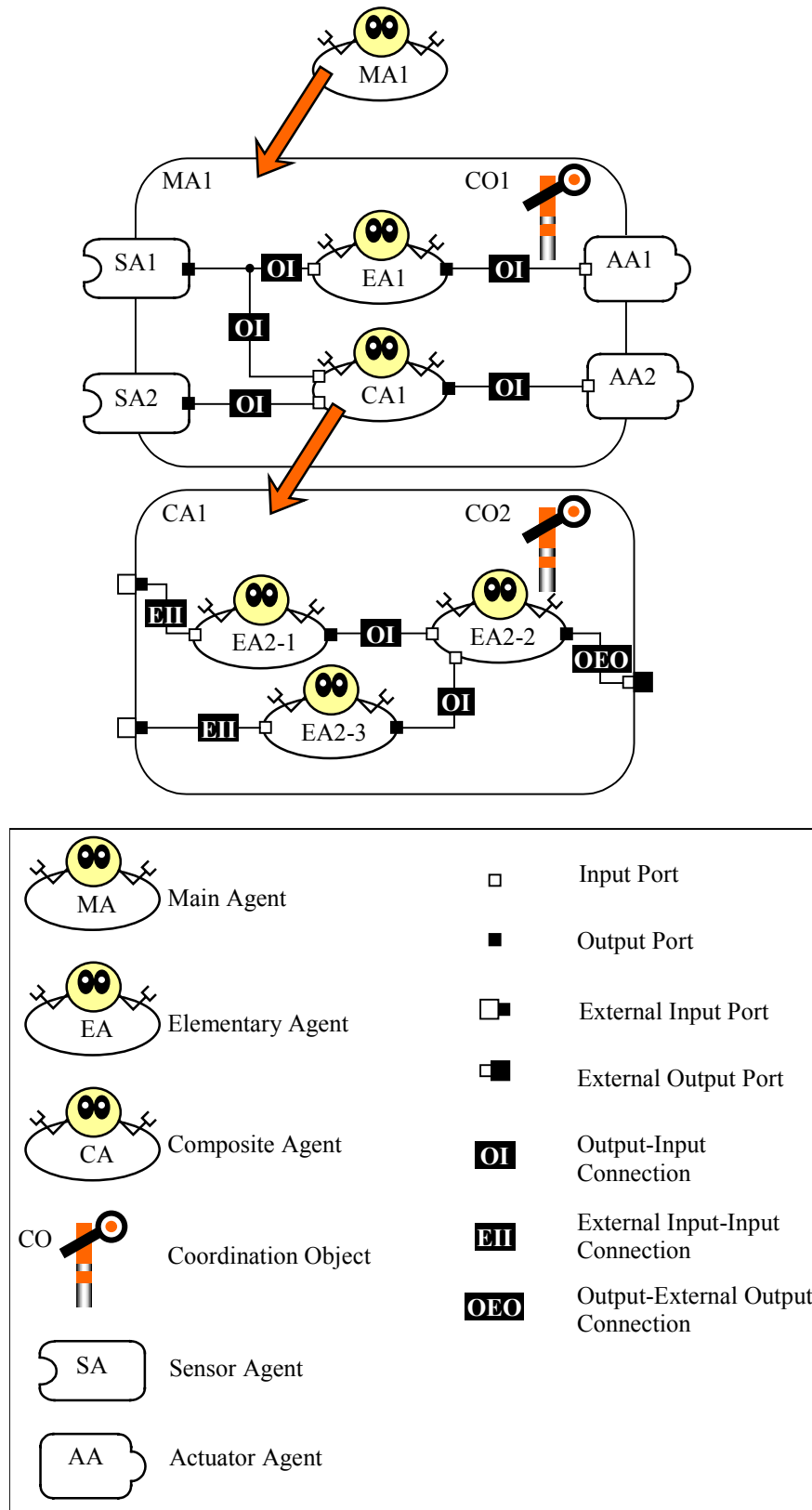


Figure 3-1: Organizational diagram of a Multi-Agent System

As indicated above, direct connections between the ports of a sub-Agent of a Composite Agent and the ports of a sub-Agent of another Composite Agent is not allowed. Any such connection should be accomplished through external ports of the corresponding Composite Agents.

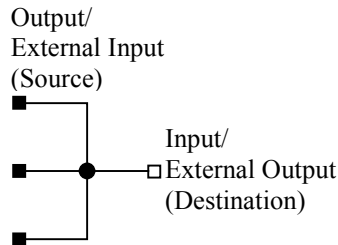


Figure 3-2: Multiple Source connected to single destination

In all the connections, the data types of the connecting ports should be the same. In addition, conflict situations may arise due to combinations of connections, which result in multiple source and single (or multiple) destination of the data (see Figure 3-2). For EII connections and OI connections, such ambiguous situations are prohibited. For OEO connections, coordination object coordinates OEO connections and decides the active connection (active source). The active connection changes dynamically according to the operating conditions (see Section 5.3.4).

3.4 Architecture of Multi Agent Controller System

The “Multi-Agent Controller Implementation Framework” consists of the six fundamental components, namely Sensor Agent, Actuator Agent, Composite Agent, Main Agent, Coordination Object and Elementary Agent. Based on the features of these components, the components can be categorized in a hierarchical structure as shown in the UML (United Markup Language) class diagram (Figure 3-3). A description of UML is presented in Appendix B.

In the Multi-Agent Controller System, Main Agent can be considered as an extension (specialization) of Composite Agent. In the object-oriented concept, Main Agent is considered as inherited from Composite Agent. In addition, Composite Agent and Elementary Agent can be categorized as Controller Agent. Controller Agent is an abstract (conceptual) class. In other words, Controller Agent itself is not one of the six base components of the Multi-Agent Controller System but it accommodates the common behaviors of Elementary Agent and Composite Agent. Finally, Sensor Agent, Actuator Agent, Controller Agent and Coordination Object can be considered as a sub type of Agent. Agent is also an abstract class.

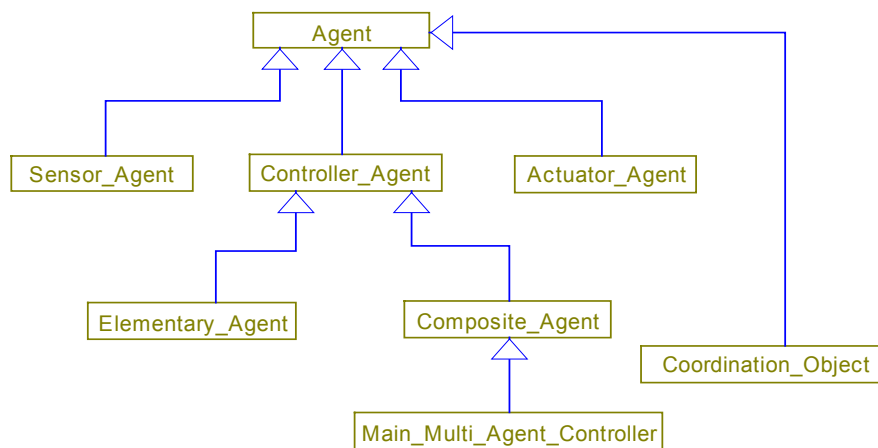


Figure 3-3: Generalization (inheritance) of Components of Multi-Agent System (UML class diagram)

The six fundamental components of the Multi-Agent Controller can be described in terms of the classification presented in Figure 3-3. Description of the classification is presented in a top-down approach. Higher classes (or categories) are presented beforehand. Sub classes inherit the architecture of their super class, so the features, already presented in their super class, are not reiterated in the sub classes.

3.4.1 Agent

An Agent is a generic component of the Multi-Agent Controller System. The architecture of an Agent is illustrated in Figure 3-4. An Agent has a name field (of string type), which identifies the agent. It also consists of a set of 'Input Ports' and a set of 'Output Ports'. 'Input Port' imports data from other agents whereas 'Output Port' exports data to other agents. Agents also have 'Parameters'. 'Parameters' are variables, which can be initialized while creating (instantiating) an Agent.

'States' are variables used within the Agent that retain a value over sample instants. An Agent has 'start():void' and 'stop():void' methods. Both the methods have a return type of 'void' and no arguments (no inputs to the methods). The 'start():void' method is the startup sequence of the Agent and the 'stop():void' method is the shutdown sequence of the Agent.

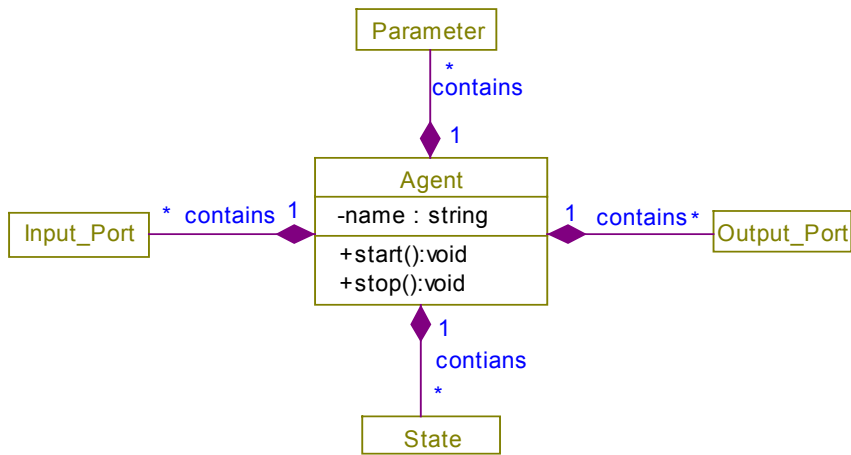


Figure 3-4: Architecture of Agent (UML class diagram)

3.4.2 Sensor Agent

Sensor Agent is one of the base components of the Multi-Agent Controller System. Sensor Agent specializes its super class, Agent, and has an additional method ‘sense():void’. The ‘sense():void’ method acquires data from AD-Converter (or the environment). The architecture of Sensor Agent is shown in Figure 3-5(a).

3.4.3 Actuator Agent

Actuator Agent is a base component of the Multi-Agent Controller System and is a kind of Agent. Actuator Agent has an additional method ‘actuate():void’. The ‘actuate():void’ method transfers data to DA-Converter (or the environment). The architecture of Actuator Agent is shown in Figure 3-5 (b).



Figure 3-5: Architecture of (a) Sensor Agent (b) Actuator Agent (UML class diagram)

3.4.4 Controller Agent

Controller Agent is an abstract Agent (i.e. not one of the base components of the Multi-Agent Controller System). Controller Agent specializes Agent. In addition, Controller Agent has a boolean attribute ‘operating_state’. ‘operating_state’ indicates the operating state of the Controller Agent. If ‘operating_state’ is true, then the Controller Agent is active else the Controller Agent is inactive. Controller Agent has two additional methods ‘acknowledge(boolean ack):void’ and ‘activation():real’. The first method has a boolean argument ‘ack’ and a return type of ‘void’ and the second method has

no arguments and a return type of 'real'. The 'activation():real' method processes the activation request of the Controller Agent and returns the degree of activation of the Controller Agent. The degree of activation ranges from 0 to 1, 0 indicating the Controller Agent wants to be inactive. The 'acknowledge(boolean ack):void' method receives acknowledgement of the activation request. The Controller Agent is activated if ack is true. The architecture of Controller Agent is shown below.

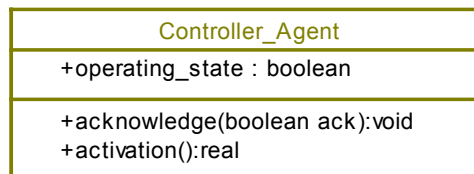


Figure 3-6: Architecture of Controller Agent (UML class diagram)

3.4.5 Elementary Agent

Elementary Agent is one of the base components of the Multi-Agent Controller System. Elementary Agent is a kind of Controller Agent hence inherits all the features of Controller Agent. In addition, Controller Agent has four methods 'initialize():void', 'finalize():void', 'calculate():void' and 'update():void'. The 'initialize():void' method is executed when an inactive Elementary Agent becomes active and the 'finalize():void' method is executed when an active Elementary Agent becomes inactive. The 'calculate():void' method is executed when the Elementary Agent is active and it performs the necessary calculation of an active Elementary Agent. The 'update():void' method is executed both in active and inactive state of the Elementary Agent and it normally updates the 'State's of the Elementary Agent. The architecture of Elementary Agent is illustrated below.

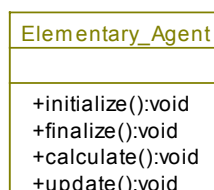


Figure 3-7: Architecture of Elementary Agent (UML class diagram)

3.4.6 Composite Agent

Composite Agent is one of the base agents of the Multi-Agent Controller System. Composite Agent is also a kind of Controller Agent thus inherits all the behaviors of Controller Agent. Composite Agent comprises a set of Controller Agents (sub Agents). Composite Agent has a Coordination Object, which coordinates its Controller Agents (sub Agents). Composite Agent also contains a set of 'OI Connection's, 'EII Connection's and 'OEO Connection's. The architecture of Composite Agent is illustrated in the figure below.

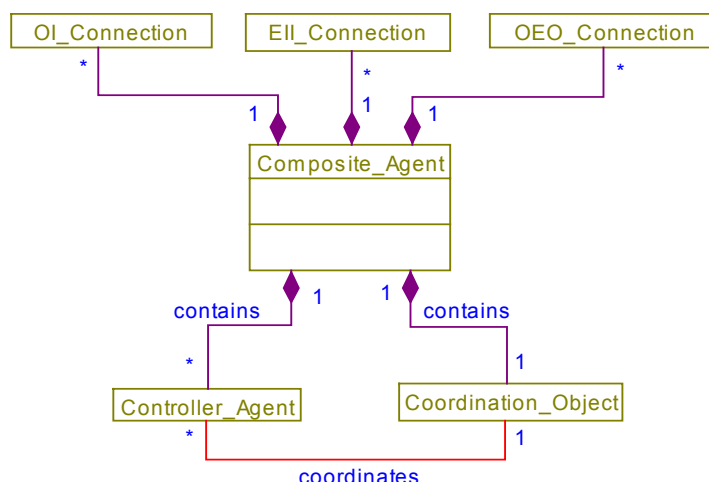


Figure 3-8: Architecture of Composite Agent (UML class diagram)

3.4.7 Coordination Object

Coordination Object is also one of the base components of the Multi-Agent Controller System. Apart from the inherited behaviors from its super class, Agent, Coordination Object consists of three additional methods ‘resolute():real’, ‘decide(boolean ack):void’ and ‘combine():void’. Coordination Object exists within a Composite Agent (and also in a Main Agent) as illustrated in Figure 3-8. The ‘resolute():real’ method processes activation requests of Controller Agents (sub Agents) of the Composite Agents (or the Main Agent) and returns the degree of activation of the Composite Agent (or the Main Agent). The ‘decide(boolean ack):void’ decides upon the activation of the Controller Agents (sub Agents) of the Composite Agent (or the Main Agent). The ‘combine()’ method combines the output of the Controller Agents. The architecture of Coordination Object is presented in Figure 3-9.

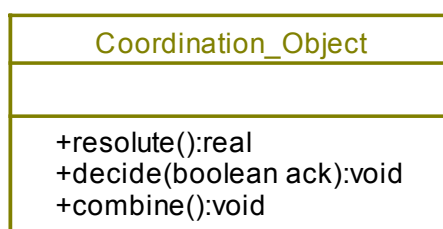


Figure 3-9: Architecture of Coordination Agent (UML class diagram)

3.4.8 Main Agent

Main Agent is one of the base components of the Multi-Agent Controller System. Main Agent is a kind of Composite Agent and has all the behaviors of a Composite Agent. As Main Agent is the prime agent of any Multi-Agent Controller System, Main Agent interacts with the environment of the system through its ‘Sensor Agent’s and ‘Actuator Agent’s. Main Agent has an additional method ‘tick():void’. The ‘tick():void’ method is called at every

sampling instant of the Multi-Agent Controller System. The architecture of Main Agent is illustrated below.

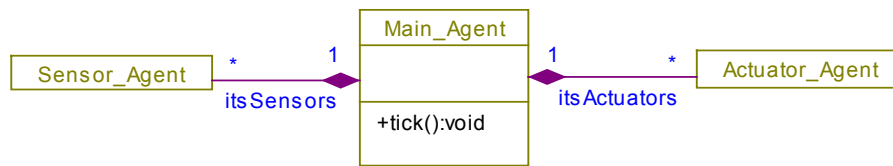


Figure 3-10: Architecture of Main Agent (UML class diagram)

3.5 Conclusion

An introduction of Multi-Agent Controller Systems as implemented in this thesis has been presented in this section. The architecture of the Agents has been presented, which forms a basis for the specification of Multi-Agent Controller Systems. The architecture of Agents has been structured in a hierarchical order and has been presented in UML notations, which forms a formal documentation of the Agents.

CHAPTER 4

SPECIFICATION OF MULTI AGENT CONTROLLER SYSTEM

4.1 Introduction

A specification language for Multi-Agent Controller Systems has been developed in this project. The specification language is named as Multi-Agent Controller Specification Markup Language (MacSML). MacSML is based on Extensive Markup Language (XML). A brief description of MacSML is presented in this chapter. The structure of MacSML is defined in XML Schemas and a brief introduction of the Schemas developed is presented in this chapter.

A brief introduction of another specification language, Multi-Agent Controller Specification Language (MACL), developed in van Breemen (2001) [23], is also presented in this section. In addition, an introduction to the tools support by IDITmac (tool developed in this project) for MacSML is also presented.

4.2 Multi-Agent Controller Specification Language

Multi-Agent Controller Specification Language (MACSL) is a specification language for Multi-Agent Controller recommended by van Breemen (2001) [23]. The specification of a Multi-Agent Controller System is described in a specific format in a text file. The six base components of Multi-Agent Controller Systems are defined by six different keywords namely, 'sensor', 'actuator', 'cagent', 'coordination', 'mac' and 'cagency'. Van Breemen (2001) [23] also describes a set of keywords, which can be used in the specification.

A simple example of an Elementary Agent implementation of a P-Controller (Proportional Controller) is schematically represented in Figure 4-1. The P-Controller has inputs, outputs, parameters and a 'calculate():void' method as listed in the figure below.

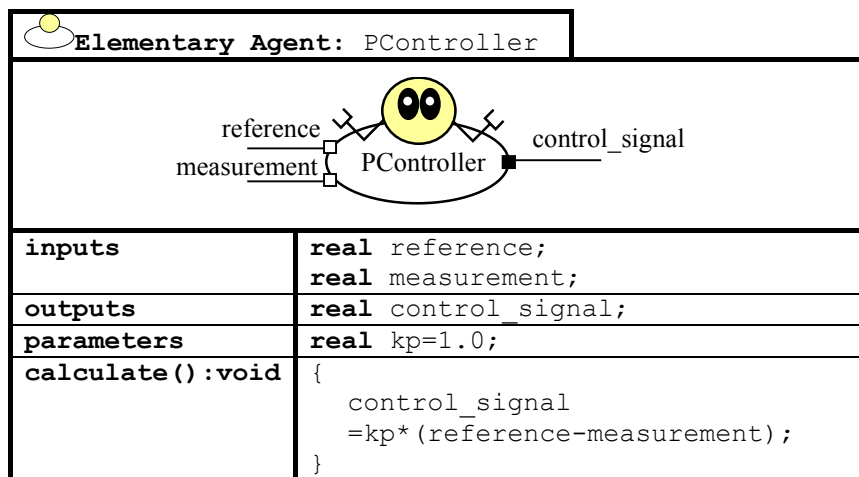


Figure 4-1: Schematic representation of an Elementary Agent, PController

MACL specification the P-Controller is presented in Table 4-1. The specification is in a plain text format though **boldface** typesetting is used to highlight keywords of MACL.

Table 4-1: An Example of MACL (specification of the P-Controller in Figure 4-1)

```

PController.msf
cagent PController;
/*Specification of a MACS implementation of
P-Controller in MACSL*/
  inputs
    real reference;
    real measurement;
  outputs
    real control_signal;
  parameters
    real kp=1.0;
  calculate {
    control_signal=kp*(reference-measurement);
  }
end;
    
```

4.3 Multi-Agent Controller Specification Markup Language (MacsML)

A specification of Multi-Agent Controller Systems named as Multi-Agent Controller Specification Markup Language (MacsML) has been developed in this project. The specification, MacsML is base on the architecture of Multi-Agent Controller Systems present in Section 3.4) Architecture of Multi Agent Controller System. In the specification, Multi-Agent Controller Systems are described in an XML file. Collection of the XML specification files defines the Multi-Agent Controller System. The structure of the XML specification is defined in XML Schemas.

Advantages of the use of XML as a specification format in the MacsML are listed below.

Standard: XML is a standard Markup Language, defined by W3C recommendation [35]. XML is becoming increasingly popular in the field of software engineering and it is applied in almost all new software development projects.

Extensibility: XML can be easily extended to add functionality. New tags can be defined and implemented without changing much of the existing systems. Thus, any additional functionality of the Multi-Agent Controller System that might be introduced in future could be easily incorporated.

Portability: The XML recommendation is platform independent and is supported by all major computing platforms available. XML is becoming a common markup language over the Internet, which will facilitate possible future deployment of the Multi-Agent Controller System to multiple platforms over the Internet.

Structure: XML documents are well structured with Markups thus less susceptible to errors during processing of the document.

Usability: XML Schema and Document Type Definition (DTD) can be used to define and check the structure of the XML documents. Various ready-to-use and proven parsers such as Simple API for XML (SAX), Document Object Model (DOM) parsers are available for parsing (processing) XML documents against the defined Schemas. In addition, automatic generation of XML specifications, for example from a graphical user interface, is quite simple with the tools mentioned.

A MacsML specification of the P-Controller (schematically illustrated in Figure 4-1) is illustrated in Table 4-2. Similar to MACL, MacsML also uses a plain text format. In the example, **boldface** typesetting is used to highlight XML tags. The specification specifies the name, input ports, output port, parameter and 'calculate():void' method of the Elementary Agent. A Schema CagentSchema.xsd is assigned to the specification (PController.xml, Line: 6, *xsi:noNamespaceSchemaLocation="CagentSchema.xsd"*). The structure of the document is according to the structure defined by the Schema. A description of the Schemas used in MacsML is presented in the next section. The MACL specification of the same P-Controller is presented in Table 4-1.

Table 4-2: An Example of MacsML (specification of the P-Controller in Figure 4-1)

```

PController.xml
<?xml version="1.0"?>
<!--*Specification of a MACS implementation of
P-Controller in MacsML-->
<cagentclass
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">

  <name>PController</name>

  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>reference</name>
      </input>
      <input>
        <type>real</type>
        <name>measurement</name>
      </input>
      <output>
        <type>real</type>
        <name>control_signal</name>
      </output>
    </ports>
    <parameterdefs>
      <parameterdef>
        <type>real</type>
        <name>kp</name>
        <defaultvalue>1.0</defaultvalue>
      </parameterdef>
    </parameterdefs>
  </interface>

  <implementation>
    <elementary>
      <calculate>
        <![CDATA[
{
  control_signal=kp*(reference-measurement);
}
]]>
      </calculate>
    </elementary>
  </implementation>
</cagentclass>

```

4.3.1 XML Schema for MacsML

As the specification of the Agent is described in an XML document, XML Schemas have been developed which define the structure of the specification. Three Schemas, namely CagentSchema.xsd, CoordinationSchema.xsd and MacSchema.xsd, have been developed. CagentSchema.xsd defines the structure

of the specification of Sensor, Actuator, Elementary and Composite Agents. CoordinationSchema.xsd describes the structure of the specification of Coordination Objects and MacSchema.xsd defines the structure of the specification of Main Agents. Important descriptions of the structure of the MacsML are presented in following sections.

4.3.2 Basic Structure of MacsML

Every XML document should have a root element and all the other tags should be nested within the root element. A common root tag `<cagentclass>` is being used for Sensor Agents, Actuator Agents, Elementary Agents, Composite Agents and Coordination Object (CagentSchema and CoordinationSchema). For Main Agents (MacSchema), another root tag `<mac>` is introduced. The root tag of a specification differentiates Main Agent from other components, as Main Agent is a special agent of Multi-Agent Controller Systems, which interacts with the environment (the plant).

As illustrated in Table 4-3, Table 4-4 and Table 4-5, both the `<cagentclass>` and `<mac>` elements have a sequence of nested elements of `<name>`, `<interface>` and `<implementation>`. The `<name>` is an elementary node, which should contain the name of the agent. The `<interface>` node contains information of interface of the agent to other agents or in the case of Main Agent, the environment (the plant). The `<interface>` node is optional for CoordinationSchema, as Coordination Object may not require declaration of an interface, as the ports of a Coordination Object are defined implicitly to facilitate dynamic size of ports (see section comment).

The implementation of agents is described within the `<implementation>` node. The `<implementation>` node of Sensor, Actuators, Elementary and Composite Agents have a `<sensor>`, `<actuator>`, `<elementary>` and `<composite>` sub node respectively. The `<implementation>` node of Coordination Object has a `<coordination>` sub node and Main Agent has a `<composite>` sub node. The implementation of agents is nested within the respective sub nodes.

Table 4-3: Basic Structure of specification of Sensor, Actuator, Elementary and Composite Agents (CagentSchema)

```

<cagentclass
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>.....</name>
  <interface>
  .
  .
</interface>
  <implementation>
  .
  .
</implementation>
</cagentclass>

```

Table 4-4: Basic Structure of specification of Coordination Agent (CoordinationSchema)

```

<cagentclass
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CoordinationObject.xsd">
  <name>.....</name>
  <interface>
  .
  .
  </interface>
  <implementation>
  .
  .
  </implementation>
</cagentclass>
    
```

Table 4-5: Basic Structure of specification of Main Agent (MacSchema)

```

<mac
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MacSchema.xsd">
  <name>.....</name>
  <interface>
  .
  .
  </interface>
  <implementation>
  .
  .
  </implementation>
</mac>
    
```

4.3.3 Input Ports and Output Ports

Input Ports and Output Ports of an agent are described by *<input>* and *<output>* tags. Type and name of an Input Port or Output Port is defined in the nested tags, *<type>* and *<name>*. For Sensor, Actuator, Elementary, Composite and Coordination Agents, the *<name>* node contains the name of the variable of the port and the *<type>* node contains the type of the variable. Variables defined in *<input>* node and *<output>* node can be referred to in the implementation of the specification. The only variable type currently implemented for the ports of Multi-Agent Controller Systems is ‘real’ (or equivalent C++ variable declaration, ‘double’). An example of an input port and output port declaration for an agent is shown in Table 4-6.

Table 4-6: Input Port and Output Port declaration of an agent

<pre> <input> <type>real</type> <name>position</name> </input> </pre>	<pre> <output> <type>real</type> <name>current</name> </output> </pre>
---	--

For Main Agents, the type of an Input Port is class name of a Sensor Agent and the type of an Output Port is the class name of an Actuator Agent. The name

field contains the instance name of the Agent. An example of an input port and an output port declaration for a Main Agent is shown in Table 4-7.

Table 4-7: Input Port and Output Port declaration of an agent

<pre> <input> <type>TwenteSensor</type> <name>positionSensor</name> </input> </pre>
<pre> <output> <type>TwenteActuator</type> <name>currentSensor</name> </output> </pre>

4.3.4 Parameters Definitions and States

Parameter definitions and states are described by `<parameterdef>` and `<state>` tags respectively. Similar to input and output ports, they also have `<type>` and `<name>` sub nodes. In addition, a parameter definition has a `<defaultvalue>` tag, which is the initial value of the parameter when the agent is instantiated. Parameters and States can have a variable type of 'real', 'boolean' or 'Int' (or equivalent C++ variable declaration, 'double', 'bool' or 'int'). An example of a parameter definition and state declaration is shown in Table 4-8 and Table 4-9 respectively.

Table 4-8: Parameter Definition declaration of an agent

<pre> <parameterdef> <type>real</type> <name>kd</name> <defaultvalue>1.0</defaultvalue> </parameterdef> </pre>
--

Table 4-9: State declaration an agent

<pre> <state> <type>int</type> <name>cycle</name> </state> </pre>

4.3.5 Method Specification

The implementations of the methods in the Multi-Agent Controller System are defined in C++ code and are contained inside a CDATA section (as C++ code may contain special characters which would otherwise be recognized as markup). The input ports, output ports, states and parameter definitions are considered as declared variables and can be referred to inside the code. In addition, methods should return a value of the correct return type. An example of an activation method specification is illustrated in Table 4-10. The activation method returns either 1.0 or 0.0 (real type).

Table 4-10: Declaration of activation method an agent

```

<activation>
<![CDATA[
    {
        if (load_position>Max_Load_Pos_Limit)
            return 1.0;
        else
            return 0.0;
    }
]]>
</activation>

```

4.3.6 Controller Agents within Composite or Main Agent

Controller Agents in a Composite or Main Agent are specified by *<cagent>* tag. The *<type>* tag of the specification consists of the type of the Controller Agent and the *<name>* tag consists of the instance name of the Controller Agent. In addition, the *<parameters>* tags consist of initialization of parameters, which include the name of the parameter and its initial value. An example of a declaration of a controller agent is illustrated in Table 4-11.

Table 4-11: Declaration of a Controller Agent within a Composite Agent

```

<cagent>
  <type>PosLimit</type>
  <name>posLimit</name>
  <parameters>
    <parameter>
      <name>kp</name>
      <value>25</value>
    </parameter>
    <parameter>
      <name>kd</name>
      <value>22</value>
    </parameter>
  </parameters>
</cagent>

```

4.3.7 Coordination within Composite or Main Agent

Similar to the declaration of Controller Agents, a Coordination Agent declaration also has a *<type>* and *<name>* tag. However, only one Coordination Object is allowed within a Composite or Main Agent. An example of declaration of a Coordination Object is presented in Table 4-12.

Table 4-12: Declaration of Coordination Object in some agent

```

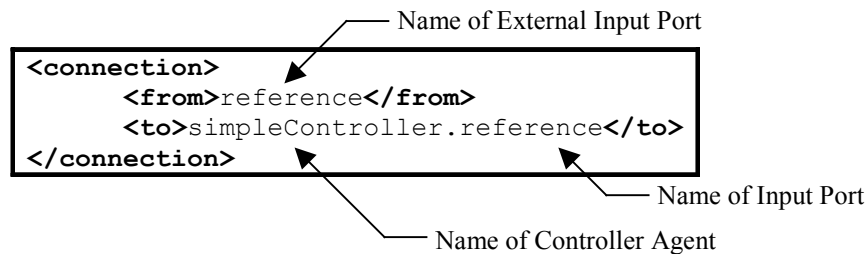
<coordinationclass>
  <type>FixedPriority</type>
  <name>fixedPriority</name>
</coordinationclass>

```

4.3.8 Connections within Composite or Main Agents

Connections within Composite or Main Agent are specified in `<from>` and `<to>` tags. The `<from>` tag contains the source of the connection, an External Input or an Output, whereas the `<to>` tag contains the destination of the connection, an Input Port or an External Output Port. As mentioned in Section 3.3.1, an EIEO connection is non functional thus connections from an External Input to an External Output are not permitted. External Input Ports and External Output Ports are specified by their name. Input ports and output ports of controller agents within a composite or main agent are specified by name of the controller agent followed by a dot and the name of the port. An example of a connection between an external input port, 'reference', and an input port, 'reference', of a controller agent, 'simpleController', is illustrated in Table 4-13.

Table 4-13: Declaration of connection



4.3.9 Including Header Files

Header files, which are required for the specification of an agent, can be enclosed within `<include>` tags. The specification can have multiple includes. An example of the `<include>` tag is presented in Table 4-14. In the example, a header file "cmath" is included in the specification, which is required for use of the "fabs" function.

Table 4-14: Declaration of inclusion of a header file

```

<include>cmath</include>
.
.
<activation><![CDATA[ {
  return ((double) (fabs (error) <maxError) );
}]]></activation>

```

4.3.10 Instances of C++ Classes

Instances of C++ classes can also be included in the specifications of Agents. Declaration of instances is specified within `<instances>` tags as illustrated in Table 4-15. The name of the class is included in a `<type>` tag and the name of its instance is included in a `<name>` tag. The instance is treated as a pointer instance, it is created with a constructor with no arguments, and it is deleted when the Agent is destroyed.

Table 4-15: Declaration of instance of C++ class

```
<instances>
  <instance>
    <type>EncoderInterface</type>
    <name>encInt</name>
  </instance>
</instances>
```

4.4 Conclusion

The specification language, MACSL, developed by van Breemen [23] has been briefly introduced and development of the XML based specification language, MacsML, has been presented. MacsML improves the structure of the specification of Multi-Agent Controller Systems and provides a basis for future extensions. The Schemas designed define the structure of MacsML. These Schemas can be easily extended to support additional functionalities of Multi-Agent Controller Systems.

The support provided by IDITmac for creating specifications and checking the integrity of specifications has been discussed in Appendix C.1. IDITmac extracts the information of an agent from its MacsML specification to generate code for the agent. Furthermore, IDITmac can be extended to support a graphical specification of Multi-Agent Controller Systems in which the graphical specification can be translated to a MacsML specification.

CHAPTER 5

DESIGN AND IMPLEMENTATION OF MULTI-AGENT CONTROLLER SYSTEMS

5.1 Introduction

Multi-Agent Controller Systems are implemented in ANSI C++. The operating principle of Multi-Agent Controller Systems is presented in this section. Based on the operating principle, a C++ library is developed for Multi-Agent Controller Systems. The code generated for a Multi-Agent Controller System is combined with the library and simulated with 20-Sim and implemented with 20-Works in real systems. The interface of 20-Sim and 20-Works with Multi-Agent Controller Systems is presented in this section. In addition, keywords used in the specification of agents are briefly discussed.

5.2 Operating Principle of Multi Agent Controller Systems

5.2.1 Multi-Agent Controller Systems and the Environment

The developed Multi-Agent Controller Systems should support design and implementation of a control system, thus a Multi-Agent Controller System has two different environments. In the design phase, the Multi-Agent Controller System acts upon a model of a plant. The Multi-Agent Controller System interacts with the model through a simulation program, in this case 20-Sim. Therefore, 20-sim is the environment (also called Actor in UML terminology) for the Multi-Agent Controller System in its design phase. Implementation of Multi-Agent Controller Systems is accomplished in corporation with 20-Works thus it is the environment of Multi-Agent Systems for the implementation phase. Multi-Agent Systems interact with the plant to be controlled through 20-Works. The environments of Multi-Agents Controller Systems are also illustrated in the schematic diagram in Figure 5-1.

Multi-Agent Controller Systems have a common interface with their environments. A Multi-Agent Controller System has a Main Agent, which is responsible for interacting with its environment. The Main Agent has Sensor Agents and Actuator Agents to transfer data from or to the environment.

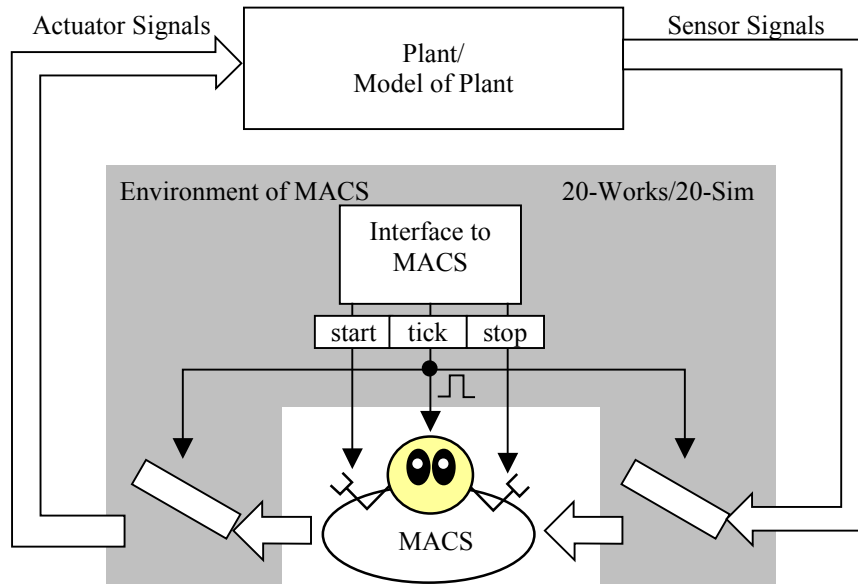


Figure 5-1: Schematic diagram of Multi-Agent Controller Systems

The Main Agent has three main operations namely ‘start():void’, ‘tick():void’ and ‘stop():void’, as defined in the framework of Multi-Agent Controller Systems, which are invoked by the environment. The ‘start():void’ method is called during the startup of the Multi-Agent Controller System. This method initializes the Multi-Agent Controller System. The Controller System is implemented as a discrete system with a fixed sampling period. At each sampling instant, the sensor signals and actuator signals of the plant are sampled when the ‘tick():void’ method is invoked. The ‘stop():void’ method is called during the shutdown of the Multi-Agent Controller System and it runs the shutdown sequence of the System.

A sequence diagram of the Multi-Agent Controller System is presented in Figure 5-2. The environment (20-Sim or 20-Works) calls the ‘start():void’ method in the startup of the system. Each sampling event of the environment invokes the ‘tick():void’ method. Finally, at the shutdown of the system, the ‘stop():void’ method is called. A detailed description of the operating principle of 20-Sim and 20-Works is presented in section 5.4 and section 5.5 respectively.

5.2.2 Main Agent

Main Agent has sub agents of type Sensor Agent, Actuator Agent, Coordination Object and Controller Agent. As described in the previous section, the environment invokes three operations of Main Agent. Each of these events triggers different events of its sub agents. Sequence diagrams of each of the

three events are presented in Figure 5-3 (a), (b) and (c). These sequence diagrams can be considered as cascaded with the main sequence diagram of the Multi-Agent Controller System presented in Figure 5-2. In a Main Agent, Controller Agents are coordinated by its Coordination Object as described in Section 5.2.5.

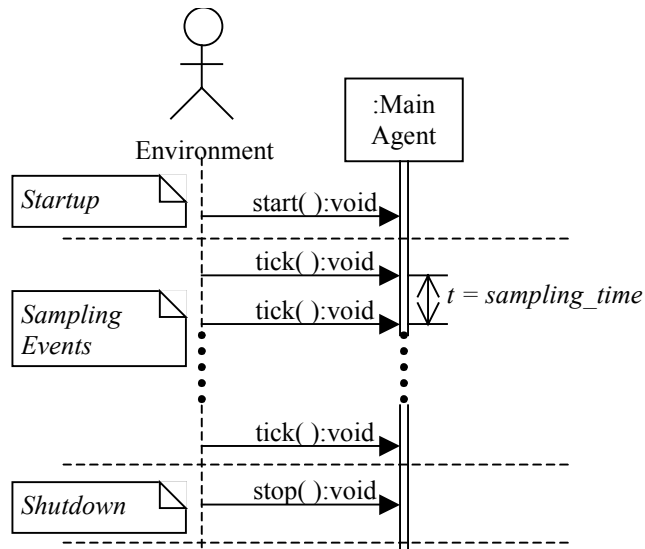


Figure 5-2: Sequence diagram of operation of MACS

In addition, Main Agent could have multiple Sensor Agents and Actuator Agents. Main Agent invokes methods of each of the Agents. The order of execution of methods is the same as the order of definition of the Agents in their specification. For example, in the specification of a Main Agent ‘MA1’, Sensor Agents ‘SA1’ and ‘SA2’ are defined consecutively (as illustrated in Table 5-1). The start method of MA1 will invoke the start method of SA1 first and then the start method of SA2.

Table 5-1: A part of Specification of a Main Agent ‘MA1’

```

Main Agent: MA1.xml
<name>MA1</name>
<interface>
  <ports>
    <input>
      <type>SensorType1</type>
      <name>SA1</name>
    </input>
    <input>
      <type>SensorType2</type>
      <name>SA2</name>
    </input>
  </ports>
</interface>

```

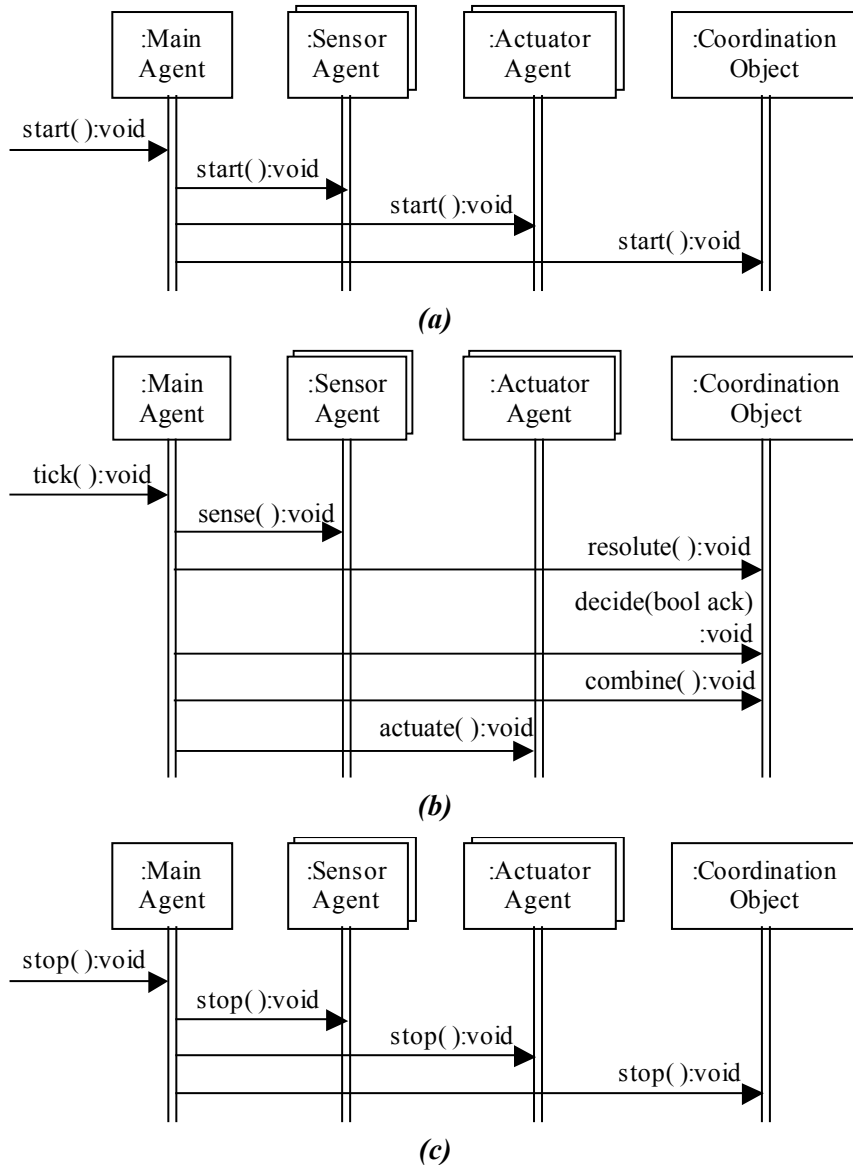


Figure 5-3: Sequence diagrams of Main Agent

5.2.3 Sensor Agent

Sensor Agent is a basic agent, thus does not have sub agents. Thus, the event received by Sensor Agent does not trigger other events. Sensor Agent has three events namely, ‘start():void’, ‘sense():void’ and ‘stop():void’. These methods execute the code sequence stated in its specification. The ‘sense():void’ method contains code sequence to acquire sensor data from the environment. Other two methods run the startup and shutdown sequence of the agent.

5.2.4 Actuator Agent

Actuator Agent has ‘start():void’, ‘actuate():void’ and ‘stop():void’ methods. These methods are stated in the specification of the agent. The ‘actuate():void’

method provides actuator signals to the environment. The other two methods run the startup and shutdown sequence of the actuator.

5.2.5 Coordination Object

Coordination Object is contained within a Composite Agents or Main Agent. Composite or Main Agent have a set of Controller Agents and Coordination Object is associated with the Controller Agents. Events of Coordination Object trigger events of the Controller Agents. Sequence diagrams of Coordination Object are shown in Figure 5-4. Controller Agent could be a Composite Agent or an Elementary Agent.

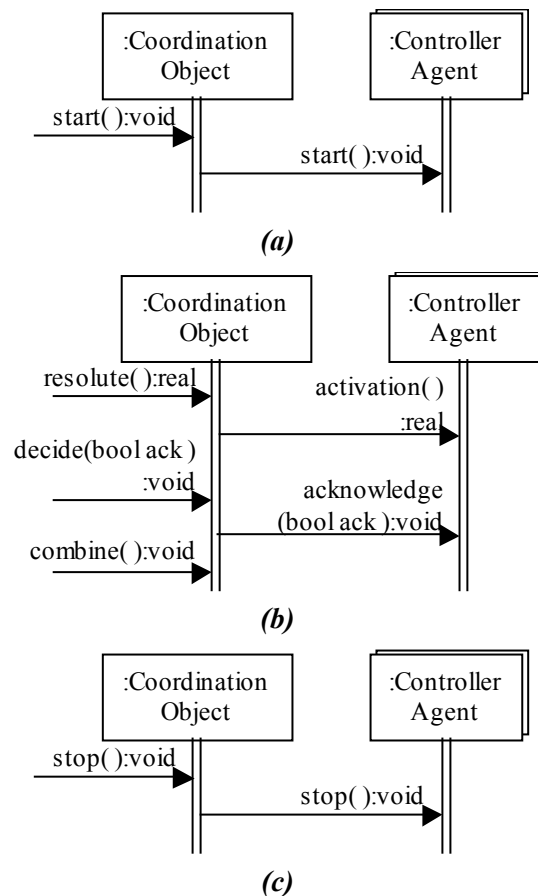


Figure 5-4: Sequence diagrams of Coordination Object

5.2.6 Composite Agent

Composite Agent is a kind of a Controller Agent. Composite Agent has a set of Controller Agents (sub agents) and a Coordination Object. Composite Agent conveys its event to its Coordination Object. Sequence diagrams of Composite Agent are illustrated in Figure 5-5.

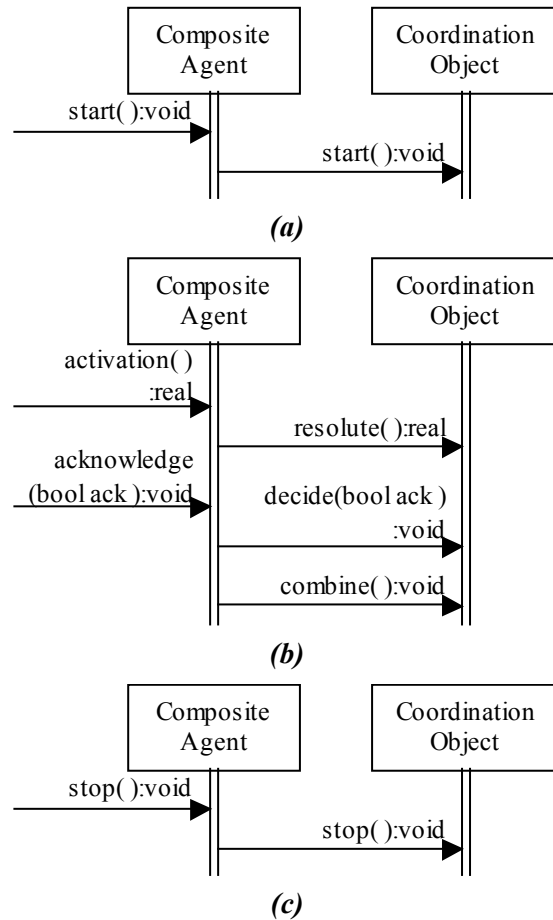


Figure 5-5: Sequence diagrams of Composite Agent

5.2.7 Elementary Agent

Elementary Agent is a basic agent. Elementary Agent is a kind of Controller Agent and has ‘start():void’, ‘stop():void’, ‘activation():real’, ‘acknowledge(bool ack):void’, ‘initialize():void’, ‘calculate():void’, ‘finalize():void’ and ‘update():void’ methods. All the methods except for the ‘acknowledge(bool ack):void’ method are specified in the specification of the Elementary Agent. The first two methods are responsible for the startup and shutdown sequence of the Elementary Agent. The activation method processes the activation request of the Elementary Agent and returns its preferred degree of activation. The degree of activation ranges from 0.0 to 1.0, 0.0 meaning the Agent does not want to be active and 1.0 meaning the Agent wants to be active.

The acknowledge of an activation request is processed by the ‘acknowledge(bool ack):void’ method. The argument of the method, ‘ack’ is the acknowledgement signal. This method triggers a sequence of events as illustrated in sequence diagram of Figure 5-7. Elementary Agents have two states, namely Active state and inActive state. As illustrated in the statechart of Figure 5-6, if the argument, ‘ack’, is true the Agent is activated (switched to Active state) else the Agent is deactivated (switched to inActive state). The

'initialize():void' method is invoked when an inActive agent becomes Active. This method contains code sequence for the transition of an agent from Active state to inActive state. The 'calculate():void' method is invoked if an agent is Active and it contains the code sequence of calculation of an Active agent. The 'finalize():void' method is invoked when an Active agent becomes inActive and it contains the code sequence for transition of an agent from inActive to Active state. The 'update():void' method is invoked irrespective to the state of the agent, thus contains code sequence for both Active and inActive states. This method usually updates the states of the agent.

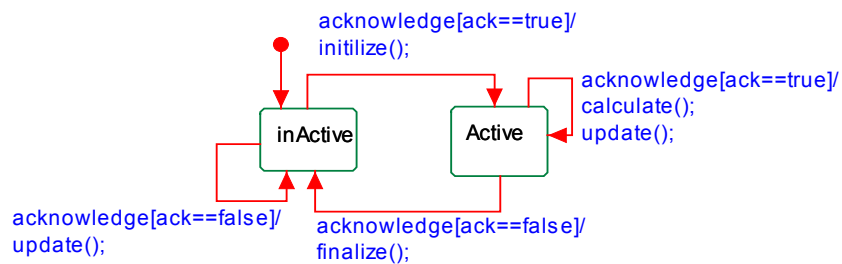


Figure 5-6: Statechart of Elementary Agent

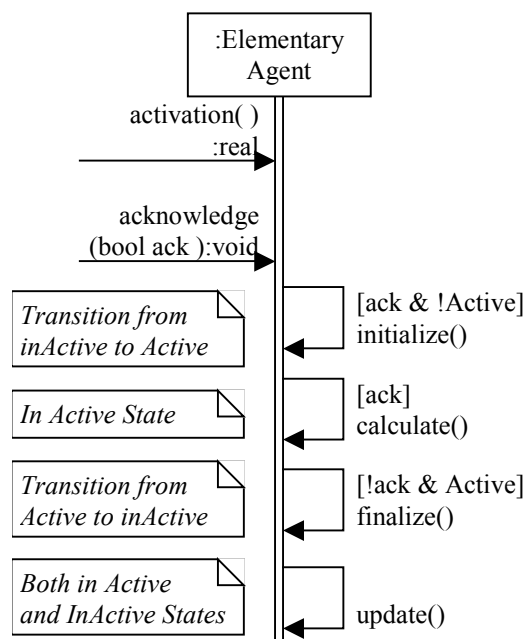


Figure 5-7: Sequence diagram of Elementary Agent

5.2.8 Realization of Multi-Agent Control Systems

Based on the operating principle described above an ANSI C++ library is developed. The library can be used with the Microsoft Visual C++ Compiler (for 20-Sim) and the GNU C++ Compiler (for 20-Works) to implement Multi-Agent Controller Systems. The specifications of Multi-Agent Controller System can be translated to C++ code, which inherits the operating principle captured in the library.

5.3 Keywords

5.3.1 Ports, Parameters and State Variables types

Multi-Agent Controller Systems implemented in the thesis currently support the port type of “real” (or equivalent C++ type “double”). For parameters and state variables, “real” (“double”), “boolean” (“bool”) and “int” are supported. The declaration of the variable type is included inside the <type> tag (see Section 4.3.3 and Section 4.3.4)

5.3.2 Sensor and Actuator types for 20-Sim

For the design of a Multi-Agent Controller System with 20-Sim, sensor readings and actuator signals are transferred to and from the system through 20-Sim. Thus, the type of sensor agent and the type actuator agent for 20-Sim implementations are included in the Multi-Agent Controller System framework. Keywords, “TwenteActuator” and “TwenteSensor”, are defined for the sensor agent and the actuator agent. Sensor and actuator types are declared in the <type> tag (see Section 4.3.3). The sensor agent has an output port, “output” and the actuator agent has an input port, “input”.

5.3.3 Operating State of Elementary Agents

As mentioned in Section 5.2.7, Elementary Agents have two operating states, “active” and “inactive”. The operating state of an Elementary Agent is specified by a boolean variable, “active”. The variable, “active”, can be used in the specification of the agent to acquire its operating state. Its value is “true” if the agent is in the “active” state and “false” if the agent is in the “inactive” state. The operating state of the agent is particularly useful in processing its activation request to determine its operating state.

5.3.4 Keywords of Coordination Objects

Coordination Objects are encapsulated in Composite Agents. The Coordination Object of a Composite Agent is associated with its sub agents and is responsible for coordinating them (see Section 5.2.5). The Coordination Object has information of its sub agents, which can be accessed through the defined keywords.

Some of the information of the sub agents is stored in a vector template class. In C++, a vector is a STL (Standard Template Library) container, which provides a form of dynamic array that can be expanded to accommodate additional elements. The dynamic property of vectors enables specification of generic Coordination Objects, which supports virtually any number of sub agents. The elements of a vector can be accessed randomly. For example, the i^{th} element of a vector, “v”, can be accessed by “v [i]”. The size of a vector is given by “size():int” function. The size of the vector, “v” is given by “v.size()”.

In a Coordination Object, the result of activation request of its sub agents, (“real:activation()”) is stored in a vector, “mu”. Acknowledgment signals to its sub agents are stored in another vector, “ack”. Another vector, “subAgentOutput”, gives first output of its sub agents. A method, “double getSubAgentOutput (int subAgentIndex, int outputIndex)” gives an output indicated by the output index, “outputIndex”, of a sub agent indicated by the sub agent index, “subAgentIndex”. Outputs of the Composite Agent are stored in a vector, “out”. An example of the default (when no coordination is specified) Coordination Object of Multi-Agent Controller Systems is presented in Table 5-2. In the default coordination, a Composite Agent wants to be active if all its sub agents want to be active. The acknowledge signal of the Composite Agent is conveyed to its sub agents. The Coordination Object is embedded in the library.

Table 5-2: Specification of the default Coordination Object

 Coordination Object: Default.xml
<pre> <?xml version="1.0"?> <!--Default.xml--> <cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance" xsi:noNamespaceSchemaLocation="CoordinationObjectSchema.xs d"> <name>MasterSlave</name> <implementation> <coordination> <resolute><![CDATA[{ double minMU=1.0; for (int i=0; i<mu.size(); i++) { if (mu[i]<minMU) minMU=mu[i]; } return minMU; /*wants to be active if all sub agents want to be active*/ }]]></resolute> <decide><![CDATA[{ for (int i=0; i<ack.size(); i++) ack[i]=acknowledge; /*convey acknowledge signal to all sub agents*/ }]]></decide> </coordination> </implementation> </cagentclass> </pre>

5.4 Design of Multi-Agent Controller Systems with 20-Sim

Multi-Agent Controller Systems is implemented in C++, which is linked with 20-Sim, for simulation purposes, through a static dll. In the 20-Sim environment, the simulator attaches the dll-file when a single run or a multiple run simulation is started and it detaches the dll-file when the simulation is stopped. 20-Sim recognizes four predefined functions (“Initialize():int”, “InitializeRun():int”, “TerminateRun():int”, “Terminate():int”) within a dll. The “Initialize():int” and the “Terminate():int” functions are called after attaching and before detaching the dll-file by the simulator. The “InitializeRun():int” and the “TerminateRun():int” functions are called before starting and after finishing every run of the simulation. The dll-file can also have user-defined functions, which can be called from 20-Sim during the simulation. An example of dll-function call is presented in Table 5-3. The 20-Sim function, “*dll*”, takes two string arguments and a vector of data type real. The first string arguments is the file name of the dll-file and the second argument is the name of the user-defined function, within the dll-file, to be called. The vector argument passes an input array to the user-defined function. The “*dll*” function outputs a vector, which gives an output array of the user-defined function.

An interface has been developed to use dll-function calls of 20-Sim. A dll-file is generated for a Multi-Agent Controller System, which interfaces the Multi-Agent Controller System with 20-Sim. The “Initialize():int” function instantiates the Main Agent (and Main Agent instantiates its sub agents and coordination object subsequently). The “Terminate():int” function destroys the instance of the Main Agent. The “InitializeRun():int” and the “TerminateRun():int” runs “start():void” and “stop():void” events of Main Agent. Apart from the four predefined functions, the dll-file also contains an additional function, “mac”. The “mac” function sets the sensor values of the Multi-Agent Controller System, calls the “tick():void” method of Main Agent and gets the actuator values of the Multi-Agent Controller System to 20-Sim. The sequence diagram of the interface is illustrated in Figure 5-8.

The first four functions are called automatically by 20-Sim. This function is called by the 20-Sim function, “*dll*”. An example of 20-Sim submodel, which interfaces with a Multi-Agent Controller System, is presented in Table 5-3. As illustrated in the table, the first argument specifies the file name of the dll-file and the second specifies the function to be called, which is the “mac” function. In the example, the “dllin” is a vector (of size 1x1), which gives the sensor signals (x_motor) to the Multi-Agent Controller System. The “dll” function outputs another vector, “dllout”, which consists of the actuator signal (voltage). As the “mac” calls the “tick():void” function of the Multi-Agent Controller System, it should be called at a fixed time interval (in terms of simulation time).

Thus, the dll function call should be implemented in a discrete system of 20-Sim.

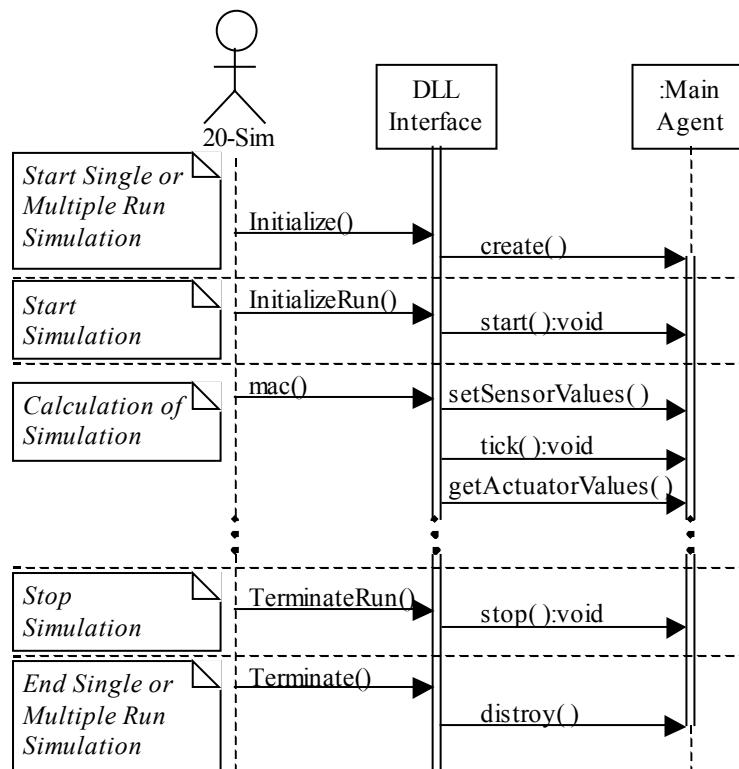


Figure 5-8: Sequence Diagram of interface to 20-Sim with Multi-Agent Controller System

Table 5-3: Example of a 20-Sim submodel for interfacing Multi-Agent Controller System with a dll-function call

```

MacInterfaceModel.em
equations
  dllin = [x_motor];
  dllout = dll('DemoLin.dll', 'mac', dllin);
  voltage = dllout[1];
  
```

5.5 Implementation of Multi-Agent Controller Systems with 20-Works

The Multi-Agent Controller System framework defines three events, the start, tick and stop events that have to be invoked by an external interface. The start event and the stop event are triggered while starting and stopping the Multi-Agent System. The tick event is triggered at each sampling event. 20-Works provides the interface, which calls the tick event at a fixed time interval.

20-Works is software developed at Control Laboratory, University of Twente to support application of control systems. It is a collection of classes, which can be used to implement controllers. 20-Works has a user interface based on Borland

Turbo Vision. It also provides a timer, which delivers the sampling events and a graphical screen, which contains “views” for online plotting of its variables. 20-Works also provides sets of sensors, actuators, controllers, generators and storers. 20-Works executes two loops: a synchronous loop and an asynchronous loop. Sensors, actuators, controllers and generators are executed in the synchronous loop, which is synchronized by the timer and storers and views are executed in the asynchronous loop.

For Multi-Agent Controller Systems, the timer and the views of 20-Works is used to get the timer events and to plot the variables of Multi-Agent Controller Systems. The timer event is used to call the tick function of Multi-Agent Controller Systems. The adopted implementation of 20-Works and Multi-Agent Controller Systems is described in Figure 5-9. The start event and stop event of Multi-Agent Controller Systems are executed before starting and after ending the loops.

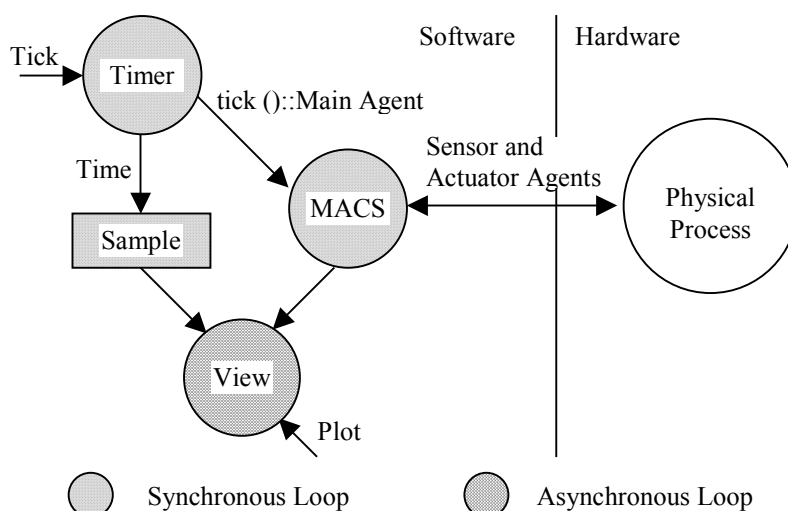


Figure 5-9: Functional Model of 20-Works implemented in Multi-Agent Controller Systems

5.6 Conclusion

A brief description of operating principle of Multi-Agent Controller Systems is presented in this section. The operating principle is captured in a C++ library. A framework is presented for implementing Multi-Agent Controller Systems in 20-Sim and 20-Works.

The developed library is embedded in the tool, IDITmac. Automated code generation of the agents is implemented in IDITmac (see Appendix C.2). Based on the framework for 20-Sim, automated generation of dll-files is also implemented in IDITmac. An integrated approach of design and implementation of agents is realized by utilizing the same library and the same codes for both the cases.

CHAPTER 6

CASE STUDY

6.1 Introduction

A case study of a Multi-Agent Controller System is carried out in this thesis. In the case study, the concept of agents is applied in the simulation and in the real system. A brief description of the system to be controlled and the controller system designed is presented in this section. Simulation results and results of the implementation is also presented here. The main aim of the case study is to demonstrate the ability of the tool designed in this thesis for integrated design and implementation of control systems.

6.2 Demonstration Setup (DemoLin)

The developed tool is tested in a demonstration setup, DemoLin. The setup is a mass-spring-mass system, developed at Imotec BV for demonstration purposes. A schematic diagram of DemoLin is presented in Figure 6-1 and a detail diagram is presented in Appendix D. The system has a base plate (motor mass), which is driven by a linear motor (Yaskawa SGLGW 60A365A). Another mass (end effectors mass) is connected on the top of the base plate with two flexible iron plates. Both the masses are attached to pretension belts. Both the belts are supported by pulleys mounted on two shafts. In the left shaft, the pulley of the lower belt is fixed whereas the pulley of the upper belt is connected with bearings. In the right shaft, the pulley of the upper belt is fixed and the pulley of the lower belt is connected with bearings.

For the control purpose, position measurement of the motor mass is taken as a feedback signal. Since the index pulse of the linear encoder is not accessible, the calibration of the linear encoder is done with the index pulse of the rotational encoder (encoder of the end effectors mass). Within the full stroke movement of the masses, the shafts make more than one revolution thus two pulses are generated by the index signal of the rotational encoder. One of the pulses is encountered when the left edges of the masses are aligned with the left

shaft and another pulse is encountered at 70 mm from the left end stop. The position of the first index pulse is taken as 0 mm position.

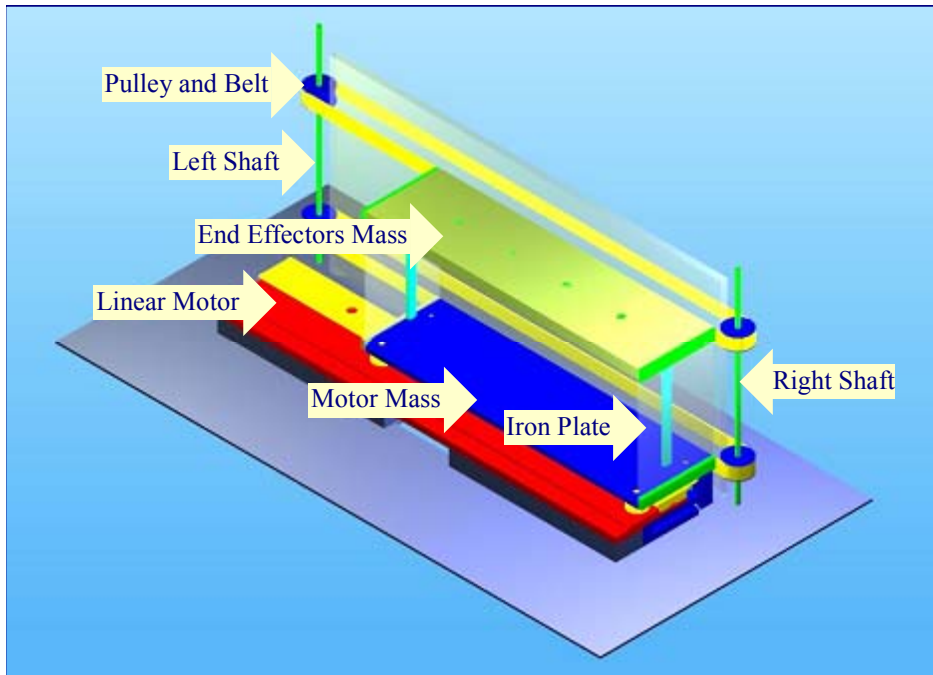


Figure 6-1: Schematic Diagram of DemoLin

6.3 Control Problem

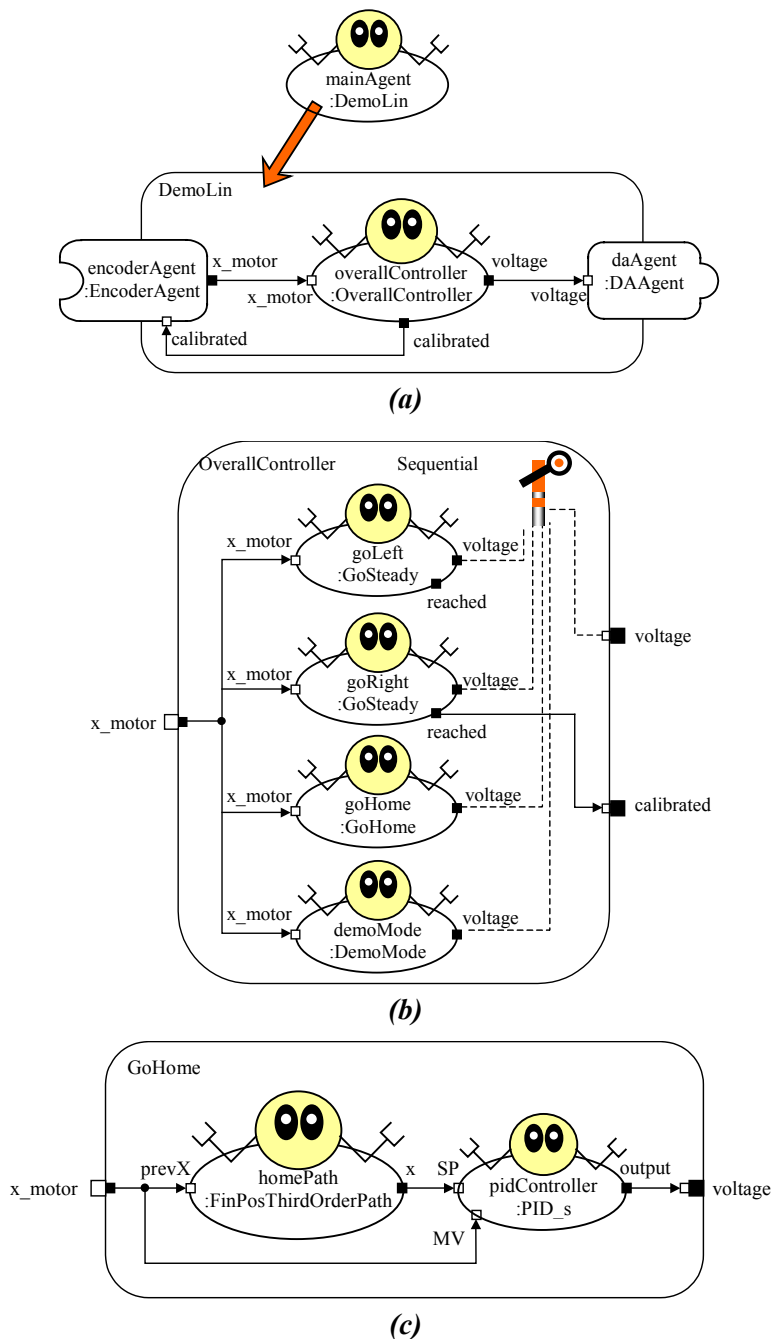
DemoLin is a demonstration setup, designed for demonstration of controller performances at Imotec BV. A basic controller should be implemented in DemoLin to demonstrate the application of Multi-Agent Controller System. The criteria of the controller are stated as follows.

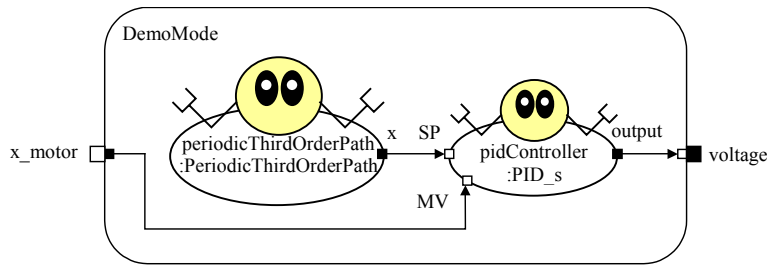
- ❑ Calibration: Since the encoders in DemoLin are incremental encoders, the encoders should be calibrated.
- ❑ Homing: After calibrating, the masses should be positioned in a home position (at $x_{\text{motor}} = 0$ mm).
- ❑ Motion: DemoLin should make repetitive strokes of + 115 mm (from the home position) and -115 mm (towards the home position).
- ❑ Safety: Safety should be implemented in the system to prevent unstable and oscillating motions of the masses, which may occur due to disturbances.

6.4 Control Strategy

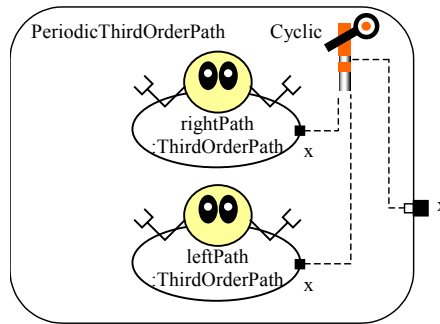
The controller is implemented in a Personal Computer. The position of the masses is fed through an encoder interface card. The motor is actuated by a voltage reference from the DA card (Digital to Analog Card) of the computer. A Multi-Agent Controller System is designed and implemented in DemoLin. The design of the controller is tested in 20-Sim with the dll-file generated by

IDITmac. The implementation of the designed controller is accomplished in 20-Works with the code generated for the Multi-Agent Controller System by IDITmac. The organizational diagram of the designed controller is presented in Figure 6-2 and the MacsML specification of the controller is presented in Appendix F.





(d)



(e)

Figure 6-2: Organizational diagram of Multi-Agent Controller System for DemoLin

6.4.1 Sensor and Actuator Agents

As illustrated in Figure 6-2(a), the control system has a “mainAgent” of class (or type) “DemoLin”. The “DemoLin” agent has a sensor agent, “encoderAgent:EncoderAgent”, and an actuator agent, “daAgent:DAAgent”. The “DAAgent” provides a voltage output (“voltage”) to the DA-Card. The “EncoderAgent” gives position of the motor mass (“x_motor”) and has an input (“calibrated”). The encoder has to be calibrated for absolute position readings. The input, “calibrated”, indicates the calibration status of the sensor. The input value of less than 0.5 signifies that the encoder is not calibrated, so the reading of the linear encoder is reset when the index pulse of the rotational encoder is encountered. In other cases, the linear encoder is not reset. During the startup, the input, “calibrated”, is set to 0.0, as the encoder is not calibrated. The input is set by another agent after a calibrating motion is completed, which is explained the next section.

6.4.2 Motions of DemoLin

DemoLin performs three motions in a sequential order. As the encoders are not calibrated, DemoLin firstly performs a calibrating motion during which the linear encoder is calibrated. After calibration, it performs a homing motion, which positions the masses at the home position ($x_{\text{motor}} = 0 \text{ mm}$). After the masses are homed properly, a demo motion is performed, where the setup makes a repetitive stroke of +115 mm and -115 mm. The motions performed are described as follows.

Calibrating motion

The first task of the controller is to calibrate the encoder. The encoder is initially set in the calibrating mode and the linear encoder reading is reset whenever the index pulse of the rotational encoder is encountered. As mentioned in Section 6.2, the index pulses occur twice during the full stroke motion of the setup. To ensure the encoder is reset correctly, the correct index pulse (when the left edges of masses align with the left shaft) should be encountered at the end of the calibrating motion. Thus, DemoLin first performs a left to right motion until it is stopped by the right end stop to ensure it is in right side of the correct index pulse. Then it moves from right to left direction until it is stopped by the left end stop. At the end of calibrating motion, the masses encounter the correct index pulse, thus, the encoder is reset correctly. Thus, the “calibrated” signal is sent to the encoder after the calibrating motion is completed.

The calibrating motion is performed by “goRight:GoSteady” and “goLeft:GoSteady” agents of “OverallController” agent. Both the agents are instances of “GoSteady” agent class. The “GoSteady” is an elementary agent, which implements a proportional velocity control, which can be summarized by following equation.

$$V = (\Delta x_{ref} - \Delta x) \cdot K$$

where,

V is voltage to the DA card,

Δx_{ref} is reference incremental change in position,

Δx is measured incremental change in position,

and K is gain factor of the controller.

The “GoSteady” agent deactivates when the masses are stopped (by the end stops), and gives a signal, “reached”. The output, “reached”, of the “goLeft” agent is connected to the “calibrated” input of the “encoderAgent”, thus the sensor switches to non-calibrating mode as soon as the “goLeft” agent is deactivated i.e. the calibrating move is completed.

Homing Motion

Homing motion positions the masses at the home position ($x_{motor} = 0$ mm), which is accomplished by “goHome:GoHome” agent. The “GoHome” agent is a composite agent, which contains a PID-controller agent, “pidController:PID_s_safety” and a third order path generator agent, “homePath:FinPosThirdOrderPath”. The “FinPosThirdOrderPath” generates a path from the current position of the masses (given by the input “prevX” before activating) to a final position, specified in a parameter of the agent, “Hfin”.

Demo Motion

Demo Motion performs a repetitive, back and forth motion of 115 mm stroke length from the home position and is implemented in “demoMode:DemoMode” agent. The “DemoMode” agent has a periodic third order path generator “periodicThirdOrderPath:PeriodicThirdOrderPath” and a PID-controller, “pidController:PID_s_safety”. The “PeriodicThirdOrderPath” agent generates a cyclic path and is implemented with a left to right stroke generator, “rightPath:ThirdOrderPath” and a right to left stroke generator, “leftPath:ThirdOrderPath”. The “ThirdOrderPath” agent generates a third order path from an initial position, specified in a parameter of the agent, “Ho”, and with a stroke length, specified in a parameter of the agent, “Hm”.

6.4.3 PID-Controller and Safety

The implemented PID-controller, “PID_s_safety” is a series form controller. The implemented controller is based on a 20-Sim PID-Controller implementation, “Controller-PID_s” and safety is added to the implementation. The controller deactivates as soon as the error signal exceeds a certain maximum value (specified in a parameter of the agent, “maxError”). Once the controller is deactivated due to safety reason, it cannot be reactivated. The design of the PID-Controller is presented in Appendix E.

6.5 Simulation Result

6.5.1 Model of DemoLin

The model used for simulation of DemoLin is a mass spring mass model as shown in Figure 6-3. A viscous friction model is implemented in the model. A detailed description of the model is presented in Appendix D.

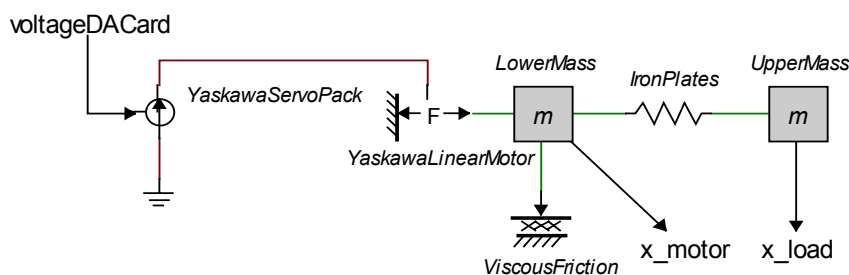
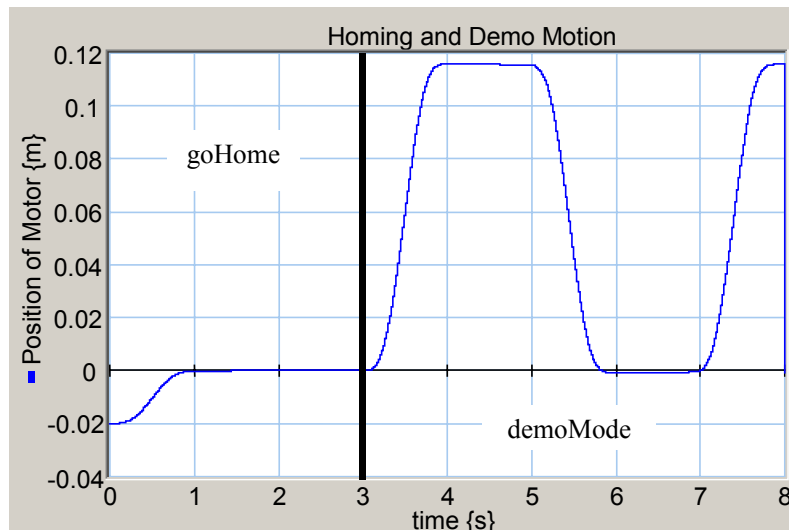
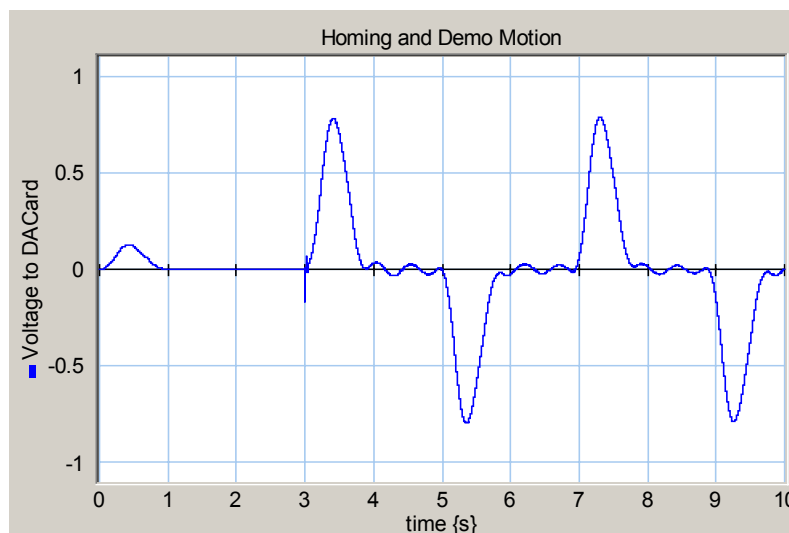


Figure 6-3: Model of DemoLin

Calibration of the encoder is not implemented in the model thus the calibrating motion is not simulated. The Homing Motion and Demo Motion are simulated with the dll-File generated for the designed Multi-Agent Controller System. In the model, the position of the motor mass is initialized to -0.2 mm. DemoLin performs Homing Motion and Demo Motion as shown in Figure 6-4.



(a)



(b)

Figure 6-4: Simulation result of Homing and Demo Motions

6.6 Implementation Result

The designed control system is implemented in a real system with 20-Works. Results of the implementation are presented in the figures below. The Calibrating Motion and the Demo Motion are presented in Figure 6-5. The Demo Motion is presented in Figure 6-6 and Figure 6-7. A case of Safety is also illustrated in Figure 6-8, in which a disturbance (manual obstruction of the masses) is introduced which result in the tracking error greater than the maximum limit and the PID-Controller is deactivated.

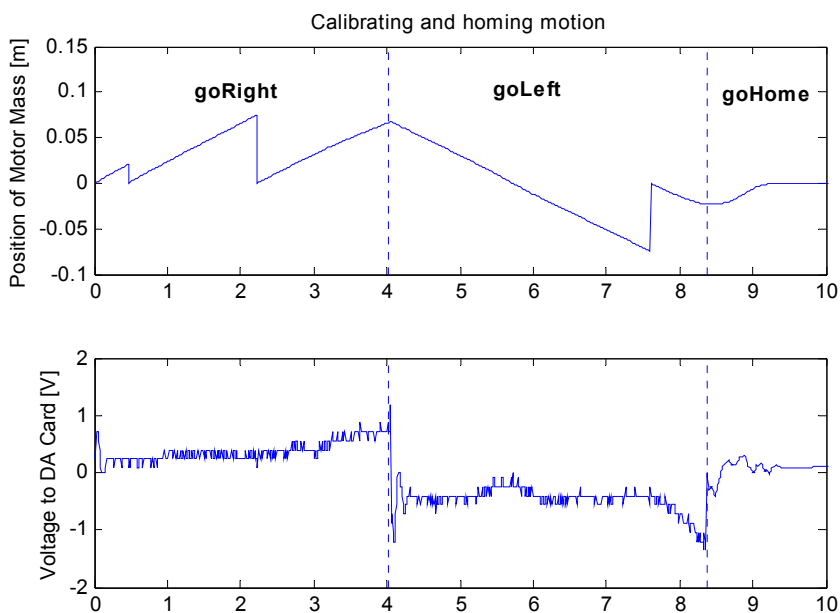


Figure 6-5: Plots of Calibration and Homing Motions

In the calibrating motion, the Motor position reading is reset whenever the index pulse of the rotational encoder is encountered.

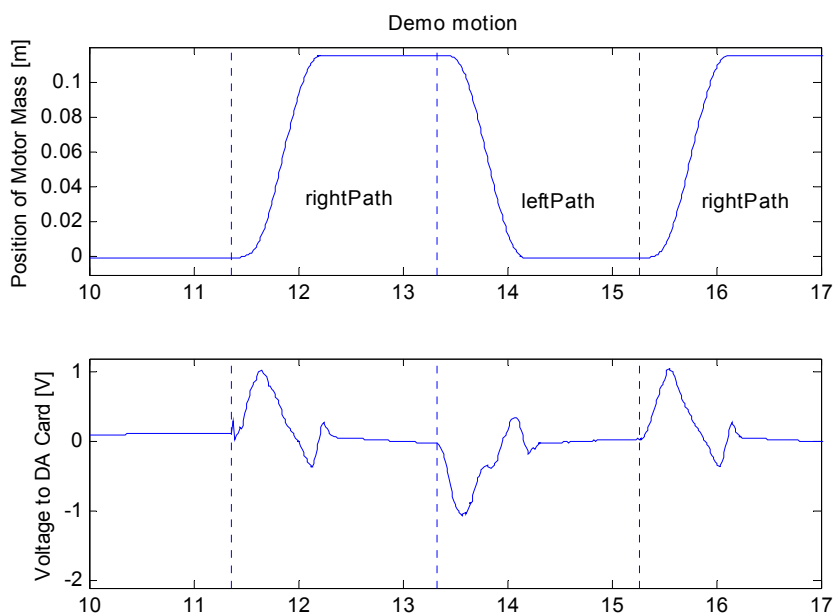


Figure 6-6: Plot of Motor Mass position and Voltage in Demo Motion

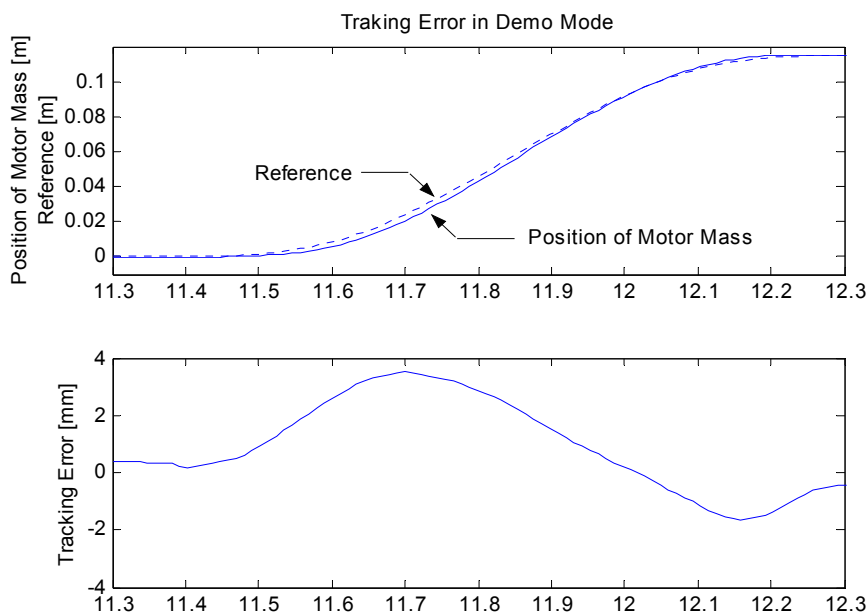


Figure 6-7: Tracking error in Demo Motion

In Demo Mode, the error between the reference path and the measurement is within ± 4 mm and steady state error is within ± 0.4 mm.

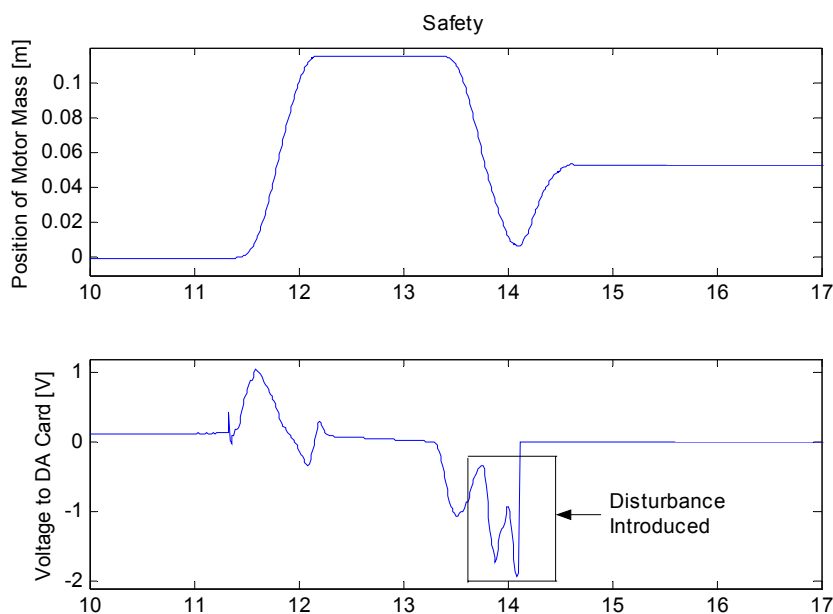


Figure 6-8: Activation of Safety when error exceeds maximum error limit

As observed in the figure above, the controller attempts to overcome the disturbance and as the error exceeds the safety limit, the controller is deactivated. The motion of the motor mass after the safety is activated is because of the disturbance.

6.7 Conclusion

A Multi-Agent Controller System is successfully designed and implemented on DemoLin with the tool developed in this thesis. The ability of agents to organize the control solutions is demonstrated in the case study. An integrated approach of design and implementation is pursued by simulating the designed Multi-Agent Controller System in 20-Sim and implementing the simulated design with 20-Works.

In addition, some aspects of the plant, such as calibration of sensors and interface with DA Card and Encoder Interface Card, which are not included in their models, can be easily added in the implementation. Similarly, an implemented Multi-Agent Controller System can be easily extended to support additional functionality.

CHAPTER 7

CONCLUSION AND RECOMMENDATION

7.1 Conclusion

This project is evolved from the MACSIF (Multi-Agent Controller Systems Implementation Framework) and the MACSL (Multi-Agent Controller Specification Language) developed by van Breemen [23]. The achievements of this thesis are illustrated in Figure 7-1. The figure is extension of the proposed structure in Figure 2-3. The achievements are summarized as follows.

Structured and competent specification of Multi-Agent Controller Systems

A XML based specification, MacsML, has been designed for Multi-Agent Controller Systems. The MacsML provides a structural aspect to specifications of agents. MacsML is extensible. Support tools for MacsML can be easily designed, as various tools are available for XML.

Platform Independent Implementation of Multi-Agent Controller Systems

An ANSI C++ based library of classes has been developed in the thesis, which can be used in various platforms. The designed library has been successfully tested in a Microsoft Compiler and a GNU compiler.

Fully functional tool for Integrated Design and Implementation

The designed tool for Integrated Design and Implementation, IDITmac, provides a fully functional tool for design and implementation of Multi-Agent Controller Systems.

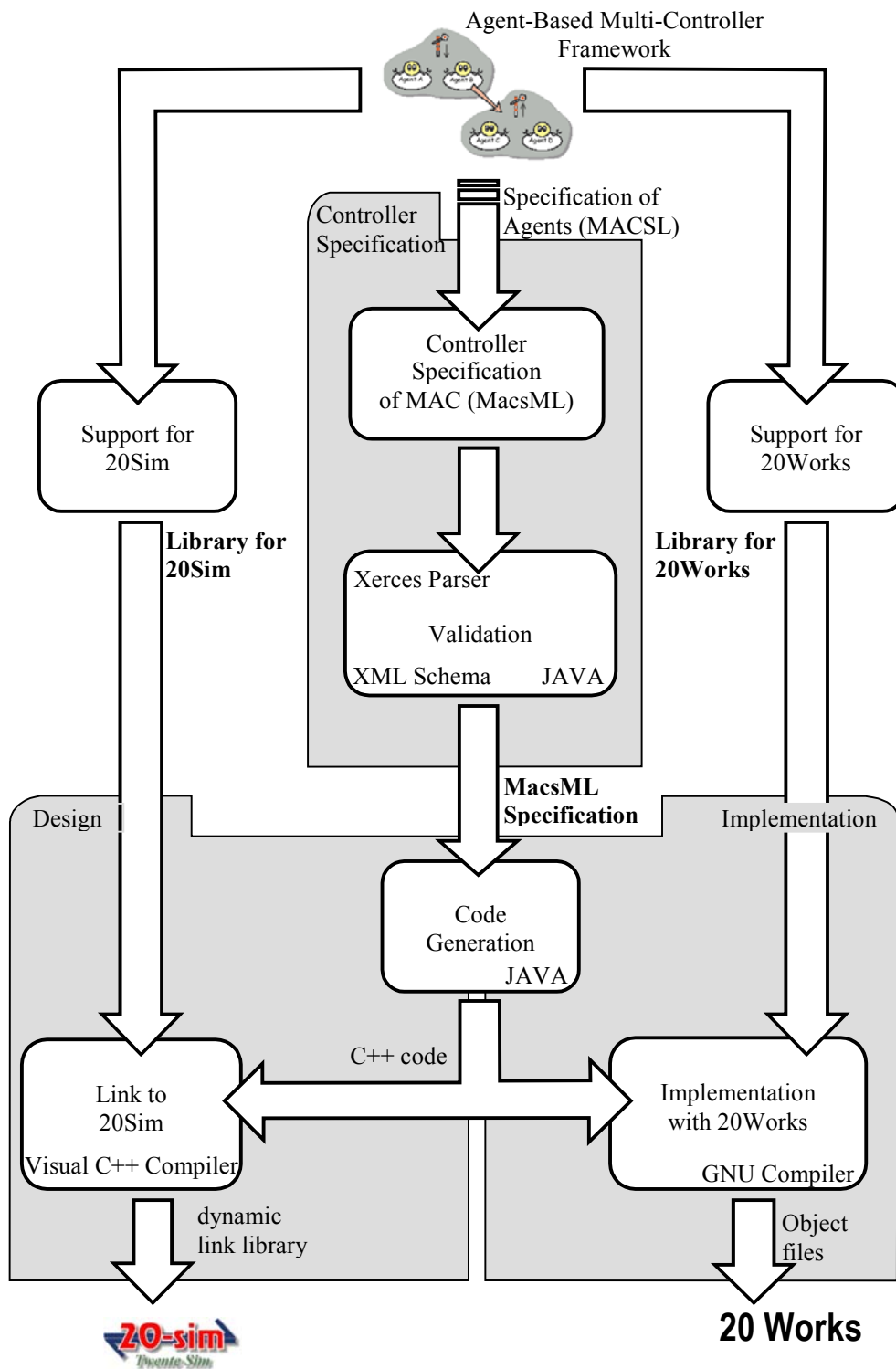


Figure 7-1: Developed tool for Integrated Multi-Agent Controller Systems

Demonstration of Multi-Agent Controller Systems

The ability of Multi-Agent Controller Systems to solve complex problems and organize the solution has been demonstrated in the case study, DemoLin. In Multi-Agent Controller Systems, a stepwise refinement process of controller design and implementation can be achieved through addition, removal and improvements of agents without affecting the rest of the design.

7.2 Recommendation

The tool developed in this thesis provides basic facilities for integrated design and implementation of Multi-Agent Controller Systems. The desirable improvements to the tools are presented as follows.

Probe for agents: Major constraint of the tool is lack of debugging facilities. A facility to monitor ports and states is desirable for debugging the agents in a controller system. A name field is provided in Agents, which can be used for this purpose.

Port type: Currently, ports of data type “real” (“double”) are implemented in this thesis. The data type should be extended to include more data types such as integer, boolean etc.

Mac Works: Multi-Agent Controller Systems is currently implemented with 20-Works which is designed for a different environment, thus many of the functionalities of 20-Works is not useful. 20-Works should be extended to support the agent environment.

Mac enabled 20-Sim: Interfacing of Multi-Agents Controller Systems with 20-Sim through dll-files provides a good solution for the integrated approach. However, 20-Sim should provide specialized functions for calling of start, tick and stop functions and transferring sensor and actuator data.

Off-the-shelf agents: A library of standard Agents such as PID Controller agents, Path Generators, Neural and Fuzzy Agents should be designed. Moreover, a wider range of Coordination Objects should also be designed, as Coordination Objects are the most critical and reusable components of Multi-Agent Controller Systems.

IDITmacPlus: A graphical specification of Multi-Agents Controller Systems should be introduced and tools should be provided which translate the graphical specification to MacsML specification. A DOM parser is recommended for this purpose, which is more powerful than the currently used SAX Parser. A DOS and Linux based tool is desirable as implementation of agents is mostly accomplished in these environments.

APPENDIX A

XML AND XML SCHEMA

A.1 Background of XML (Extensive Markup Language)

In 1969, IBM introduced Generalized Mark-up Language (GML) as a means of allowing the text editing, formatting and information retrieval subsystems to share documents. GML evolved as Standard Generalized Mark-up Language (SGML). In 1986, SGML was established as an ISO standard, ISO 8879. A simplified application of SGML was developed for rendering of documents in the web, which is now known as Hyper Text Markup Language (HTML). HTML has a fixed set of tags.

The World Wide Web Consortium (W3C) combined the power of SGML with the simplicity of HTML and came up with Extensive Markup Language (XML). XML is a subset of SGML. W3C released recommendation of XML 1.0, in February 1999 and recommendation of XML 1.0 (Second Edition), in October 2000 [35].

A.2 XML (Extensive Markup Language)

Markup Language: Markup gives meaning to a document. An example of Markup is highlighting of text in a document. A Markup Language is just a set of rules defining Markup structure. In XML, anything in angle brackets (<...>) is considered Markup (also referred as tag).

Like its predecessor, SGML, XML is a meta language. Unlike HTML, XML doesn't have predefined tags and tags can be invented according to the need of the XML application. Thus, XML can be used to describe, store and process virtually any kind of data.

Tags: XML documents are text documents. An example of XML document is illustrated in Table A.1 Each opening tag, such as <name>, in an XML document has a closing tag, such as </name>. All XML documents should have a root element (also referred as start tag and end tag), in the example <cagentclass> </cagentclass> is the root element of the XML document. The elements (tags) of XML document should be properly nested. For example, <name>, <interface> and <implementation> elements are nested inside <cagentclass> tag.

Comments: In XML documents, comments are enclosed in `<!-- -->`. Comments may appear anywhere in a document outside other markup and anything inside the comment block is ignored by XML processors (parsers).

Processing Instruction (PI): Processing instructions (PI) begin with `<?>` and end with `?>`. The processing instructions are information for the application. The PI begins with the PITarget used to identify the application to which the instruction is directed. In the example, the XML declaration `<?xml version="1.0"?>` is a PI, `xml` is the PITarget and `version="1.0"` is version declaration.

CDATA: CDATA Sections (`<![CDATA ...]]>`) are used to escape blocks of text containing characters, which would otherwise be recognized as markup. XML processor treats them as a character data. CDATA blocks are used to include special characters as character data, as inside the `<calculate>` tag in the example.

XML has become quite popular for data storing thus various parsers and tools are available for XML parsing. The tags of XML document are not predefined by the W3C recommendation and different tags can be defined according to the requirement of the application. XML documents can be checked if the documents consist of the tags being defined and if the structure of the tags is according to the defined syntax. XML Schema, DTD (Document Type Definition) are used to define the tags of XML documents. Description of XML Schema is presented in next section.

A.3 W3C XML Schema

XML Schema was originally proposed by Microsoft, but it became an official W3C recommendation in May 2001 [36] [37] [38]. An XML Schema establishes a set of rules for constraining the structure and articulating the information set of XML document instances. XML Schema is an XML document itself. XML Schema defines legal building block of an XML document. XML Schema can be customized to support any syntax. The designed XML Schema can be used to check XML documents against the given syntax. XML documents which validating against XML Schema is called validating XML document.

Table A.1: An Example of Multi-Agent Controller Specification Markup Language (specification of a P-Controller)

PController.xml
<pre> <?xml version="1.0"?> <!--A Comment which is ignored by parser--> <cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="CagentSchema.xsd"> <name>PController</name> <interface> <ports> <input> <type>real</type> <name>reference</name> </input> <input> <type>real</type> <name>measurement</name> </input> <output> <type>real</type> <name>control_signal</name> </output> </ports> <parameterdefs> <parameterdef> <type>real</type> <name>kp</name> <defaultvalue>1</defaultvalue> </parameterdef> </parameterdefs> </interface> <implementation> <elementary> <calculate> <![CDATA[{ control_signal=kp*(reference-measurement); }]]> </calculate> </elementary> </implementation> </cagentclass> </pre>

The example presented in Table A.2 (a part of the xml document presented in Table A.1) assigns a Schema, “CagentSchema.xsd”, to the XML document. The structure of the document is validated against the Schema, “CagentSchema.xsd”.

Table A.2: Declaration of Schema

PController.xml
<pre> <cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="CagentSchema.xsd"> </pre>

APPENDIX B

UML

UML can be used in modeling software projects. UML defines twelve types of diagrams, organized in three categories. The diagrams used in the thesis are in the following sections.

B.1 Structural Diagrams

Structural diagrams are used to model the static structure of a system. It shows the static relationship between classes. Class Diagrams are used to describe the structure of classes within a system.

B.1.1 Class Diagram

A class is denoted by a box with the class name in bold at the top (Figure B.1). The attributes of the class appear below the class name. The key operations of the class appear below the attributes.

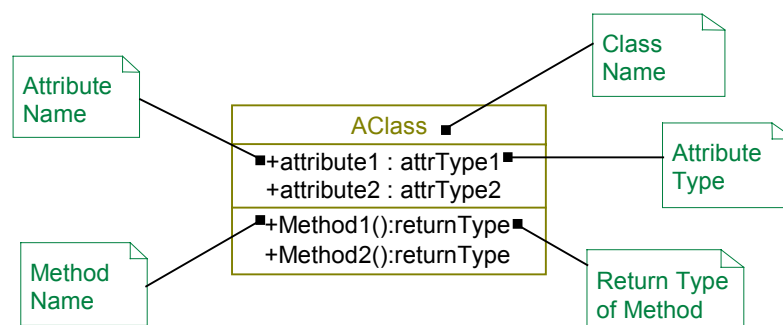


Figure B.1: A class notion

Association

Associations represent relationship between instances of classes. A line connecting both classes represents their association (Figure B.2). The multiplicity of a relationship is represented by numbers, which are printed next to each end of the association. An asterisk (*) indicates a many (zero or more) multiplicity. The example presented in Figure B.2 shows association between an employer and its employees.

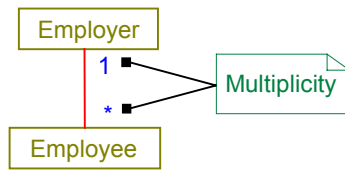


Figure B.2: Example of associations

Composition

Composition is a whole/part relationship in which a class (the whole) contains another class (the part) and creation and destruction of the part is responsibility of the whole. It shows ownerships of classes and represented by a line with a filled diamond shaped end. As shown in Figure B.3, the diamond shaped end of the relationship is in the owner's (the whole's) side.

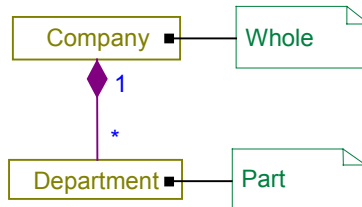


Figure B.3: Example of composition

Inheritance

Inheritance (also known as generalization) is used for modeling a kind-of-relations. It is represented by a line ending with a triangle. As illustrated in Figure B.4, the triangular end is towards the parent class.

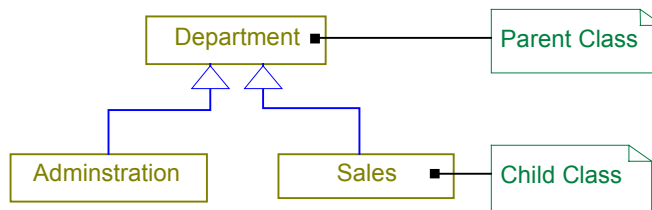


Figure B.4: Example of Inheritance

B.2 Behavior Diagrams

Behavior diagrams are used to model the dynamic behavior of the system, for example state changes within an object. Two kinds of Behavior diagrams is described in the following sections.

B.2.1 Sequence Diagram

Sequence diagrams describe how groups of objects collaborate in some behavior over time. It shows the messages that are passed between objects. In

sequence diagrams, objects are represented in rectangular boxes as illustrated in Figure B.5.

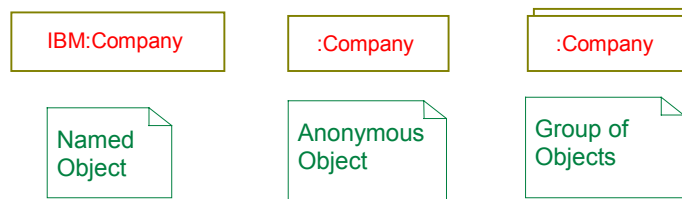


Figure B.5: Object representation in Sequence Diagrams

Each object has a ‘time line’ showing its creation and destruction. Messages of objects are represented as horizontal arrow and conditions are enclosed in square brackets. Figure B.6 shows an example the process of support request by a customer. In the example, the customer is not a part of the company (system) thus represented as an Actor.

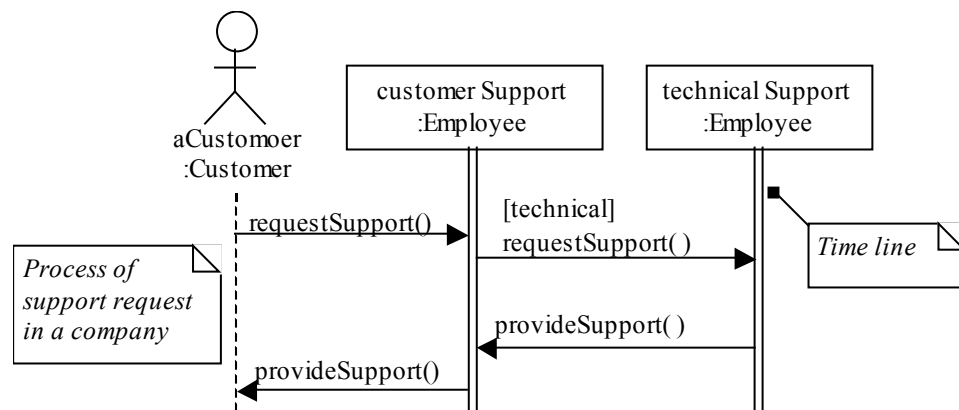


Figure B.6: Example of Sequence Diagram

B.2.2 Statechart Diagram

Statechart diagrams show the possible states of an object and the transitions that cause a change in state. States are represented in rounded rectangles and transitions are represented by arrows from one state to another. Events or conditions that trigger transitions are included beside the arrows. The action that occurs as a result of an event or a condition is expressed in the form, “/action”. An initial pseudostate is represented by a black circle. An example of a statechart of lifecycle of employees in a company is presented in Figure B.7.

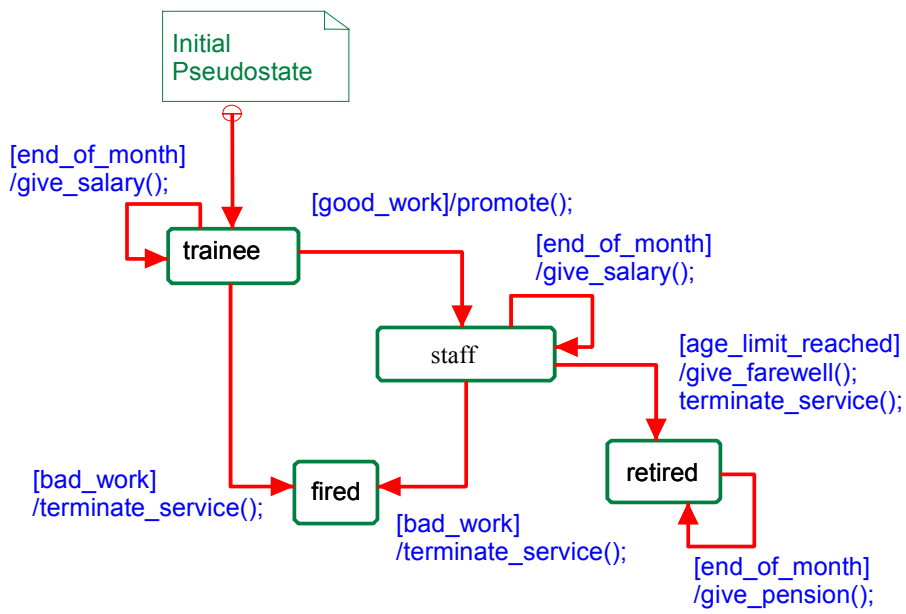


Figure B.7: Example of statechart

APPENDIX C

FEATURES OF IDITMAC

A tool, IDITmac (Integrated Design and Implementation of multi-agent controllers), is designed in this thesis to facilitate design and implementation of Multi-Agent Controller Systems. The main functionalities of the tool can be summarized as follows.

- ❑ Creating and editing of MacsML specifications
- ❑ Checking of MacsML specifications
- ❑ Code Generation of ANSI C++ code from MacsML specifications
- ❑ Generation of dll file for Multi-Agent Controller Systems from their MacsML specifications

IDITmac has a Graphical User Interface, which is developed in Java Swing (Figure C.1). As shown in the figure, the interface has four display panels. The “XML File”, the “Header File” and the “CPP File” panels is used to display the MacsML specification, the generated header file and the generated C++ file of an agent. The “Output” panel gives status of the tool.

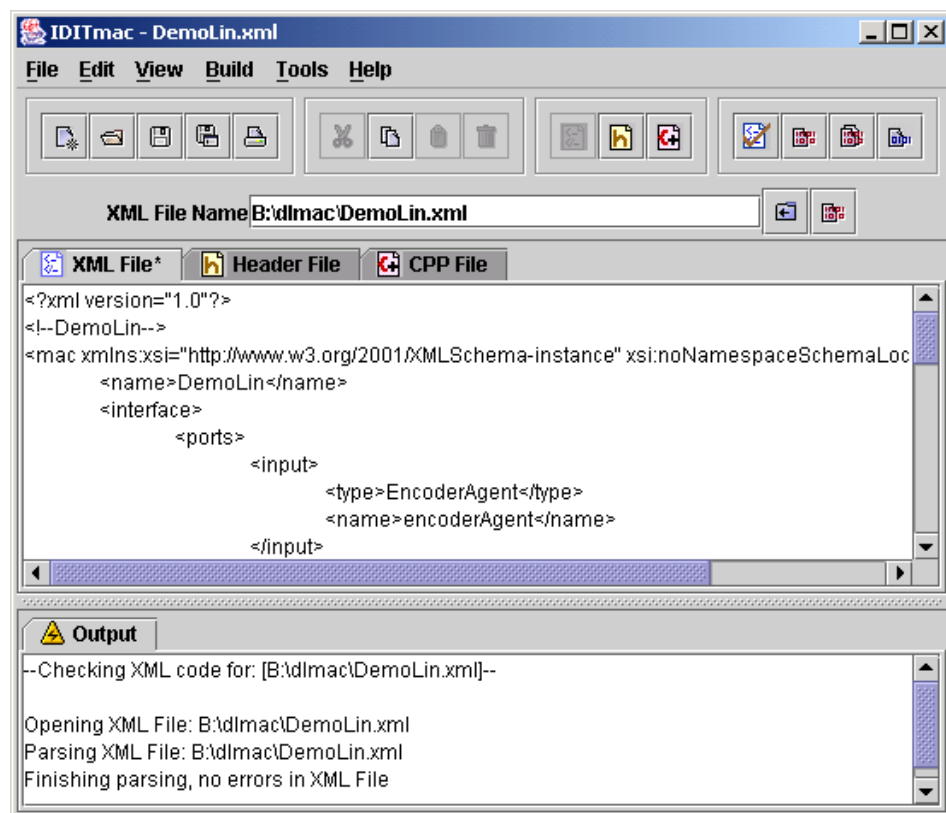


Figure C.1: Snapshot of Graphical User Interface of IDITmac

IDITmac is developed in Java and it utilizes the Xerces 2 Java Parser [33] for extracting information from MacsML specifications. The extracted information of an agent is stored in a collection of Java objects and translated to an ANSI C++ code. The tool requires the Microsoft Visual C++ Compiler for the generation of dll-file (for 20-Sim).

C.1 Support for MacsML in IDITmac

The developed tool, IDITmac, also supports complete editing and checking of MacsML specification files. The editing functionality is similar to a general text editor. MacsML specifications are checked against corresponding Schemas and the structural error in a MacsML document is reported in the output panel of the tool. IDITmac facilitates checking integrity of a document while editing. The prominent features of IDITmac regarding MacsML are as follows.

Fully functional text editor

IDITmac is a fully functional text editor especially designed for MacsML. Common features of a text editor such as opening existing MacsML files, saving files (MacsML and generated codes), printing files are included in IDITmac. Editing facilities such as cut, copy and paste are supported in the tool. Other useful functions such as an indicator for unsaved files and warning of loss unsaved data while closing or processing of a file are also added in the tool. Text editing functions are under **F**ile and **E**dit menus.

Templates for Agents

IDITmac also provides templates of MacsML documents for all the six basic types agents of Multi-Agent Controller Systems. Templates are convenient starting tools for specifying agents, as the user can easily customize the templates to build desired agents. Templates can be accessed by **F**ile+**N**ew **T**emplate menu.

Checking of XML specifications

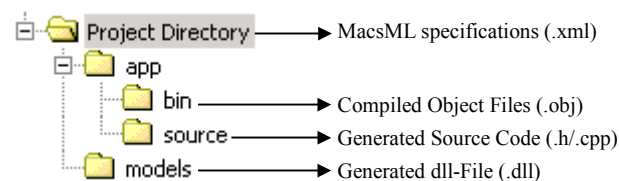
IDITmac also supports checking of MacsML specifications of the Multi-Agent Controller Systems, allowing the user to check the specifications while editing. Feedback of structural errors in a MacsML document is reported to the user by an error prompt and the description of the error is presented in the output panel. Checking of a MacsML document can be done via the **B**uild+**C**heck XML menu.

C.2 Support for Design and Implementation in IDITmac

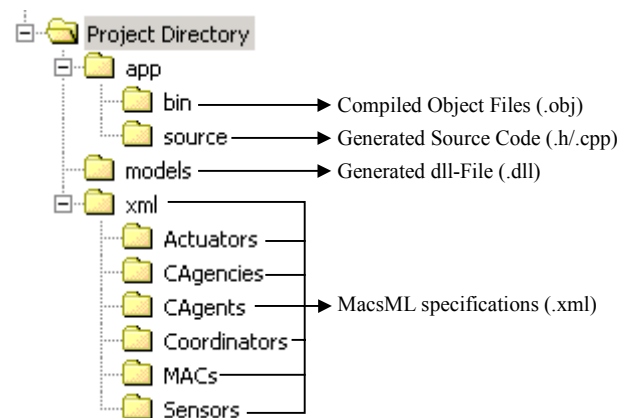
The architecture and the operating principle of Multi-Agent Controller Systems is captured in an ANSI C++ library of classes, which can be used in both the Microsoft Visual C++ compiler (for 20-Sim) and the GNU C++ compiler (for 20-Works). The MacsML specifications of a Multi-Agent Controller System can be translated to C++ code, which inherits the operating principle captured in the library. The library for Microsoft is embedded with IDITmac so that dll-files can be automatically generated for Multi-Agent Controller Systems.

Directory Structures defined by IDITmac

IDITmac supports two directory structures, a Base Directory Structure and an Organized Directory Structure, of the specification files of a Multi-Agent Controller System. In the Base Directory Structure, the specification files can be located in a single directory (Project Directory) as illustrated in Figure C.2 (a). In the Organized Directory Structure, the specification files can be organized in a predefined structure as illustrated in Figure C.2 (b). As illustrated in the corresponding figures, the generated codes (.h and .cpp) are saved in a “source” directory and the compiled object files (.obj) are stored in a “bin” directory. The generated dll-file (.dll) is stored in a “models” directory.



(a)



(b)

Figure C.2: Directory Structures defined by IDITmac
(a) Base Directory Structure (b) Organized Directory Structure

C++ Code Generation

C++ Codes for agents is generated by **Build+Generate Code** menu. The specification is checked against corresponding Schemas prior to the code generation. For Composite (and Main) Agents, facility of generation of the codes for all its sub agents is also provided (**Build+Generate Code All** menu). The generated codes can be implemented with the developed library.

Generation dll-file for 20-Sim

IDITmac can generate a dll-file for a Multi-Agent Controller System, which can be used with 20-Sim. The specifications are checked against the corresponding Schemas and translated to C++ codes. The translated codes are compiled and linked by the Microsoft Visual C++ Compiler. The compiling and linking outputs (and errors if any) are displayed in the output panel. The compiler is not embedded in the tool and the path of the compiler has to be specified (by **Tools+Resource Directories** menu). Dll-files can be generated by **Build+Build dll** menu.

The structure of the tool, IDITmac, is illustrated in Figure C.3.

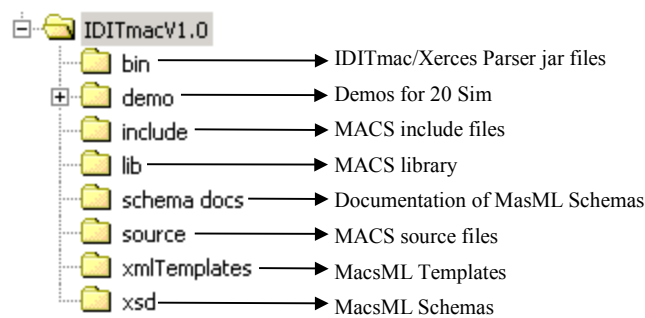


Figure C.3: Organization of IDITmac

APPENDIX D

DEMO LIN SETUP AND MODEL

D.1 Configuration of DemoLin

DemoLin is a demonstration setup developed at Imotec BV, for demonstration purpose of controller performance. It is a mass spring mass system, which is actuated by a linear motor (ironless synchronous permanent magnet). A schematic diagram of DemoLin is presented in Figure D.1. The Motor Mass is driven by the Linear Motor. The End Effectors Mass is connected to the Motor Mass with two flexible Iron Plates. Both the masses are supported by belts and pulleys, which are connected to two shafts. The pulleys of the End Effectors Mass is rigidly connected to the right shaft and connected via bearing to the left shaft. Similarly, the pulleys of the Motor Mass is rigidly connected to the left shaft and connected via bearing with the right shaft.

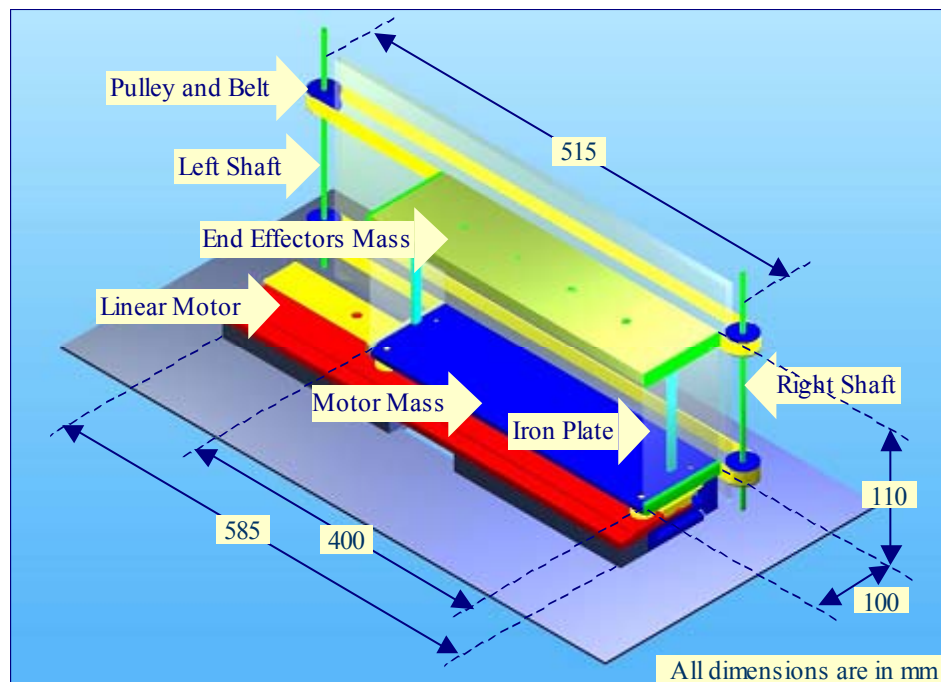


Figure D.1: Schematic Diagram of DemoLin

The dimensions of DemoLin are presented in the figure above. A detailed description of components of DemoLin is as follows.

Mechanical configuration

End Effectors Mass $m_1 = 6[kg]$

Motor Mass $m_2 = 4[kg]$

Stiffness of Iron Plates (in the direction of motion) $c = 956.63[N / m]$

Viscous friction coefficient of the Motor Mass $\mu_v = 2[s / m]$

Motor Description

Type: Ironless synchronous permanent magnet

Model: Yaskawa SGLGW60A365A

Thrust constant $k_m = 63[N / A]$

Servo Amplifier

Type: Yaskawa Servo Pack SGDH 08AE

Gain factor $A = 1.1[A / V]$

D.2 Encoders of DemoLin

A linear encoder is placed beneath the Motor Mass to measure the motor position. A rotational encoder is attached to the right shaft, which gives the position of the End Effectors Mass. Both the encoders are incremental encoders. The rotational encoder has an index pulse, which is triggered when the shaft is in a particular position. The index pulse is encountered twice within a full stroke motion of DemoLin. One index pulse is encountered when the left edge of End Effectors aligns with the left shaft and other is encountered when the left edge of the End Effectors is 76 mm right from the left shaft. The index pulse of the linear encoder cannot be detected during the experiments performed in DemoLin.

D.3 Model of DemoLin

The iconic diagram of model of DemoLin is illustrated in Figure D.2. A viscous friction model is included and the parameter of viscous friction is estimated during experiments. The viscous friction coefficient, μ_v , is taken as $2[s / m]$.

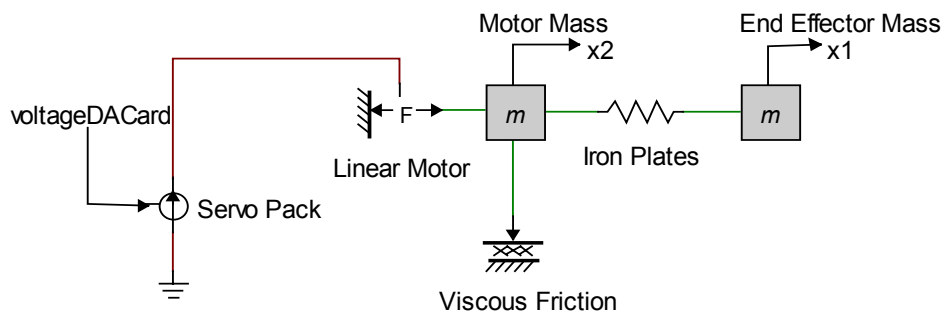


Figure D.2: Iconic Diagram of model of DemoLin

APPENDIX E

DESIGN OF PID-CONTROLLER FOR DEMOLIN

E.1 Plant Model

The model of plant used for controller design is a mass spring mass system, which is described in iconic in Figure E.1.

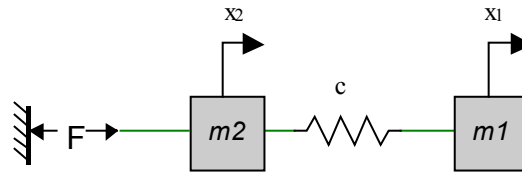


Figure E.1: Iconic Diagram of mass spring mass system

The numeric values are described as follows.

$$m_1 = 6kg$$

$$m_2 = 4kg$$

$$m = 10kg$$

$$c = 956.63N / m$$

where,

- m_1 is mass of the Motor Mass
- m_2 is mass of the End Effectors Mass
- m is total mass
- and c is spring constant of the Iron Plate in the direction of motion.

The dominant stiffness is located in the system thus it is a Flexible Mechanism [46]. Anti-Resonance frequency of the plant is given by

$$\omega_{ar} = \sqrt{\frac{c}{m_1}} = 12.63rad / s$$

Resonance frequency is given by

$$\omega_r = \sqrt{\frac{c}{m_1} + \frac{c}{m_2}} = 19.96rad / s$$

and the frequency ratio is

$$\rho = \left(\frac{\omega_{ar}}{\omega_r} \right)^2 = 0.4$$

Measurement is performed at the Motor (x_2) (Concept AR [46]) and the transfer function of the plant is given by

$$P(s) = \frac{1}{ms^2} \cdot \frac{s^2 + \omega_{ar}^2}{s^2 + \omega_r^2} \cdot \frac{\omega_r^2}{\omega_{ar}^2} = \frac{3.98 \cdot 10^2 s^2 + 6.35 \cdot 10^4}{1.59 \cdot 10^3 s^4 + 6.35 \cdot 10^5 s^2}$$

E.2 Design

A design based on Coelingh [46] is presented in this section. The Optimal dimensionless controller settings for the Concept AR is given by

Position loop quantity and position loop gain:

$$\Omega_p = 0.8$$

$$k_p = m \cdot (\Omega_p \cdot \omega_{ar})^2 = 1020.4$$

Velocity loop quantity and velocity loop gain:

$$\Omega_d = 0.9$$

$$k_d = m \cdot \Omega_d \cdot \omega_{ar} = 113.64$$

derivative time constant is given by

$$T_d = \frac{k_d}{k_p} = 0.111s$$

A PID controller in Series form is implemented

$$U = k_p \cdot \left(\left(1 + \frac{1}{s \cdot T_i} \right) \cdot \frac{1 + s \cdot T_d}{1 + s \cdot \frac{T_d}{N}} \right) \cdot e$$

Integral time constant, $T_i = 1.0$, is chosen such that the integral action does not affect the proportional and the derivative action (the maximum phase shift of the controller), which is illustrated in the bode plots of the PID controller (in Series form) below.

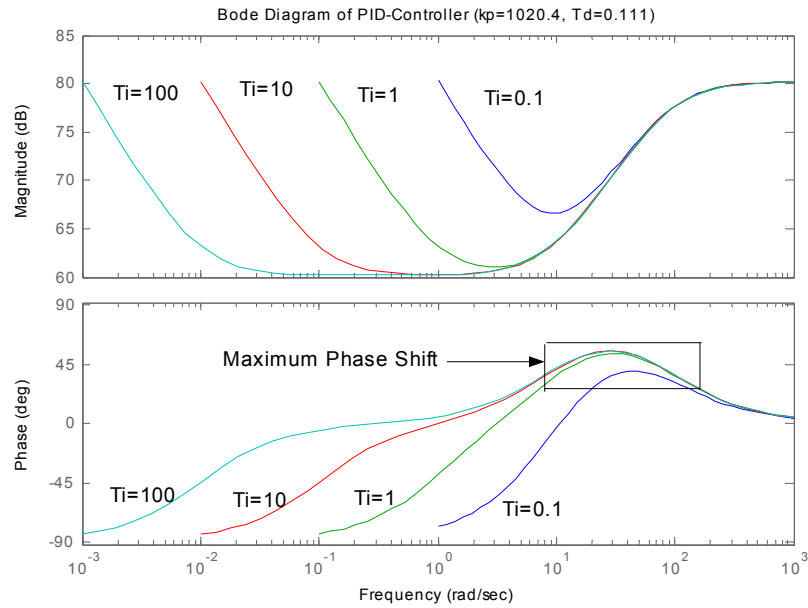


Figure E.2: Bode Plot of PID-Controller

The performance of the controller designed suffers from the friction present in the real system. A good estimation of friction on the system is not available thus, another design is consider in which the gain of the controller is increased to overcome the drawback of the friction.

E.3 Online Parameter Tuning

Firstly, the proportional gain and the derivative time constant of the controller are tuned (with a high value of the integral time constant). The system shows unstable behavior when the proportional gain is approximately 20000, so a lower value of gain is chosen.

After a good value of proportional gain and the derivative time constant is obtained, the integral time constant is adjusted so that it does not affect the proportional and the derivative action. The parameters of the implemented controller are as follows.

$$k_p = 15000.0$$

$$T_d = 0.1$$

$$T_i = 1.0$$

APPENDIX F

MULTI-AGENT CONTROLLER SYSTEM FOR DEMOLIN

DemoLin (Main Agent)

```

 Main Agent: DemoLin.xml
<?xml version="1.0"?>
<mac xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MacSchema.xsd">
  <name>DemoLin</name>
  <interface>
    <ports>
      <input>
        <type>EncoderAgent</type>
        <name>encoderAgent</name>
      </input>
      <output>
        <type>DAAgent</type>
        <name>daAgent</name>
      </output>
    </ports>
  </interface>
  <implementation>
    <composite>
      <cagency>
        <cagent>
          <type>OverallController</type>
          <name>overallController</name>
        </cagent>
      </cagency>
      <connections>
        <connection>
          <from>encoderAgent.x_motor</from>
          <to>overallController.x_motor</to>
        </connection>
        <connection>
          <from>overallController.calibrated</from>
          <to>encoderAgent.calibrated</to>
        </connection>
        <connection>
          <from>overallController.voltage</from>
          <to>daAgent.voltage</to>
        </connection>
      </connections>
    </composite>
  </implementation>
</mac>

```

EncoderAgent (Sensor Agent)


Sensor Agent: EncoderAgent.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>EncoderAgent</name>
  <include>"EncoderInterface.h"</include>
  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>calibrated</name>
      </input>
      <output>
        <type>real</type>
        <name>x_motor</name>
      </output>
      <output>
        <type>real</type>
        <name>x_load</name>
      </output>
    </ports>
    <parameterdefs>
      <parameterdef>
        <type>real</type>
        <name>enc2m_mot</name>
        <defaultvalue>4.0e-6</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>enc2m_load</name>
        <defaultvalue>4.7e-6</defaultvalue>
      </parameterdef>
    </parameterdefs>
  </interface>
  <implementation>
    <sensor>
      <states>
        <state>
          <type>int</type>
          <name>prevEnc2Count</name>
        </state>
      </states>
      <instances>
        <instance>
          <type>EncoderInterface</type>
          <name>encInt</name>
        </instance>
      </instances>
      <start><![CDATA[{
calibrated=0.0;
prevEnc2Count=0;
encInt->Initialize();
encInt->Reset1Mid();
encInt->Reset2Index();
}]]></start>

```

```

        <sense><![CDATA[{
long count;
if (calibrated<0.5)
{
    count = encInt->Enc2();

    if (prevEnc2Count!=count)
        encInt->Reset1Mid();
//reset encoder 1 according to index pulse of encoder 2
    prevEnc2Count=count;
}
count = encInt->Enc1Mid();
x_motor=count * enc2m_mot;
}]]></sense>
</sensor>
</implementation>
</cagentclass>

```

DAAgent (Actuator Agent)

 **Actuator Agent:** DAAgent.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
    <name>DAAgent</name>
    <include>"DAQ.h"</include>
    <interface>
        <ports>
            <input>
                <type>real</type>
                <name>voltage</name>
            </input>
        </ports>
    </interface>
    <implementation>
        <actuator>
            <instances>
                <instance>
                    <type>DAQ_AO</type>
                    <name>da</name>
                </instance>
            </instances>
            <start><![CDATA[{
da->AO(0.0);
}]]></start>
            <actuate><![CDATA[{
da->AO(-voltage);
}]]></actuate>
            <stop><![CDATA[{
da->AO(0.0);
}]]></stop>
        </actuator>
    </implementation>
</cagentclass>

```

OverallController (Composite Agent)


Composite Agent: OverallController.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>OverallController</name>
  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>x_motor</name>
      </input>
      <output>
        <type>real</type>
        <name>voltage</name>
      </output>
      <output>
        <type>real</type>
        <name>calibrated</name>
      </output>
    </ports>
  </interface>
  <implementation>
    <composite>
      <cagency>
        <cagent>
          <type>GoSteady</type>
          <name>goRight</name>
          <parameters>
            <parameter>
              <name>refPositionIncrement</name>
              <value>0.00005</value>
            </parameter>
            <parameter>
              <name>K</name>
              <value>40000</value>
            </parameter>
            <parameter>
              <name>minPositionIncrement</name>
              <value>10e-6</value>
            </parameter>
            <parameter>
              <name>minVoltage</name>
              <value>0.0</value>
            </parameter>
            <parameter>
              <name>maxVoltage</name>
              <value>1.5</value>
            </parameter>
            <parameter>
              <name>startingCount</name>
              <value>500</value>
            </parameter>
          </parameters>
        </cagent>
      </cagency>
    </composite>
  </implementation>
</cagent>

```



```

<type>GoSteady</type>
<name>goLeft</name>
<parameters>
  <parameter>
    <name>refPositionIncrement</name>
    <value>-0.00005</value>
  </parameter>
  <parameter>
    <name>K</name>
    <value>40000</value>
  </parameter>
  <parameter>
    <name>minPositionIncrement</name>
    <value>10e-6</value>
  </parameter>
  <parameter>
    <name>minVoltage</name>
    <value>-1.5</value>
  </parameter>
  <parameter>
    <name>maxVoltage</name>
    <value>0.0</value>
  </parameter>
  <parameter>
    <name>startingCount</name>
    <value>500</value>
  </parameter>
</parameters>
</cagent>
<cagent>
  <type>GoHome</type>
  <name>goHome</name>
</cagent>
<cagent>
  <type>DemoMode</type>
  <name>demoMode</name>
</cagent>
</cagency>
<coordination>
  <class>Sequential</class>
  <name>seq</name>
</coordination>
<connections>
  <connection>
    <from>x_motor</from>
    <to>goRight.x_motor</to>
  </connection>
  <connection>
    <from>x_motor</from>
    <to>goLeft.x_motor</to>
  </connection>
  <connection>
    <from>goLeft.reached</from>
    <to>calibrated</to>
  </connection>
  <connection>
    <from>x_motor</from>
    <to>goHome.x_motor</to>
  </connection>


```

```

        <connection>
            <from>x_motor</from>
            <to>demoMode.x_motor</to>
        </connection>
    </connections>
</composite>
</implementation>
</cagentclass>

```

GoSteady (Elementary Agent)

 **Elementary Agent:** GoSteady.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
    <name>GoSteady</name>
    <interface>
        <ports>
            <input>
                <type>real</type>
                <name>x_motor</name>
            </input>
            <output>
                <type>real</type>
                <name>voltage</name>
            </output>
            <output>
                <type>real</type>
                <name>reached</name>
            </output>
        </ports>
        <parameterdefs>
            <parameterdef>
                <type>real</type>
                <name>refPositionIncrement</name>
                <defaultvalue>0.00005</defaultvalue>
            </parameterdef>
            <parameterdef>
                <type>real</type>
                <name>K</name>
                <defaultvalue>1.0</defaultvalue>
            </parameterdef>
            <parameterdef>
                <type>real</type>
                <name>minPositionIncrement</name>
                <defaultvalue>4.7e-6</defaultvalue>
            </parameterdef>
            <parameterdef>
                <type>real</type>
                <name>minVoltage</name>
                <defaultvalue>0.0</defaultvalue>
            </parameterdef>
            <parameterdef>
                <type>real</type>
                <name>maxVoltage</name>
                <defaultvalue>1.5</defaultvalue>
            </parameterdef>
        </parameterdefs>
    </cagentclass>

```

```

    <parameterdef>
      <type>real</type>
      <name>startingCount</name>
      <defaultvalue>100</defaultvalue>
    </parameterdef>
  </parameterdefs>
</interface>
<implementation>
  <elementary>
    <states>
      <state>
        <type>real</type>
        <name>prevX_motor</name>
      </state>
      <state>
        <type>int</type>
        <name>wait_counter</name>
      </state>
    </states>
    <start><![CDATA[{
wait_counter=0;
reached=0.0;
voltage=0.0;
}]]></start>
    <finalize><![CDATA[ {
voltage=0.0;
reached=1.0;
}]]></finalize>
    <activation><![CDATA[ {
if (wait_counter<startingCount)
return 1.0;
else
return ((double) (fabs(x_motor-
prevX_motor)>minPositionIncrement));
}]]></activation>
    <calculate><![CDATA[ {
voltage=(refPositionIncrement-(x_motor-prevX_motor))*K;
if (voltage>maxVoltage)
voltage=maxVoltage;
if (voltage<minVoltage)
voltage=minVoltage;
wait_counter++;
prevX_motor=x_motor;
}]]></calculate>
  </elementary>
</implementation>
</cagentclass>

```

GoHome (Elementary Agent)


Elementary Agent: GoHome.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>GoHome</name>
  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>x_motor</name>
      </input>
      <output>
        <type>real</type>
        <name>voltage</name>
      </output>
    </ports>
  </interface>
  <implementation>
    <composite>
      <cagency>
        <cagent>
          <type>FinPosThirdOrderPath</type>
          <name>homePath</name>
          <parameters>
            <parameter>
              <name>Tm</name>
              <value>1.0</value>
            </parameter>
            <parameter>
              <name>Td</name>
              <value>2.0</value>
            </parameter>
            <parameter>
              <name>Hfin</name>
              <value>0.0</value>
            </parameter>
            <parameter>
              <name>samplingtime</name>
              <value>0.001</value>
            </parameter>
          </parameters>
        </cagent>
        <cagent>
          <type>PID_s_sp</type>
          <name>pidController</name>
          <parameters>
            <parameter>
              <name>K</name>
              <value>5000.0</value>
            </parameter>
            <parameter>
              <name>Td</name>
              <value>1.2</value>
            </parameter>
            <parameter>
              <name>Kd</name>
              <value>0.0</value>
            </parameter>
          </parameters>
        </cagent>
      </cagency>
    </composite>
  </implementation>
</cagentclass>

```

```

        <name>N</name>
        <value>10</value>
    </parameter>
    <parameter>
        <name>Ti</name>
        <value>1.0</value>
    </parameter>
    <parameter>
        <name>minimum</name>
        <value>-3.0</value>
    </parameter>
    <parameter>
        <name>maximum</name>
        <value>3.0</value>
    </parameter>
    <parameter>
        <name>MV_scale</name>
        <value>1.0</value>
    </parameter>
    <parameter>
        <name>output_scale</name>
        <value>0.01443</value>
    </parameter>
    <parameter>
        <name>macError</name>
        <value>0.01</value>
    </parameter>
    <parameter>
        <name>sampletime</name>
        <value>0.001</value>
    </parameter>
    </parameters>
</cagent>
</cagency>
<connections>
    <connection>
        <from>x_motor</from>
        <to>homePath.prevX</to>
    </connection>
    <connection>
        <from>homePath.x</from>
        <to>pidController.SP</to>
    </connection>
    <connection>
        <from>x_motor</from>
        <to>pidController.MV</to>
    </connection>
    <connection>
        <from>pidController.output</from>
        <to>voltage</to>
    </connection>
</connections>
</composite>
</implementation>
</cagentclass>

```

DemoMode (Composite Agent)

 **Composite Agent:** DemoMode.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>DemoMode</name>
  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>x_motor</name>
      </input>
      <output>
        <type>real</type>
        <name>voltage</name>
      </output>
    </ports>
  </interface>
  <implementation>
    <composite>
      <cagency>
        <cagent>
          <type>PID_s_sp</type>
          <name>pidController</name>
          <parameters>
            <parameter>
              <name>K</name>
              <value>15000.0</value>
            </parameter>
            <parameter>
              <name>Td</name>
              <value>0.1</value>
            </parameter>
            <parameter>
              <name>N</name>
              <value>10.0</value>
            </parameter>
            <parameter>
              <name>Ti</name>
              <value>1.0</value>
            </parameter>
            <parameter>
              <name>minimum</name>
              <value>-3.0</value>
            </parameter>
            <parameter>
              <name>maximum</name>
              <value>3.0</value>
            </parameter>
            <parameter>
              <name>MV_scale</name>
              <value>1.0</value>
            </parameter>
            <parameter>
              <name>output_scale</name>
              <value>0.01443</value>
            </parameter>
          </parameters>
        </cagent>
      </cagency>
    </composite>
  </implementation>
</cagentclass>
    
```

```

        </parameter>
        <parameter>
            <name>macError</name>
            <value>0.005</value>
        </parameter>
        <parameter>
            <name>sampletime</name>
            <value>0.001</value>
        </parameter>
    </parameters>
</cagent>
<cagent>
    <type>PeriodicThirdOrderPath</type>
    <name>periodicThirdOrderPath</name>
</cagent>
</cagency>
<connections>
    <connection>
        <from>x_motor</from>
        <to>pidController.MV</to>
    </connection>
    <connection>
        <from>periodicThirdOrderPath.x</from>
        <to>pidController.SP</to>
    </connection>
    <connection>
        <from>pidController.output</from>
        <to>voltage</to>
    </connection>
</connections>
</composite>
</implementation>
</cagentclass>

```

FinPosThirdOrderPath (Elementary Agent)

	Elementary Agent: FinPosThirdOrderPath.xml
---	---

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
    <name>FinPosThirdOrderPath</name>
    <interface>
        <ports>
            <input>
                <type>real</type>
                <name>prevX</name>
            </input>
            <output>
                <type>real</type>
                <name>x</name>
            </output>
        </ports>
        <parameterdefs>
            <parameterdef>
                <type>real</type>
                <name>Tm</name>
                <defaultvalue>1.0</defaultvalue>
            </parameterdef>
        </parameterdefs>
    </interface>
</cagentclass>

```

```

        </parameterdef>
        <parameterdef>
            <type>real</type>
            <name>Td</name>
            <defaultvalue>2.0</defaultvalue>
        </parameterdef>
        <parameterdef>
            <type>real</type>
            <name>Hfin</name>
            <defaultvalue>1.0</defaultvalue>
        </parameterdef>
        <parameterdef>
            <type>real</type>
            <name>samplingtime</name>
            <defaultvalue>0.001</defaultvalue>
        </parameterdef>
    </parameterdefs>
</interface>
<implementation>
    <elementary>
        <states>
            <state>
                <type>real</type>
                <name>time</name>
            </state>
            <state>
                <type>real</type>
                <name>Ho</name>
            </state>
            <state>
                <type>real</type>
                <name>Hm</name>
            </state>
        </states>
        <initialize><![CDATA[ {
Ho=prevX;
Hm=-Ho+Hfin;
time=0.0;
}]]></initialize>
        <activation><![CDATA[ {
if (!active)
    return 1.0;
else
    return (time<(Tm+Td));
}]]></activation>
        <calculate><![CDATA[ {
double u;
u=time/Tm;
if (u<0.25)
    x=Ho+Hm*(16.0*u*u*u/3);
else
    if (u<0.75)
        x=Ho+Hm*(1.0/6.0-2.0*u+8.0*u*u-(16.0*u*u*u/3));
    else
        if (u<1.0)
            x=Ho+Hm*(-13.0/3.0+16.0*u-
16.0*u*u+16.0*u*u*u/3.0);
        else
            x=Ho+Hm;
}]]></calculate>
    </elementary>
</implementation>
</model>

```



```

    time=time+samplingtime;
    }]]></calculate>
  </elementary>
</implementation>
</cagentclass>

```

PID_s_safety (Elementary Agent)



Elementary Agent: PID s safety.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>PID_s_safety </name>
  <interface>
    <ports>
      <input>
        <type>real</type>
        <name>SP</name>
      </input>
      <input>
        <type>real</type>
        <name>MV</name>
      </input>
      <output>
        <type>real</type>
        <name>output</name>
      </output>
    </ports>
    <parameterdefs>
      <parameterdef>
        <type>real</type>
        <name>K</name>
        <defaultvalue>1020.0</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>Td</name>
        <defaultvalue>0.1174</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>N</name>
        <defaultvalue>10</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>Ti</name>
        <defaultvalue>1000</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>minimum</name>
        <defaultvalue>-5.0</defaultvalue>
      </parameterdef>
      <parameterdef>
        <type>real</type>
        <name>maximum</name>

```

```
        <defaultvalue>5.0</defaultvalue>
    </parameterdef>
    <parameterdef>
        <type>real</type>
        <name>MV_scale</name>
        <defaultvalue>1.0</defaultvalue>
    </parameterdef>
    <parameterdef>
        <type>real</type>
        <name>output_scale</name>
        <defaultvalue>0.01443</defaultvalue>
    </parameterdef>
    <parameterdef>
        <type>real</type>
        <name>maxError</name>
        <defaultvalue>0.05</defaultvalue>
    </parameterdef>
    <parameterdef>
        <type>real</type>
        <name>sampletime</name>
        <defaultvalue>0.001</defaultvalue>
    </parameterdef>
</parameterdefs>
</interface>
<implementation>
    <elementary>
        <states>
            <state>
                <type>boolean</type>
                <name>isSafe</name>
            </state>
            <state>
                <type>real</type>
                <name>error</name>
            </state>
            <state>
                <type>real</type>
                <name>scaled_MV</name>
            </state>
            <state>
                <type>real</type>
                <name>factor</name>
            </state>
            <state>
                <type>real</type>
                <name>uD</name>
            </state>
            <state>
                <type>real</type>
                <name>uI</name>
            </state>
            <state>
                <type>real</type>
                <name>ideal_output</name>
            </state>
            <state>
                <type>real</type>
                <name>prevError</name>
            </state>
        </states>
    </elementary>
</implementation>
</model>
```

```

        </states>
        <start><![CDATA[ {
isSafe=true;
    }]]></start>
        <initialize><![CDATA[ {
prevError=0.0;
    }]]></initialize>
        <finalize><![CDATA[ {
output=0.0;
    }]]></finalize>
        <activation><![CDATA[ {
return (isSafe);
    }]]></activation>
        <calculate><![CDATA[ {
scaled_MV = MV_scale * MV;
error = SP - scaled_MV;
if (fabs(error)>maxError)
{
    output=0.0;
    isSafe=false;
    return;
}
factor = 1 / ( sampletime + Td / N );
uD = factor * (sampletime * K *error + Td * K * (error
- prevError) + Td * uD / N );
uI = uI + sampletime * uD / Ti ;
ideal_output = uI + uD;
output = output_scale * ideal_output;
if (output<minimum)
    output=minimum;
if (output>maximum)
    output=maximum;
prevError=error;
    }]]></calculate>
        <update><![CDATA[ {
scaled_MV = MV_scale * MV;
error = SP - scaled_MV;
    }]]></update>
        </elementary>
    </implementation>
</cagentclass>

```

PeriodicThirdOrderPath (Composite Agent)

 **Composite Agent:** PeriodicThirdOrderPath.xml

```


<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CagentSchema.xsd">
  <name>PeriodicThirdOrderPath</name>
  <interface>
    <ports>
      <output>
        <type>real</type>
        <name>x</name>
      </output>
    </ports>
  </interface>
  <implementation>
    <composite>
      <cagency>
        <cagent>
          <type>ThirdOrderPath</type>
          <name>rightPath</name>
          <parameters>
            <parameter>
              <name>Tm</name>
              <value>0.95</value>
            </parameter>
            <parameter>
              <name>Td</name>
              <value>1.0</value>
            </parameter>
            <parameter>
              <name>Ho</name>
              <value>0.0</value>
            </parameter>
            <parameter>
              <name>Hm</name>
              <value>0.1</value>
            </parameter>
            <parameter>
              <name>samplingtime</name>
              <value>0.001</value>
            </parameter>
          </parameters>
        </cagent>
        <cagent>
          <type>ThirdOrderPath</type>
          <name>leftPath</name>
          <parameters>
            <parameter>
              <name>Tm</name>
              <value>0.95</value>
            </parameter>
            <parameter>
              <name>Td</name>
              <value>1.0</value>
            </parameter>
            <parameter>
              <name>Ho</name>
              <value>0.0</value>
            </parameter>
            <parameter>
              <name>Hm</name>
              <value>0.1</value>
            </parameter>
            <parameter>
              <name>samplingtime</name>
              <value>0.001</value>
            </parameter>
          </parameters>
        </cagent>
      </cagency>
    </composite>
  </implementation>
</cagentclass>
    
```

```

        <name>Ho</name>
        <value>0.1</value>
    </parameter>
    <parameter>
        <name>Hm</name>
        <value>-0.1</value>
    </parameter>
    <parameter>
        <name>samplingtime</name>
        <value>0.001</value>
    </parameter>
</parameters>
</cagent>
</cagency>
<coordination>
    <class>Cyclic</class>
    <name>cyclic</name>
</coordination>
</composite>
</implementation>
</cagentclass>

```

Sequential (Coordination Object)

 **Coordination Object:** Sequential.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CoordinationObjectSchema.xs
d">
    <name>Sequential</name>
    <implementation>
        <coordination>
            <states>
                <state>
                    <type>Int</type>
                    <name>lastActiveAgentIndex</name>
                </state>
            </states>
            <start><![CDATA[ {
lastActiveAgentIndex=0;
}]]></start>
            <resolute><![CDATA[ {
if (lastActiveAgentIndex>=mu.size())
    return 0.0; //all agents activated
double maxActivation;
maxActivation=0.0;
for (int i=lastActiveAgentIndex; i<mu.size(); i++)
{
    if (maxActivation<mu[i])
        maxActivation=mu[i];
}
return maxActivation;
}]]></resolute>
            <decide><![CDATA[ {
if (acknowledge)
{
    if (lastActiveAgentIndex>=mu.size())

```

```

        return;          //all agents activated
    for (int i=lastActiveAgentIndex; i<mu.size(); i++)
    {
        if (mu[lastActiveAgentIndex]>0)
        //active agent still wants to be active
        {
            ack[lastActiveAgentIndex]=acknowledge;
            return;
        }
        else
        {
            lastActiveAgentIndex=(lastActiveAgentIndex+1);
        //activate next agent
            if (lastActiveAgentIndex==mu.size())
                return;          //all agents activated
        }
    }
}
//else
//lastActiveAgentIndex=0;      //restart after
activating;
}
]]></decide>
    <combine><![CDATA[
        out [0]= subAgentOutput[lastActiveAgentIndex];
        //return output of active agents
    ]]></combine>
</coordination>
</implementation>
</cagentclass>

```

Cyclic (Coordination Object)



Coordination Object: Cyclic.xml

```

<?xml version="1.0"?>
<cagentclass xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="CoordinationObjectSchema.xs
d">
    <name>Cyclic</name>
    <implementation>
        <coordination>
            <states>
                <state>
                    <type>Int</type>
                    <name>lastActiveAgentIndex</name>
                </state>
            </states>
            <start><![CDATA[
lastActiveAgentIndex=0;
]]></start>
            <resolute><![CDATA[
double maxActivation;
maxActivation=0.0;
for (int i=0; i<mu.size(); i++)
{
    if (maxActivation<mu[i])
        maxActivation=mu[i];
}
]]></resolute>

```

```
}
return maxActivation;
}
]]></resolute>
<decide><![CDATA[
{
if (acknowledge)
for (int i=0;i<mu.size(); i++)
{
if (mu[lastActiveAgentIndex]>0)
//active agent still wants to be active
{
ack[lastActiveAgentIndex]=acknowledge;
return;
}
else
{
lastActiveAgentIndex=(lastActiveAgentIndex+1) %
mu.size(); //activate next agent
}
}
else
lastActiveAgentIndex=0; //restart cycle if inactive
]]></decide>
<combine><![CDATA[{
out [0]= subAgentOutput[lastActiveAgentIndex];
//return output of active agents
}]]></combine>
</coordination>
</implementation>
</cagentclass>
```


BIBLIOGRAPHY

- [1] T. A. Johansen and B. A. Foss. *ORBIT - Operating Regime Based Modeling and Identification Toolkit*, Control Engineering Practice, 6:12, 77–86, 1998
- [2] Leith, D. and W. Leithead, *Analytic framework for blended multiple model systems using linear local models* International Journal of Control (1999)
- [3] P. Antsaklis, X. Koutsoukos, and J. Zaytoon, *On hybrid control of complex systems: A survey* European Journal of Automation, 32(9-10): 1023–1045, 1998.
- [4] M. Zefran and J. W. Burdick, *Design of switching controllers for systems with changing dynamics*, Proceedings 37th Conference on Decision and Control, pp. 2113–2118, 1998
- [5] G.J. Pappas, G. Lafferriere, S. Sastry. *Hierarchically consistent control systems*. IEEE Transactions on Automatic Control, vol. 45:6, pp. 1144–1160, 2000.
- [6] T. Moor, J. Raisch, J.M. Davoren, *Computational advantages of a two-level hybrid control architecture*, Proceedings 40th IEEE Conference on Decision and Control, pp. 358-363, December 2001
- [7] L. Wills, S. Kannan, B. Heck, G. Vachtsevanos, C. Restrepo, S. Sander, D. Schrage, and J. V. R. Prasad, *An open software infrastructure for reconfigurable control systems*, in Proceedings 19th American Control Conference (ACC-2000), Chicago, IL, June 2000, pp. 2799-2803.
- [8] M. Luck, M. d’Inverno. *A conceptual framework for agent definition and development*. The Computer Journal, 44(1):1–20, 2001.
- [9] S. Franklin, A. Graesser, “Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents”, Intelligent Agents III: Agent Theories, Architectures, and Languages, Proceedings of ECAI’96 Workshop (ATAL), Hungary, Aug. 1996 (From Lecture Notes in Artificial Intelligence 1193, pp. 21-35, 1997).
- [10] <http://www.dcs.gla.ac.uk/mac/>, MAC - Multi-Agent Control: Probabilistic reasoning, optimal co-ordination, stability analysis and controller design for intelligent hybrid systems, A four year Research Training Network within the European Commission's 5th Framework, starting in April 2000.
- [11] <http://www.ee.byu.edu/ee/control/MASC.html>, MASC - Multi-Agent Satisficing Control Group, Brigham Young University, Department of Electrical & Computer Engineering, Provo, UT, USA.
- [12] <http://www.cc.gatech.edu/ai/robot-lab/research/multi-agent.html>, Multiagent Robotic Systems, Mobile Robot Lab, Georgia Institute of Technology
Atlanta, Georgia, USA
- [13] <http://www.lti.pcs.usp.br/mappel/>, MAPPEL - Multi-Agent Collaborative and Adversarial Perception, Planning, Execution, and Learning, Project in collaboration of Escola Politécnica da Universidade de São Paulo,

- Universidade de São Paulo, Universidade Estadual de Campinas, Instituto Tecnológico de Aeronáutica, Carnegie Mellon University.
- [14] <http://hms.ifw.uni-hannover.de>, [HMS] - *Holonic Manufacturing Systems*, project Intelligent Manufacturing Systems (IMS) program in collaboration with The Broken Hill Proprietary Co. Ltd. (Australia), University of Calgary (Canada), Softing GmbH (Europe), Yaskawa Electric Corporation (Japan), Rockwell Automation/Allen-Bradley LLC (USA)
- [15] <http://www.rt.el.utwente.nl/agent/>, *Agent-Oriented Design of Control Systems*, Control Laboratory, Faculty of Electrical Engineering, Mathematics & Computer Science, University of Twente, Enschede, The Netherlands
- [16] R. A. C. Bianchi, A. H. R. C. RILLO, *A distributed control architecture for a purposive computer vision system* IEEE Symposium on Image, Speech and Natural Language Systems (ISNL) - IEEE International Joint Symposia on Intelligence and Systems 1996, P. 288-294. 1996
- [17] S. Bussmann, K. Schild: *Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology*, Proceedings 4th International Conference on Multi-agent Systems (ICMAS' 2000), Boston, MA, USA, 2000, pp.87-94.
- [18] R. W. Brennan, M. Fletcher, D. H. Norrie, *An agent-based approach to reconfiguration of real-time distributed control systems*, IEEE Transactions on Robotics and Automation, Vol.18, Iss.4, 2002 Pages: 444- 451
- [19] A.J.N. van Breemen. *Water vessels under control*, January 1996. Practical assignment report, University of Amsterdam, Intelligent Autonomous Systems
- [20] A.J.N. van Breemen, T.J.A. de Vries. *Design and implementation of a room thermostat using an agent-based approach*. Control Engineering Practice, 9(3):233-248, 2001.
- [21] A.J.N. van Breemen, T.J.A. de Vries. *An agent-based framework for designing multi-controller systems*. Proc. Of the Fifth International Conference on The Practical Applications of Intelligent Agents and Multi-Agent Technology, pages 219-235, 2000.
- [22] A.J.N. van Breemen, T.J.A. de Vries, J.B. Striper. *An agent-based framework for local model approaches*. 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, EPFL, August 2000.
- [23] A.J.N. van Breemen. *An Agent-Based Multi-Controller Systems, A design framework for complex control problems*. PhD thesis, University of Twente, Enschede, The Netherlands, 2001.
- [24] J. van Amerongen, H.J.Coelingh and T.J.A. de Vries. *Computer support for mechatronic control system design*, Robotics and Autonomous Systems, vol. 30, nr. 3, pp. 249 - 260, PII: SO921-8890(99)00090-1, 2000.
- [25] <http://www.mathworks.com/>, The MathWorks Consulting Services, Matlab/Simulink
- [26] <http://www.dspace.de>, dSpace

- [27] Masanobu Koga, *MaTX/RtMaTX: A Freeware for Integrated CACSD*, Proc. of CACSD'99, Kohala Coast-Island, Hawai'i, U.S.A., pp.451-456 (1999)
- [28] <http://www.ni.com/>, National Instruments, LabView
- [29] <http://20sim.com>, Controllab Products B.V., 20-Sim
- [30] <http://www.informatik.hu-berlin.de/top/pnml/> *Petri Nets Markup Language*, Standardization of XML based interchangeable format for Petri Nets.
- [31] G. Frey, M. Minas. *Internet-based development of logic controllers using signal interpreted Petri nets and IEC 61131*. In Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2001), Orlando (FL) USA, volume 3, pages 297-302, July 2001.
- [32] <http://www.holobloc.com/> HOLOBLOC™, Function Block Based, Holonic Systems Technology
- [33] <http://xml.apache.org/xerces2-j/index.html>, Xerces 2 Java Parser
- [34] <http://www.w3.org/XML/Schema>, W3C (World Wide Web consortium) XML Schema
- [35] <http://www.w3.org/TR/2000/REC-xml-20001006>, Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000
- [36] <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/> XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001
- [37] <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/> XML Schema Part 1: Structures, W3C Recommendation, 2 May 2001
- [38] <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/> XML Schema Part 2: Data types, W3C Recommendation, 2 May 2001
- [39] http://www.xmlspy.com/products_ide.html/ XMLSpy 5
- [40] C. S. Horstmann, *Practical object-oriented development in C++ and Java*, Wiley Computer Publications, 1997, ISBN: 0-471-14767-2 (pbk)
- [41] B. P. Douglass, *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*, Addison-Wesley, 1999, ISBN: 0-201-49837-5
- [42] H.M. Deitel, *XML: how to program*, Prentice Hall, 2001, ISBN: 0-13-028417-3
- [43] C. Hughes, T. Hughes, *Mastering the standard C++ classes: an essential reference*, Wiley computer publications, 1999, ISBN: 0-471-32893-6 (pbk./CD-ROM)
- [44] <http://java.sun.com/j2se/1.4.1/docs/api/>, Java™ 2 Platform, Standard Edition, v 1.4.1, API Specification, Java™ Technology
- [45] <http://msdn.microsoft.com/library/default.asp>, MSDN Library, Microsoft Corporation

- [46] H.J Coelingh, Design Support for Motion Control Systems: a Mechatronic Approach, Ph.D. Thesis, University of Twente, Enschede, The Netherlands. 2000
- [47] <http://www.w3schools.com/>, W3CSchools Online Web Tutorial, W3CSchools.com