Agent Based Design Validation and Verification

Dissertation

Submitted to School of Computing, University of the West of Scotland, in cooperation with the Hamburg University of Applied Sciences

Maan Al-Homci

Agent Based Design Validation and Verification

Dissertation

UWS UNIVERSITY OF THE WEST of SCOTLAND

Hamburg University of Applied Sciences

Submitted to School of Computing, University of the West of Scotland, in cooperation with the Hamburg University of Applied Sciences

By Maan Al-Homci, born in Hamburg-Germany,

to reach the degree

PhD in Computing Science

Supervisors:
Prof. Dr. Colin Fyfe (University of the West of Scotland)
Prof. Dr.-Ing. Wolfgang Gleine (Hamburg University of Applied Sciences)

Keywords: Multi-agent systems, agents learning, control learning, context based learning, system validation and verification agents, functional testing agents

ABSTRACT

ABSTRACT

This thesis describes an approach for the modelling of agent-oriented software systems for the validation and verification of complex systems which are built of software modules acting autonomously and target-oriented (pro-active) and communicating with their environment, so-called software agents. This approach shows a new methodology for the validation and verification process as an improvement of the classical testing process for highly complex systems.

More and more software systems, specifically in the domain of system verification, are characterized by distributability, dynamics and openness. Software agents shall facilitate the development of such complex systems. Agent-based systems are built modular, robust and dynamically adaptable. The term “software agent” is designated by the core features autonomy, pro-activity and structured interaction. By means of these features, agents of objects and components are delimited.

The software-technical development of agent-based systems demands a suitable language of modelling and a process-oriented model which regulates utilization of language. Up to now only inadequate proposals exist in the field of validation and verification of systems for both. This thesis contributes to the solution of this set of problems as follows:

System abstraction:

A new abstract, layered and formal model of the systems to be tested shows that utilization of different agent implementations can be realised more efficiently. Unlike the existing models, not only a single agent but a system of agents which may have different tasks and characteristics, (multi-agent system) can be modelled. The use of the layered abstraction with its appropriate knowledge representation has enabled the use of agent-oriented software development in the area of system validation and verification and has been demonstrated in the case studies of this thesis. For the knowledge representation new expression formulations from which agents’ knowledge and resulting agents’ rules can be derived, show that thereby comprehensive test requirements can rapidly and efficiently be converted into agents’ rules.

Context-oriented agent models:

New context-oriented agent models which are applied in the different layers of system abstraction show that better adaptivity with respect to the system and its environment can be realized by context-oriented learning capability. The case studies of this thesis have demonstrated how context-based learning radically improves the test preparation and test coverage, especially when limited knowledge about the system to be tested is given.

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

First and foremost, I wish to express my deepest thanks to my supervisor Prof. Dr. Colin Fyfe (University of the West of Scotland) for his valuable advice and guidance of this work and for reviewing the thesis and for his valuable comments and criticism during the preparation of the manuscript as well as for revising the English language of the manuscript. He has helped me in shaping the research from the first day, pushed me to get through the unavoidable research setbacks, and encouraged me to achieve to the best of my ability. Without his support and encouragement, this dissertation would not have been possible.

Special thanks to my second supervisor Prof. Dr. Wolfgang Gleine (Hamburg University of applied Sciences) for his intense support, valuable advice and sincere comments which helped me immensely to finish this study. I also want to express my thanks to Airbus Germany for their support, including working equipment and research space. I would like to thank many people for their support, encouragement and guidance during my years as a employee in Airbus Germany.

I also thank my department members, whose comments were helpful in refining the dissertation into its final form. Special thanks are due to the staff of the verification and validation team (EVC) in Airbus Germany, and particularly to Jan Bobolecki, Josef Kruse for their assistance.

My family has always been, and continues to be there for me at all times. Finally, I am particularly grateful to my wife Heidrun Mayer who has been incredibly supportive, understanding, and encouraging as she has supported me through the entire graduate experience.

PREFACE

PREFACE

The research presented in this dissertation is original work by the author except where otherwise indicated. Some parts of the dissertation include revised versions of published papers. The dissertation has not been submitted in whole or part for a degree at any other University or Institution.

The length of this dissertation excluding footnotes and appendices is approximately 61.000 words.

Maan Al-Homci

CONTENTS

TABLE OF FIGURES

PART I
INTRODUCTION
&
LITERATURE REVIEW

PART ONE - INTRODUCTION AND STATE OF THE ART

Part one of the thesis identifies and describes the problem that has been analysed and solved in this research. This part consists of two chapters. The first one is an introduction, by which the thesis has been summarized. The objective of the introduction was to provide a general view of the research work and a better understanding of the problem to be solved. It explains what the thesis is about and, more importantly, justifies why our research work is significant. The problem motivating the research for this thesis is described. A rating of the problem is presented and the new contribution is put into context.

The second chapter provides the state of the art regarding agent-based studies, research and techniques. It describes existing modelling approaches, their advantages and disadvantages, their applicability, their usability within inter-system integration complexity. This chapter lays out a detailed study of previous and current research. It brings the reader up to date with the latest research and development in the area. It also provides an understanding and knowledge of the present and most recent work.

CHAPTER ONE INTRODUCTION

1. Introduction

Systems become more and more complex and thus require improvement of the process with respect to their design and implementation. In the last few years, in languages and methods, the paradigm of object-orientation for mastering the complexity in software development has been established. Current and future systems, as for example airplane testing systems and their inter-connection, are characterized by more and more dynamics and openness. The development of such systems shall be facilitated and mastered on the basis of specific concepts, such as software agents. Suitable models and methods for the development of agent-based testing systems are the subject of the current research. Problems involved therewith are the origin of this thesis. After introductory motivation, the issues of this thesis are delimited and, based thereon, the objectives of the thesis are formulated.

For the improvement of the validation and verification process on equipment, system and multi-system level, new research and new methods in the last years have handled the complexity and the applicability with limited options for a real improvement because of their own complexity and the difficulty of using them in the industrial world. The use of agent-oriented approach has shown a promising research approach for the industrialisation of this novel methodology and can be used in the area of validation and verification of complex systems. The case studies of this thesis have demonstrated the new use of this approach in several verification areas with promising results so that the use of agent-oriented methodology for the verification of systems can be considered as a serious approach. In this thesis the use of software agents in the area of complex system testing has been researched for the first time and has been demonstrated in three real world case studies of the validation and verification of aircraft systems.

1.1. The typology of agents

Software agent modelling is a rapidly developing research area. This introduction section presents a typology of agents to overview their areas of applicability, characteristics and the rapidly evolving area of software agents.

This section attempts to place existing agents into different agent classes, i.e. its goal is to investigate a typology of agents. A typology refers to the study of types of characteristics. There are several dimensions to classify existing software agents. Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of static or mobile agents.

Secondly, they may be classified as either deliberative or reactive. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic, reasoning model, and they engage in planning and negotiation in order to achieve coordination with other agents. Work on reactive agents originated from research carried out by [Brooks 1991]. These agents on the contrary do not have any internal, symbolic models of their environment, and they act using a stimulus/response type of behaviour by responding to the present state of the environment in which they are embedded [Ferber 1999]. Indeed, Brooks has argued that intelligent behaviour can be realised without the sort of explicit, symbolic representations of traditional AI [Brooks 1991].

Thirdly, agents may be classified by several ideal and primary attributes which agents should exhibit. In this research work we have identified a minimal list of four: autonomy, learnability, cooperability and portability. We assume that any such list is contentious, but it is no more or no less so than any other proposal. Hence, we are not claiming that this is a necessary or sufficient set for our application of software agents within the system V&V (validation and verification) perimeter. Autonomy refers to the principle that agents can operate on their own, depending on their internal state and knowledge. A key element of their autonomy is their pro-activeness, i.e. their ability to take the initiative rather than acting simply in response to their environment [Wooldridge 1995]. Cooperation with other agents is essential: it is the results of having multiple agents in the first place in contrast to having just one. In order to cooperate, agents need to possess a social ability, i.e. the ability to interact with other agents and possibly humans via some communication language [Wooldridge 1995]. Lastly, for agent systems to be truly adaptive, they would have to learn as they react and/or interact with their external environment. In our view, agents are (or should be) intelligent. Though we did not attempt to define what intelligence is, we maintained that a key attribute of any intelligent being is its ability to learn and to be adaptive to environmental changes. To be portable is the ability of agents to run on different platforms and in different environments. We used these four minimal characteristics in Figure 1 to derive four types of agents to be included in our typology: collaborative agents, learning agents, interface agents and truly adaptive agents.

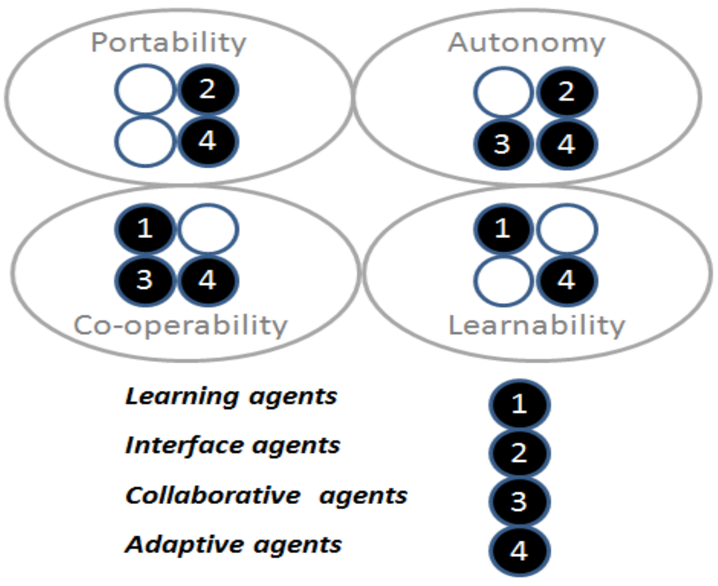


Figure 1 A part view of an agent typology

1.2. Motivation

In the last decades many industrial researchers tried to solve three main issues, the management and definition of tests, the requirements and the verification activities in the area of the validation and verification process. In my own work I was involved in several research projects handling these three main issues:

<u>Test management:</u>

For test management and planning the CATEGA (Computer Aided Test Generation Assistant) research project has been started in 2003 based on the Software Testing Plan (IEEE 829). The aim of this project has been the definition of tests and how they have to be handled in the very complex field of aircraft testing. Within this project a test process and testing tool chain has been developed and has already been established and used in aircraft cabin testing since 2006.

Requirements management

In order to be able to manage a large number of requirements including their incorporation in the test definition activities, different research activities have been founded to solve this complex problem.

- Care Marcos (Formal management using excel automation) to formalise requirements and to enable the traceability of them. The linking between them and the CATEGA toll chain has been established as well. The problem of this approach was the highly complex update management of such solutions because of the missing data base management.
- DOORS (Dynamic Object Oriented Requirements System) is an IBM solution (originated by the company TeleLogic) for the management of requirements. The aim of this research work was the integration of the tool DOORS in the CATEGA tool chain. This solution has already been established and used in aircraft cabin testing since 2007.

Test modelling

In the last years the modelling of systems has been established as a part of the system development process and therefore the need for a complementary solution for testing was required. Several research works have been founded to handle this complex issue:

- KATO (System modelling by using HybridUML - UML with real time extension); this research project was a cooperation with the University of Bremen (2004-2007) for using UML modified by the University for realtime capability. The problem of this approach was that it was highly complex, time- costly and requires fundamental changes in the system development process.
- AVMA (Advanced virtual modelling approach - using SysML); this research project was in cooperation with the University of Hamburg (2009-2011). The problem of this approach was the complex configuration, not the missing robustness against evolutionary development life cycle.
- The current approach (Agent based system validation and verification approach) developed in this thesis is a continuation of these testing improvement researches in order to solve this complex issue with appropriate solutions that can be used in the testing of aircraft systems. This thesis handled the design and implementation of agent-oriented software for the validation and verification of complex systems. This dissertation is attended by a research and technology project founded by the Test Department of Airbus GmbH Germany in cooperation with the UWS, HAW (High School of Applied Sciences) and implemented by the company BKUS (Germany Hamburg). The evaluation of this approach has been performed by three case studies of a real application within the aircraft cabin perimeter.

An agent-oriented approach is useful for systems where the inputs are unpredictable, the environment rapidly changes and the decisions or reactions are independent and autonomous. To satisfy the required tasks, agents shall be able to exhibit the following capabilities:

- Rapid reaction capability to handle a rapidly changing environment.
- Be autonomous and independent, especially when the required request or consolidation time is not given.
- Failure of agent shall be handled as environmental change and rapidly intercepted by other agents involved.

The limitations and weaknesses of existing agent-oriented methodologies and the demands to develop the multi-agent integration platform (MAIP) for system validation and verification are discussed in Chapter 2, however the detailed discussion of such problems is not the main scope of this thesis; consequently, no detailed discussion of such problems has been provided here, though a brief discussion has been provided. The discussion has not been limited to problems of a single particular methodology. Instead, it will address problems that relate to one methodology or relate to a number of methodologies. The following is a discussion of problems found during this research work:

Solving the problems of test modelling provides motivation to come up with a novel approach towards a comprehensive agent-oriented software engineering methodology for multi-agent systems development for the validation and verification of complex systems. This novel methodology is an attempt to overcome most of the limitations stated above.

This thesis provides a new approach to agent-oriented software engineering that can be used for system engineering, especially in the validation and verification of inter-system functions. The new approach is based on exploitation and reuse of existing methodologies. This is done by means of unifying existing methodologies by combining their strong points as well as avoiding their limitations and weaknesses. It can be considered as a generic solution by using different simulation, system, agent and agent-helper-models. The composition of different technologies and models facilitates the new agent-based approach with the capability of handling the complex system validation and verification challenge.

1.3. Scope of this Research

Software agent development is a large research area containing many overlapping sub-problems:

- to model agents for performing certain tasks,
- to specify the appropriate multi agent platform that can host distributed agents,
- to determine management capability required for a certain usage domain,
- to determine the right agent type for a certain task,
- to define the sufficient state and state attributes required to perform a certain task,
- to determine an appropriate knowledge management capability regarding learnability and adaptivity,
- to ensure deterministic behaviour if required to perform a certain task,
- to ensure real time capability if required to perform a certain task in a given time constraint.

Based on the natural progression, these tasks can be roughly categorised into three sub-areas which are essential for modelling agents for the system validation and verification domain:

- analysing of the main objectives to be achieved and requirements to be fulfilled,
- determining if existing agent system implementations are sufficient or need to be improved or extended,
- modelling of agents and multi agent platforms for these tasks.

The core of this dissertation is a framework that illustrates the benefits and the feasibility of applying an agent based approach to system validation and verification processing. Given the size of the research area, it is not feasible, and also not necessary, to conduct all agent modelling methods on all potential tasks. The benefits and the feasibility are illustrated by application of an improved agent modelling approach to the system testing domain. This domain has been chosen for a number of reasons. Firstly, this domain has not been sufficiently investigated for the use of an agent based approach due to the complexity of the entities involved and due to the constantly expanding difficulty of the test case and test coverage definitions. The potential opportunity exhibited by using an agent-based approach could be an enabler for exploring new development approaches in order to reduce the complexity of system testing domains regarding test case definition and test coverage securing.

The second reason is the scope of challenges represented by these documents. In order to apply adaptive and smart methods to the system testing domain, certain problems must be solved.

In particular, this dissertation proposes an agent-based approach to system validation and verification and, in supporting this approach, a knowledge representation scheme, an agent communication method, a distribution of agent helpers and an evaluation method were determined. All of these can be applied to other engineering domains with little modification.

Finally, the classical system validation and verification approaches are commonly used in the real world of complex system testing; they are one of the approaches accepted by established research institutions and regulating organisations. Using a real world application adds potential industrial impact to the research. Overall, the complexity, diversity and relevance of the system testing domain make it suitable as an agent based application in this dissertation.

1.4. Thesis Question

The principal question addressed in this thesis is:

Can agents derive test cases from existing component abstraction models which are relevant and applicable within a certain testing context in real-time, highly mutable, collaborative and competitive environments?

More specifically, the thesis contributes to an agent architecture enabling the use of layered abstraction and context-based learning techniques to improve an agent's behaviour in the domain of system validation and verification with the following characteristics:

- a need for real-time decision-making;
- several independent agents with the same well-defined high level goal (verification of systems);
- interaction with different sources of knowledge;
- the ability to process the sensory information and to use it to update a world model.

This thesis contributes to a successful method using layered abstraction and context-based learning with respect to equipping such agents with sufficient behaviours in such a domain.

1.5. Contributions

This thesis makes two distinct contributions to the fields of system verification and multi-agent systems.

<u>Layered-Abstraction (LA)</u>

The layered-abstraction architecture is suitable for domains with periodic opportunities for fully safe communication among distributed agents and testing entities (test equipments and systems to be verified), interleaved with periods of limited communication and a need for real-time action. This architecture includes mechanisms for task decomposition and dynamic role assignment as well as for communication in distributed single, time-critical communication environments. It is designed and implemented in both simulated and real systems. This abstraction is essential for including knowledge representation and enabling context representation in which different types of adaptivity and learning can be applied.

<u>Context-Based Learning (CBL)</u>

Context-based learning is a nested agent learning paradigm that combines multiple agent learning modules, each directly affecting the next. Context-based learning is described in general and then illustrated as a set of four interconnecting learned behaviours (function, interface, state and performance behaviours) within a complex, real-time, collaborative and competitive domain.

Through real implementation in the complex aircraft cabin testing domain, this dissertation does not only show the feasibility, but also the benefits arising from the multi agent integration platform

(MAIP) and the multi-level agent helper aided verification method including knowledge representation derived by test requirements generated for the testing tasks. In addition, opportunities for generalisation to other domains are discussed.

1.6. The Organisation of this Dissertation

This dissertation is divided into three main parts, as illustrated in Figure 2:

- Part one: Introduction and literature review.
- Part two: Methodology.
- Part three: Case study and discussion.

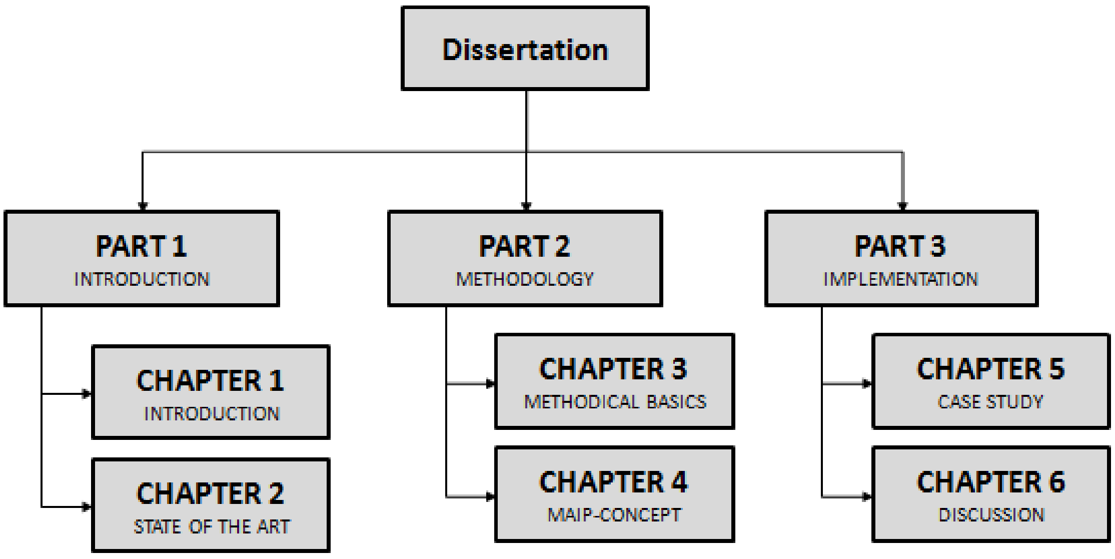


Figure 2 A Organisation of the dissertation

An essential requirement of this thesis is that today's standard of system validation and verification is based on the support of the agent based approach. For the development of agent models in the field of system validation and verification, a new approach of layered system abstraction and context based learning form the effective starting point.

- PART ONE – Introduction and State of the Art
  - Chapter 1 introduces the motivation and the main objectives of the thesis. This chapter presents the rationale behind the development of a new multi-agent system development methodology.
  - Chapter 2 presents the problem statement of this research which defines the difficulties of the agent-oriented methodologies and provides a detailed discussion of the limitations and shortcomings of existing state of the art methodologies. This chapter presents the agent definition of this research with respect to the generic approach of functional validation and verification of multi-system solutions.

- PART TWO – Methodology
  - Chapter 3 presents the definition, the methodical basis of the MAIP entities and the test process to be realized by using an agent based approach.
  - Chapter 4 presents the definition and a general overview of the MAIP concept on the one hand and a specific overview of the functional validation and verification of multi-system solutions on the other hand.

- PART THREE – Implementation
  - Chapter 5 introduces a case study of the MAIP framework implementation for some well-known existing validation and verification methodologies. The case study looks at common features among different methodologies used in building agents for the area of system testing and at how agent behaviour is captured.
  - Chapter 6 presents the research contributions, deficiencies, discussion and conclusion of the thesis and future work.

CHAPTER TWO STATE OF THE ART

2. State Of The Art

This chapter is an introduction to the research fields of system validation and verification, testing workflow, and agent based approach. The first part of this chapter (Section 2.1) states definitions of the central terms for the research fields, and discusses how these research fields are related. The rest of the chapter describes the agent architecture, methodologies and multi agent platforms in more detail, identifies some research challenges in these research fields, and describes how this approach relates to these research fields and challenges.

The difficulties of innovation development are due to the interactions between the systems and their environment. Thinking comprehensively, the corresponding solution would be to describe as much information as possible in models so as to avoid undesired side effects of the construction and of the use of technical systems. The increase of information increases the complexity of the functions and models. The increase of the complexity is definitely allowed and is, contrary to the trends of the last decades, desired today [Lottor 2010].

Science has discovered that technical systems as parts of a complex environment cannot be developed by representing this environment in an oversimplified way. It must try to reproduce reality as real as possible. Consequently, the complexity of the functionality and of its environment models requires new methods of protection for these functions, as previous methods of classical testing were designed for less complex systems.

With the increasing complexity of the problems to be modelled the importance of intelligent software agents has significantly increased since the late eighties, [Timm 2003], [Wooldridge 1995]. Especially in the areas of information logistics, in which distributed, dynamic processes play a major role, the proliferation of agent technology has significantly increased. Some of these areas are production, e-commerce, process control, telecommunications, logistics, and business process management [Foster 2004]. Here, the implicit distribution by modelling independent software entities will be taken into account; these accomplish their goals either independently or on behalf of another agents [Timm 2003].

2.1. Verification, Validation and Test

As explained in the previous paragraph, the objective of system development is to realise users' wishes which a new technical system has to fulfil, via software and/or hardware. Instead, the wishes are represented and communicated to the system developers by means of requirements. In the process of requirements establishment, due to the different perceptions with respect to the system to be developed, requirements could be generated which do not correspond to the users' wishes any more. The objective of validation of a system or a component is to show that “the right system, respectively the right component” is developed and that all users' wishes are considered. The IEEE software glossary defines validation as “The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements“. One possibility of validation is the review. A review is “a process or a meeting in the course of which a work result or a group of work results is presented to project members, managers, users, clients or other interested parties for the purpose of commenting, voting or approval’’.

When verifying a system or a component, it is assumed that the requirements have already been validated. Thereby it is supposed that the requirements with respect to the question “Are we developing the proper system?” are correct. Verification responds to the question “Are we developing the system properly?”. Its objective is to verify if the developed system (the component, respectively) fulfils the requirements. IEEE 610.12 defines verification as “The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase".

In order to show that a system fulfils the requirements made in the beginning of the development, either its correctness can be proved, or the system can be made subject to testing. It can be tested if a software program fulfils its formal specification. The definition of proof of correctness is as follows: "Proof of correctness - (1) A formal technique used to prove mathematically that a computer program satisfies its specified requirements. (2) A proof that results from applying the technique in (1)" reasoning requires a mathematical process which per se requires a formal data basis. Frequently, proofs of correctness are performed on the basis of implementation models (or parts thereof).

Depending on the size and specification of the system, its correctness cannot always be proved, since mathematical processes either require too much calculating time and/or memory capacity, or there is no formal specification of the software program. Testing is a further process to verify a system. Contrary to proofing, testing, however, does not offer a one hundred percent statement with respect to the correctness of the system. The definition of testing as a verification process is as follows: An activity in which a system is activated under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system. At the time of testing, the system to be verified is called "system under test "(SUT). During execution of the test cases against the SUT are performed. A test case is "(1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (2) (IEEE Std 829-1983 151) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item." The test case includes requirements to the SUT and its test environment which must be fulfilled at the time of testing, input parameters and the expected behaviour of the SUT in the form of output parameters. In the case where the observed behaviour corresponds to the expected behaviour, the test case has been successful, otherwise it has failed, and the differing behaviour will be examined and evaluated.

In the last years the V-model of the validation and verification (V&V) process has been established in the industry to manage the V&V activities during the whole system development process. The V-model represents a system development process which may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V-shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively. In our thesis we distinguished between three types of testing activities, as shown in Figure 3:

- The left side of the V-Model describes the validation activities within the system design phase starting with functional requirements, definition, system specification and equipment detailed specification. Usually only the equipment level specification is detailed enough so that the supplier can start with the manufacturing activities. The validation (ensuring that the right function or product is implemented) activities (activity type 1 in Figure 3) are to be performed to ensure that the equipment level specification, the system specification and the global function requirements are appropriate for the airlines' needs. These validation activities are called "design validation" in this thesis.
- The right side of the V-Model describes the verification activities that are consisting of two types of testing activities: the verification activities to ensure that the function or product is correctly implemented (activity type 2 in Figure 3) and the validation activities to ensure that the implemented function or product is the correct one. The verification activities on the right side of the V-Model are called "product validation" in this thesis (activity type 3 in Figure 3).

In this thesis the verification and the product validation activities were handled and proposed. The handling of the design validation activities could be focused in the future work based on this thesis.

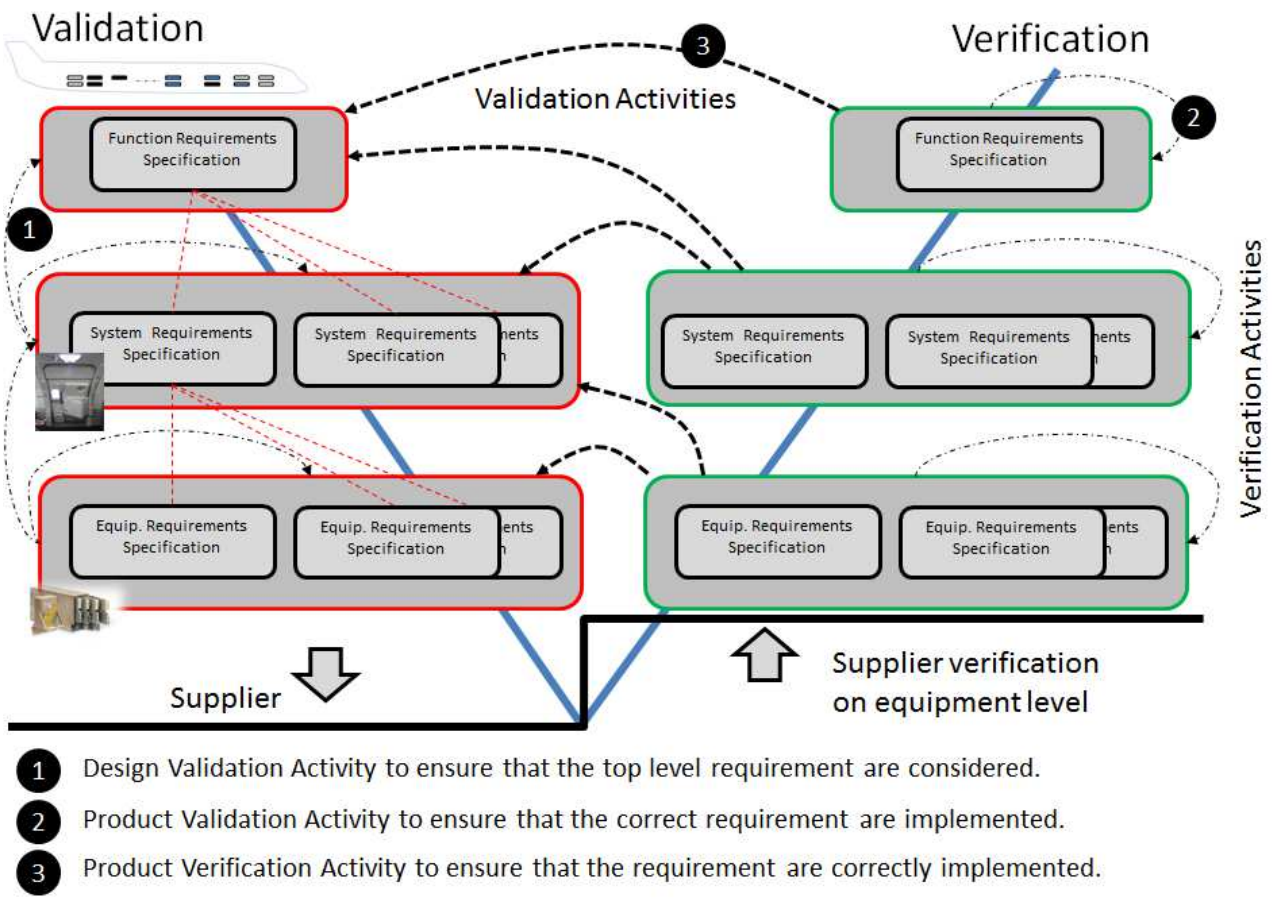

Validation
Verification
Validation Activities
Verification Activities
Function Requirements Specification
Function Requirements Specification
System Requirements Specification
System Requirements Specification
System Requirements Specification
System Requirements Specification
Equip. Requirements Specification
Equip. Requirements Specification
Equip. Requirements Specification
Equip. Requirements Specification
Supplier
Supplier verification on equipment level
1 Design Validation Activity to ensure that the top level requirement are considered.
2 Product Validation Activity to ensure that the correct requirement are implemented.
3 Product Verification Activity to ensure that the requirement are correctly implemented.

Figure 3 V-Model of validation and verification

2.2. Agent Definition

In the literature there are many attempts to define the term agent. An early definition for example is still heavily based on programming language concepts, such as the definition of [Mihalis Giannakis 2011] shows:

"[Agents] communicate with their peers by exchanging messages in an expressive agent control language. While Agents can be as simple as subroutines, typically they are larger entities with some sort of persistent control."

Once a problem has to be cooperatively solved, a direct or indirect interaction shall take place. A direct interaction could be termed communication and indirect interaction via the environment, stigmergy [Bonabeau 1999], [Ricci 2011]. In agent technology, message exchange by means of a dedicated agent language is important but not the main focus. Instead of classical function or method calls, data and respective information will be sent by messages. Exchange of messages can be seen as a highly flexible basis for agent communication, since a system of agents can fully dynamically react to structure at run-time taking into account environmental changes. Message exchange by adding or removing of agents is still possible, but a function call is not possible any more. This enhanced flexibility allows their use in open environments [Mihalis Giannakis 2011]. Subsequent definitions are more general and understand agents as software entities that can perceive and change their environment [Weiss 2001],[Wooldridge 2002].

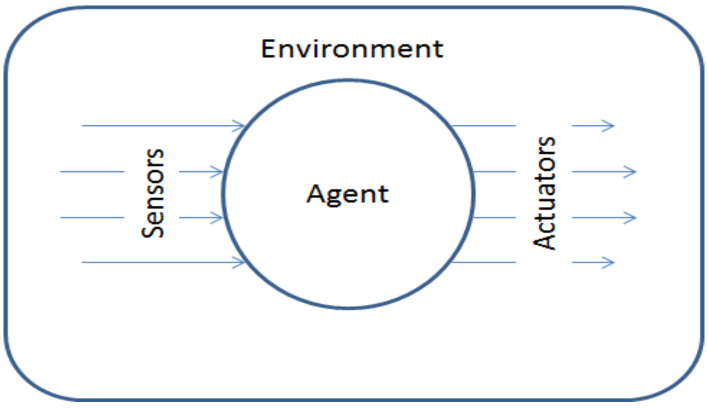


Figure 4 Agent environment

Figure 4 shows an abstract view of an agent which is in an environment. This environment can be perceived and changed by an agent by means of sensors and effectors / actuators. Perception and influence of the environment form a cycle; a new environment change can cause reactions by the agent. As an example an agent may be mentioned here that monitors the inventory of a warehouse. If the stock falls below a certain value, then the agent will trigger an order. Once the order has arrived, the environment of the agent has changed so that a new order for a certain period of time is no longer necessary.

A large part of the agent community accepts a generic and sufficient agent definition based on the article by Michael Wooldridge and Nicholas Jennings [Wooldridge 1995]. The so-called "Weak Notion of Agency" has four basic properties:

- Autonomy: Agents operate without the direct intervention of humans or other entities and have at least partial control over their actions and their conditions.
- Social ability: Agents interact with each other (or with humans) via some kind of communication language.
- Reactivity: Agents perceive their environment and react promptly to changes.
- Pro-activeness: Agents not only react to changes in environment, but actively pursue their own goals.

The term of agents by Wooldridge and Jennings [Wooldridge 1999] is defined as follows:

"An agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible autonomous action in that environment in order to meet its design objectives."

The topic "Software Agents" has become one of the most striking topics in computer science research. The term "Software Agent" leads to a wide argument with respect to what a software agent is, and how it could be clearly distinguished from a program.

Examining the question, "What is a software agent?" raises many arguments on what a software agent is, and what the difference between a software agent and a computer program is. Researchers have proposed many definitions for the concept of a software agent. Each of them introduced his/her definition according to his/her point of view. Some of them concentrated on artificial intelligence approaches, others concentrated on software engineering approaches. We concentrate on the definitions that are well known and most accepted by agent researchers, such as [Wooldridge 2000], [Xudong 2012] etc.

"A software agent (or autonomous agent or intelligent agent) is a computer program which works toward goals (as opposed to discrete tasks) in a dynamic environment (where change is the norm) on behalf of another entity (human or computational), possibly over an extended period of

time, without continuous direct supervision or control, and exhibits a significant degree of flexibility and even creativity in how it seeks to transform goals into action tasks." [Krupansky 2008]

These definitions are summarized as follows:

Agent: A persistent software unit that performs a set of tasks on behalf of a user or a computer system with the following capabilities:

- Functioning with high degree of autonomy (autonomy is the agent ability to perform tasks with minimum intervention by the user. The agents have control over their tasks and resources and will take part in cooperative activities only if they decide to do so).
- Interacting with other agents or humans using specific agent communication languages.
- Adapting and acting on its environment through sensors, and reacting to the changes of the environment through respective effectors.
- Applying its knowledge to make decisions.
- Cooperating with other agents or humans either by negotiation or coordination to achieve the same common goals.
- Achieving its goals by performing appropriate roles and following appropriate plans.
- Gaining experience and knowledge to store the successful plans.
- Being adaptable with the environment changes by responding in a timely convention.
- Having initiative (self-acting).

Agent modelling with Petri-Nets: For the modelling of agents and the representation of their basic properties the approach defined by Rölke has been chosen [Heiko Rölke 2004]. In this thesis this approach has been used to specify agents and their properties generically so that they can be implemented independently by using another approach or specification mark-up languages. Based on the definition of Rölke the following aspects have been considered while agent modelling:

- **Clearly encapsulating:** any external interaction with an agent can be only performed by messaging (input / output interface). We extend the capability of agents to be able to delegate some tasks directly to their Helper (supporting programs) in order to perform specific tasks where adaptivity is not required (e.g. parsing of file systems, analysis of formally defined interface specification and so on)
- **Autonomy:** Only the internal state and the personal knowledge of an agent represented by its instance is used to define or identify the next step to be performed
- **Adaptivity:** Messages received by agent (sent by other agents or real world sensory) can lead to changes of the agent instance. The decision as to whether any changes are required depends on the internal algorithms of the agent represented by its instance.
- **Capability to interact with other agents and environment:** A specific communication language and protocol has enabled the interaction capability between all multi-agent system entities. In this thesis the socket-based communication defined and used by the test engine suppliers has been chosen. A specific adaption of this kind of communication has been realized and encapsulated by the definition of the component control model (see Section 3.9).

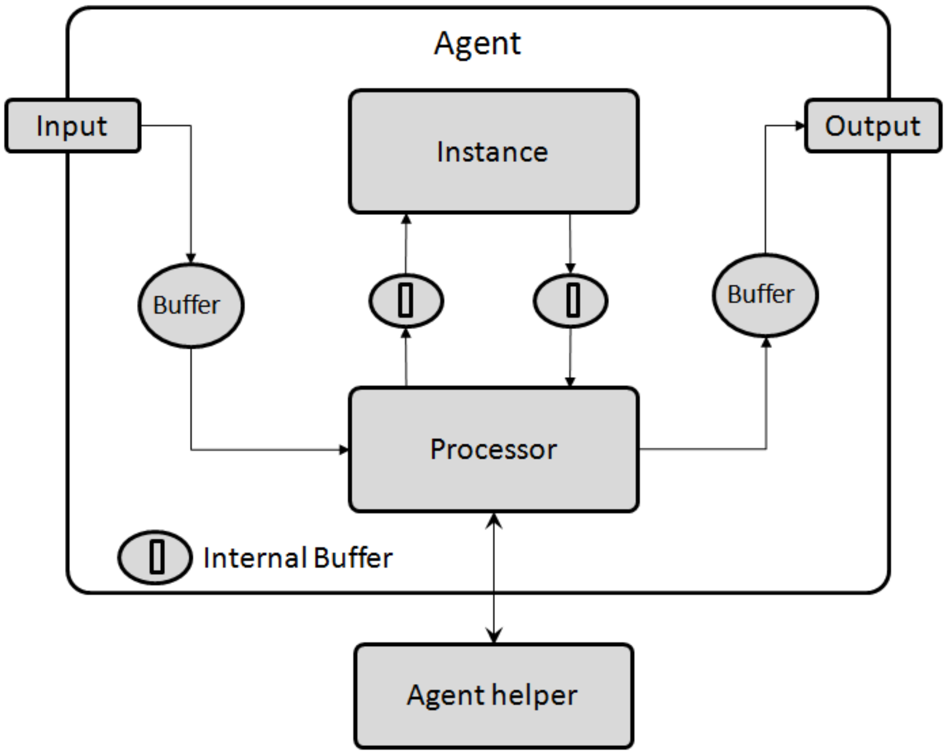


Figure 5 Agent basic model

2.3. Agent Architectures

This section describes the internal structures of agents from a technical perspective in order to provide a basis for implementation of agent properties (see Chapter 2.2). In the literature there are a number of different approaches; most of them are closely linked to an application domain or a specific problem. Müller [Müller 1997] describes four basic types of independent application-layer architecture considering the other classifications [Kirn 2006-Ref3], [Weiss 2001], [Wooldridge 1995]. These are briefly explained in the following sections.

2.3.1. BDI Agent

The BDI architecture is one of the most well-known, established and studied software agents' architectures [Georgeff 2009]. This architecture is based on four fundamental components: beliefs, desires, intentions, and plans. In the BDI architecture, the agent's beliefs represent information that the agent gains about the world that in many cases may be incomplete or incorrect [D'Inverno 1997]. The contents of these beliefs can be anything from knowledge about the agent's environment to general facts an agent must know in order to act rationally. The desires of an agent are a set of long-term goals, wherein a goal is typically a description of a desired state of the environment. An agent's goals could simply represent some desired end state. These goals may be defined by a user or may be adopted by the agent. New goals may be adopted by an agent due to an internal state change in the agent, an external change of the environment, or because of a request from another agent. State changes may cause goals or plans to be triggered or new information to be adopted that may cause the generation of a new goal or new information. Requests for information or services from other agents may cause an agent to adopt a goal that it currently does not possess. An agent's desires provide it with motivations to act. When an agent chooses to act on a specific desire that desire becomes an intention of the agent. The agent will then try to achieve these intentions until it believes the intention is satisfied or the intention is no longer achievable or valid [D'Inverno 1997]. The intentions of an agent provide a commitment to perform a plan. Although not mentioned in the acronym, plans play a significant role in this architecture. A plan is a representation outlining a course of action that, when executed, allows an agent to achieve a goal or desire.

2.3.2. Reactive Architecture

Reactive agents perform actions on certain stimuli in their environment. But in the strict implementation of the reactive architecture neither an internal model of the environment nor some kind of deduction will be required.

In this thesis the interaction with the environment has been realized by using specific agent-adapters and event-based messaging (see concept in Chapter 4). Intelligent behaviour in terms of “Weak Notion of Agency” can still be achieved if a large number of reactive agents solve common tasks. In addition, Brooks [Brooks 1991] mentions that the intelligent behaviour can arise without explicit representation and without deduction as emergent properties of complex systems. One example is in the NetLogoTM [Wilensky 1999] model developed in [Resnick 1994]

Wooldridge [Wooldridge 2002] mentioned the following positive features of Weak Notion of Agency: simplicity, economic performance and robustness against errors. The negative features of the Weak Notion of Agency approach are the limited accessibility and actuality of sufficient information that stimulates the agents. In this thesis the accessibility and actuality of sufficient information has been increased by using agent-adapters which are able to access the relevant information needed to achieve the test objectives (see concept in Chapter 4).

Furthermore, obsolete information can be limitedly used only (short-term view). The possibilities of using learning reactive agents are also limited. A classical and established software engineering approach for adequate methods does currently not exist, since a large number of agents in environments possess enormous interaction complexity and dynamics. This so-called emergent behaviour is by definition very difficult to model and formalize in advance. However, they are not actually truly reactive agents. The majority of reactive architectures can be modelled using a basic “IF-THEN” rule structure. In this thesis modified reactive agents have been used to realize the MAIP implementation approach.

2.3.3. Deliberative Architecture

Traditional studies and approaches of artificial intelligence suppose that intelligent behaviour can be built on the basis of a representation of the environment and the desired behaviour. In this sense, deliberative agents have been initially designed as encapsulated expert systems, where knowledge by deduction was used to solve theorems [Mihalis Giannakis 2011], [Wooldridge 2002] Later concepts from different areas have extended these approaches, such as behavioural research. These involve objectives and plans. The BDI approach is a common deliberative agent model [Bonini 2013] which can be implemented in the agent world according to Weiss [Weiss 2001] as follows:

- Beliefs (knowledge): The current world model of the agent.
- Desires (goals): Fundamental factors affecting behaviour. Goals represent desirable states.
- Intentions (plans): From the world model and based on a chosen target (autonomous) a plan will be generated that includes the intentions of the agent.

The advantage of reactive based deliberative agent architectures is a lesser planning effort to perform or to achieve the goal-oriented action or action sequence. The reactive agent defined by stimulus-response patterns allows apparently faster action with a given input. Consequently, the reactive agents have a higher efficiency for similar objectives in most cases [Timm 2003]. On the other hand, the advantage of deliberative agents is that new ways of solving problems can be identified with an appropriate knowledge base. Therefore, reactive agents can be closely linked to behaviour leading to the design and implementation decision, and consequently efficient responses can be realized.

The deliberative agent architecture contains an explicitly represented, symbolic model of the world. Decisions (for example on what actions are to be performed) are made via logical reasoning, based on pattern matching and symbolic manipulation [Mihalis Giannakis 2011].

In order to build an agent in this way that can be used for the approach of V&V (Validation and Verification) of complex systems, there are at least two important problems that have to be solved:

- The transduction problem: The transduction of the real world into an accurate, adequate description or model will be realized by use of specific agent-adapters and event-based messaging (see concept in Chapter 3).
- The representation/reasoning problem: The symbolically represented information on complex real-world entities has been built in this thesis by using multi-level agent architecture (see concept in Chapter 3).

2.4. Multi-Agent Systems

In the literature, there are a number of definitions of multi-agent systems (MAS) [Xudong 2012], [Kirn 2006-2], [Luck 2005], [Wooldridge 2002] which can be subsumed under the following definition:

"Multi-agent systems are systems of multiple interacting agents that are integrated in an environment" [Wooldridge 2002].

The number of agents can vary widely depending on the problem and type of the agents used and agent frameworks. The applicability fields of agents vary from one agent (e.g. heating control [Weiss 2001]), some agents (supply chain management [Xudong 2012],) to thousands of agents (Dynamic Pricing in the Information economics [Kephart 2000]) or several hundred thousands agents [Marrow 2001] for simulating a look-up system for peer-to-peer networks. The agent population may be heterogeneous [Xudong 2012], wherein different organizational roles are realized by different types of agents. The role model, motivated by the social sciences and organizational theory, is a prerequisite for institutional coordination and support structuring of flexible system behaviour [Kirn 2006-Ref3]. An exemplary structure of a multi-agent system is shown in Figure 6.

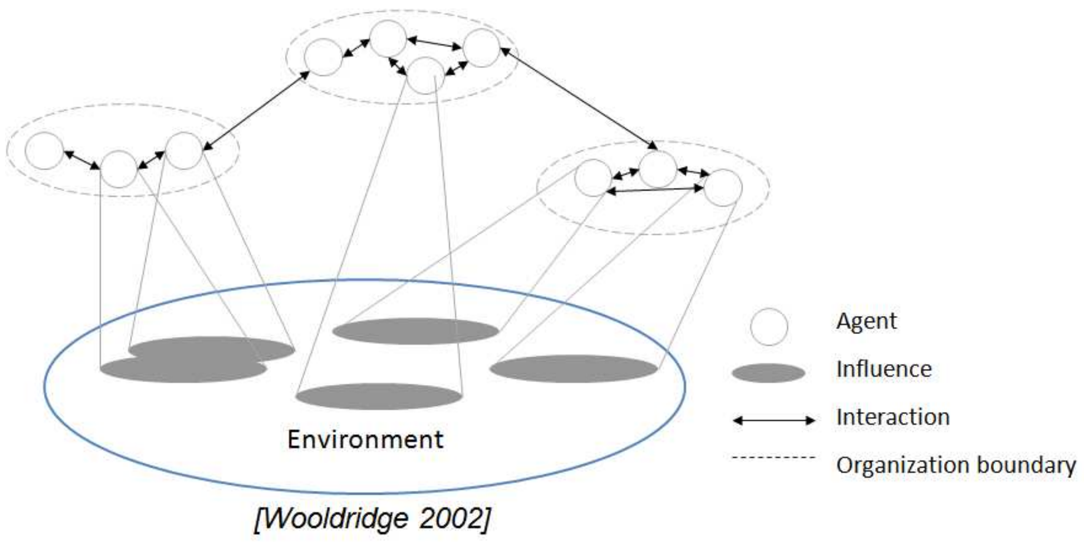


Figure 6 Example of the structure of a multi-agent system

As Figure 6 shows, an agent has incomplete information, limited possibilities and subsequently limited influence on its environment including all other agents. For all distributed agent no global system of control exists [Xudong 2012]. Therefore, communication and interaction mechanisms take a key role in any operational behaviour. A characterization of structures in multi-agent systems based on organizational structure consists of three dimensions [Kirn 2006-Ref3]:

- Capabilities: On the basis of heterogeneous agents the ability to solve problems that cannot be solved by an individual agent. In contrast to the parallel computing or grid computing flexibility - not scalability - is the primary topic here. This makes the management of heterogeneous resource providing for wide application domains an important success factor. In the research area cooperative distributed planning [Durfee 2001], [Durfee 1989], and resource management [Timm 2003] are good examples.
- Duration: Another aspect is the consideration of the duration of structures. The existence of the structure can be valid over the entire duration of the MAS, for each problem or even reassembled for each transaction. For permanent structures the design of the communication and coordination mechanisms can be statically defined and implemented. This allows efficient decision-making with little overhead regarding predefined problems. The disadvantage here is the lower flexibility of the overall system when the environment changes. Here dynamic structures are required that allow adequate and dynamic reorganization at runtime. However, it decreases the efficiency, since a certain proportion of the resources are used for restructuring.
- Decision-making: Similar to the duration of structures, the decision-making can be implemented or realized statically or dynamically [Sandholm 1999]. Here, too, the balance of desired flexibility and efficiency will decide which solution is to be implemented.

The above statement leaves open the question of the dimensions of the multi-agent system, e.g. how many agents in total or per role are required. The infrastructure provides an important basis to enable service requests and communications. For example the use of so-called Yellow Pages [Bellifemine 2011] causes a bottleneck when the number of agents, communication and information in the system increases [Marrow 2001]. One approach to circumvent this bottleneck is described in the DIET project [Marrow 2001]. No Yellow Pages are used here, but agents who interact on a peer-to-peer basis and are randomly connected on demand by the infrastructure. All these approaches still use a centralized infrastructure to produce at least the initial contact (agent framework) and system software and hardware to send messages or to use computing power, bandwidth and disk space. Similarly, all agent frameworks and investigated methodologies use the common terms agent and messages [Zambonelli, Jennings, and Wooldridge 2003], [Bellifemine 2011], [Marrow 2001], [Padgham 2002], [Leal 2001].

All approaches and of course the approach in this thesis still use a centralized infrastructure to produce at least the initial contact (agent framework) and system software and hardware to send messages or to use computing power, bandwidth and disk space. Similarly, all agents, investigated methodologies and frameworks use the terms agent and message in the same common relevance. Implicitly, the infrastructure of an agent framework can be considered as an environment of agents.

2.5. Agent communication and interaction

One of the most essential components of embodied multi-agent systems are the communication and interaction mechanisms based on it. One of the suitable metaphors has been defined by Michael Luck [Luck 2005]: "Computing as interaction”. Figure 7 shows an overview of coordination mechanisms. This section gives a brief outline of important communication and interaction mechanisms in the agent field based on Huhns [Huhns 2012].

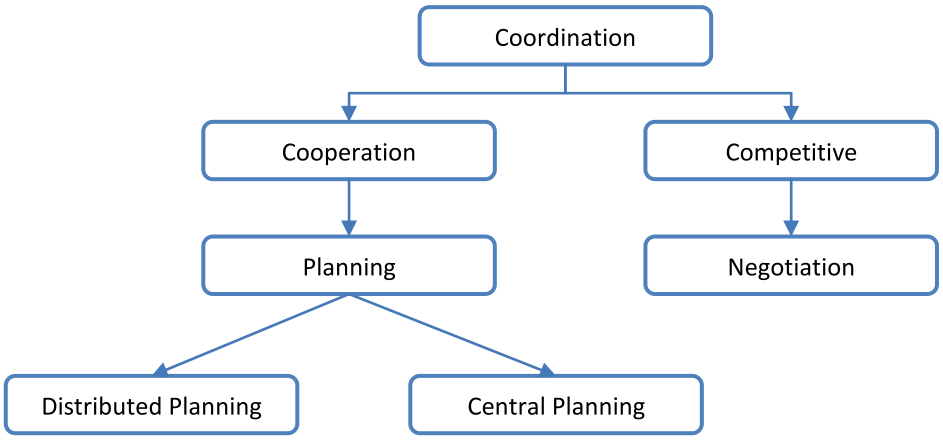


Figure 7 Coordination mechanisms

2.5.1. Agent Communication

In general - as in this work - message-based communication is under discussion, however, other kinds of communication are possible. Stigmergy [Bonabeau 1999] is a form of indirect coordination between agents by using the environment, e.g. in the form of tracks or pheromones. These approaches to communication are not considered in the present work. Communication in the agent field is different from procedural and object-oriented method calls by the (asynchronous) information exchange in the form of speech acts. The basics of speech act theory in use today go back to John Searle [Searle 1969]. According to this understanding, any speech act can be decomposed into other speech act components:

- The actual statement itself.
- The completed action by saying something.
- The consequence of the action at the receiver (action).

On the basis of speech act theory there are different agent communication languages (ACL-Agent Communication Language) for the standardizing of agent information exchange. These specify different aspects that are essential to a common communication capability. These include language, protocols and message formats. Examples of such agent communication languages are, for example FIPA-ACL (Foundation for Intelligent Physical Agents) [F. A. Board 2000] or the earlier KQML (Knowledge Query and Manipulation Language) [Miao Yu 2012]. These standards define the exchange of messages based on message structures and message protocols. Both standards support the optional use of ontology.

In this thesis the event-based messaging capability of the agent platform (JADE (1) in our case) has been used to enable the communication between agents supported by the agent-adapters for the communication with the agent environment.

2.5.2. Agent Cooperation

Cooperation achieves coordination with a common goal for all participants. Cooperative Distributed Problem Solving (CDPS) deals with the analysis of approaches to solve common problems in loosely coupled systems that cannot be solved by an agent [Durfee 1989]. Approaches to cooperative problem solving are for example task decomposition and distribution [Huhns 2012]

(1) Jade: Java Agent DEvelopment Framework -Telecom Italia

An example of cooperative problem solving is the so-called Contract Net Protocol [Davis 1988], [Smith 1988], [Weiss 2001]. It is primarily designed for the control of goods and services tenders. In an agent system, each agent can take the two roles of manager and contractor. The protocol proceeds in three steps. The manager writes an order (for example, multicast or broadcast), and

the contractor evaluates the tender and the response with one offer. From the set of offers the manager selects a suitable contractor. Generally, the Contract Net Protocol represents an efficient way for directly connected agents. This ability can facilitate cooperation in some cases and thus make the process of tendering efficient.

In this thesis the event-based messaging capability of the agent platform (JADE in our case) has been used to delegate tasks from one agent to another managed by the controlling and configuration module (CCM) and the messages and notification module (MNM) of the MAIP (multi-agent integration platform).

2.5.3. Agent Competition

Especially in the inter-organizational context it can be assumed that, in contrast to cooperation, the same aim and the associated intrinsic cooperation entities are mostly an exception. Once agents pursue different goals, agents can be in competition. A common form of interaction with different targets is negotiation [Davis 1988], [Weiss 2001]. First, the respective (conflicting) positions will be communicated and then an agreement by gradual concession will be achieved.

Negotiation mechanisms, whether environment-centred or agent-centred, should ideally [Weiss 2001] meet the following criteria:

- **Efficiency:** There should be no waste of resources at the unification process (e.g. excessive exchange of messages).
- **Stability:** No participant should have the incentive to deviate from reached agreements.
- **Simplicity:** The negotiation mechanism should have low technical requirements.
- **Autonomy:** Negotiation mechanisms should not require any central decision-instance.
- **Compatibility:** Participants should have a common negotiation interface.

The last point of the compatibility is an area of active research. Here the negotiation mechanism shall reveal the true preferences of the participants in terms of the mechanism design theory [Breskovic 2011], and a decision regarding the auction oriented intrinsic value shall result from. These auction participants should not behave strategically, but due to the design of the auction should reveal their true preferences.

In this thesis no need for the agent competition capability is indicated.

2.6. Agent Methodologies Discussion

As mentioned previously in Section 2.5, these methodologies are classified according to the different approaches as agent-based, object oriented-based, and knowledge engineering-based. A number of these methodologies have been recommended for agent-oriented development. Yet, it is not easy to select a specific one in order to employ or even to evaluate it. This is because they usually differ in their premises, covered phases, models, concepts and the supported multi-agent system properties. Therefore, a detailed discussion will be presented with respect to their advantages and difficulties.

2.6.1. Advantages of Agent Methodologies

Some of the existing agent-oriented methodologies are based on strong disciplined foundations; they possess the following advantages:

- Some methodologies take into account the idea of a society of agents or the idea of an organization that provides a coherent conceptual infrastructure for analysis and design of multi-agent systems.
- Some methodologies support both levels (micro- and macro-levels) to construct and develop agent systems.
- A few methodologies support model derivation, wherein some models can be derived directly from others.
- Some methodologies that are considered as an extension of the software engineering approach provide a solid base for the development of multi-agent systems.
- Some methodologies use well-known techniques, such as UML, which is particularly interesting. These techniques facilitate comprehension and communication between the various agents involved during software development [Sabas 2002].
- Some methodologies explicitly provide cooperation between agents and the concepts used to describe the type of control. Several others are less clearly specified.
- Some of these methodologies include an early requirements analysis phase, which assists the developers to understand an application by studying its organizational setting.
- Some methodologies nearly become complete methodologies for multi-agent systems. They treat most development phases, and they treat both inter-agent and intra-agent perspectives.
- The methodologies that constitute an extension of knowledge-based methods provide models that take into account the agents' internal states much better.
- Some methodologies provide a relatively elaborate support for reusable models, which is a valuable aspect for any methodology.

2.6.2. Difficulties of Agent Methodologies

The existing agent-oriented methodologies suffer from some difficulties which are the main reasons for a number of limitations that emerge. Those difficulties prevent agent-oriented methodologies from being utilized and practiced in a wide manner. These difficulties are:

- There is no existing agreement or accord on agent theory. Up until this point in time, no agent-oriented standards have been established and accepted as standard. No agent-oriented methodology will be able to spread unless the agent model is standardized. This standardization refers to which characteristics define an agent, which types of architecture are available for agents, which agent organizations are possible, and what types of interactions there are between agents, etc.
- None of these methodologies is widely used.
- All research that examined and compared properties of these methodologies has suggested that none of them is completely suitable for industrial development of MAS.
- There is no systematic approach to identify the components of MAS. Most current methodologies require the designers and developers to identify all agents of the system. Therefore, a designer’s experience is very important and is essential for producing a quality MAS. Designers should be trained beforehand to have the necessary skills for such projects.

- Although there are new languages for programming agent behaviour, there are no adequate development tools for representing agent structures. Languages tend to focus mainly on particular agent architectures.
- Selecting a suitable methodology to be followed for MAS development processes is not an easy task. Therefore, a precise methodology needs to be presented to guide the team of developers towards the achievement of objectives.
- Comparing methodologies is often difficult. This difficulty arises from the fact that it is not easy to evaluate them because they usually differ in their premises, covered phases, models, concepts and the supported multi-agent system properties.
- There is no agreement on what a methodology is and of what it should consist.

The BDI architecture has been used to realize the agent models in this thesis because:

- Test procedure and test cases can be realized by using a BDI agent:
  - Test procedures are the tasks to be performed by agents with respect to the defined test objectives (agent plan).
  - The test case steps (agent plan tasks) can be linked to the respective pass-fail-criteria (agent rules).
  - The test results report can be realized by using the event and messaging capability of the BDI architecture.
- In Chapter 3 of this thesis the concept of the realization of the new methodology has been discussed and demonstrated. This discussion and demonstration can be used as evidence for the BDI-architecture capability to be applied in the system V&V area.

2.7. Summary

In this chapter, the literature relating to agents, agent architectures, multi-agent systems, agent-oriented methodologies, agent development programming languages and platforms was reviewed. The field of agent-oriented methodologies was examined with the aim of establishing the characteristics of agent-based systems. An analysis of agent-oriented methodologies that gives a clearer picture of their application domain was presented including the advantages and difficulties they present.

"Agents are designed as a distributed problem solver in a dynamic environment. Their biggest promise is flexibility" [Kirn 2006]. Other properties that enabled their use in distributed systems are autonomy and social skills. Agents are service-oriented systems and grid environments are a promising approach to adaptivity through flexibility. This can be built for management structures within complex systems to enable bottom-up behaviour control and monitoring.

To achieve high flexibility, the focus is on the weighty, deliberative agents (e.g. BDI approach). This flexibility is achieved with some high computational and interaction complexity. Very simple reactive agents are suitable for pre-defined tasks that are processed together by means of simple rules (emergent). To achieve the prerequisites to act in the V&V domain, agent tasks have to be deterministic and real time capable. Therefore BDI agents have been extended in this thesis with supporting agent helpers and their management platform has been improved (see Chapter 3 and 4 for more details).

Here, however, by using simple reactive agents no flexibility can be expected regarding changes in the environment (This weakness is addressed in the work). "For larger or unknown number of agents is their interaction mechanisms unsuitable because of their poor scalability" [Weiss 2001]. The integration within distributed systems, such as Grids, requires further research. The use of Agent-based Computational Economics (ACE) is a promising approach to bottom-up analysis, modelling and coordination in massively distributed and complex systems. This work addresses some of Foster (see [Foster 2004]) listed outstanding issues relating to system management in the context of adaptation and optimization. For this purpose, an evolutionary approach is used to enable the agent capability for behaviour discovery as discussed and evaluated in Chapter 5 .

Although verification of software agents is established and well defined, for the validation and verification of complex systems using an agent based approach neither previous research works nor any existing industrial solution has been identified, so that this is the first time that this approach is designed and evaluated in the field of system validation and verification. Also in the last 15 years no basic research regarding the agent approach has been conducted. The current research handles the applicability of agents in a different research field but not the basic idea of what an agent is and how to model them..

In this thesis BDI based architecture has been used to realize the agent models of the MAIP (Multi-agent integration platform).

PART II
METHODOLOGY

PART TWO - METHODOLOGY

Part two of the thesis describes the main concept of MAIP as an improvement of the BDI based approach by extending the agent capability to act in a V&V domain. The first chapter in this part is a general description of the MAIP concept. The second chapter describes the required system layered abstraction.

The third chapter provides a new concept of knowledge representation required to represent the system requirements and their derived test requirements. The last chapter represents the agent learning by using a new context based learning approach, which provides agents with the capability to learn about their environment individually and adaptively.

CHAPTER THREE METHODICAL BASICS

3. Methodical Basics

This chapter presents the fundamental concept of the multi-agent integration platform (MAIP) framework for the system V&V. The main scope of this presentation is the methodology of using agent-based development for the V&V of complex systems. This agent based approach has been considered as a renewal of the established traditional V&V approach by using test procedures represented by implemented test cases (fixed coded or even configurable). The methodology is centred on three representations of the system to be validated or verified (and its components):

- The system component adaption as an interface to the agent environment,
- The agent environment and
- The control layer managing the data exchange between system component and agent environment.

The representations resulting from these three entities are the fundament of the MAIP-framework.

"It has been discovered that solving complex problems such as V&V of complex systems often requires combining multiple kinds of knowledge and methodologies, and the combined effect of multiple sources of knowledge can often identify the single most credible conclusion" [Durfee 1989].

When trying to solve a complex problem through an agent based approach, agents need to agree on a common knowledge representation and knowledge adaptation so that they can communicate. When searching for a suitable knowledge representation for the agent based approach to system validation and verification, the author of this dissertation considered a number of requirements derived from the state of the art and the last 15 years involvement as a system V&V engineer in the aircraft cabin testing area.

The first requirement was that the knowledge representation must support an incremental problem solving pattern. Under the MAIP framework architecture, overall solutions are constructed step by step at multiple levels of abstraction (layered abstraction). As cooperation takes place at each level of abstraction, the knowledge representation must support the agents' communication in the entire process. The second requirement was that the knowledge representation must allow agents to refer to, among many resources, system component interfaces, behaviours (function), performance, states, and system environment. The third requirement was that the knowledge representation must support inference. This is because when an agent reads information on the MAIP, it might want to derive new knowledge based on the MAIP content and its internal knowledge. The fourth requirement was that the knowledge representation must be flexible enough to allow information, especially partial solutions, to be considered as it becomes available. This is because in the MAIP architecture the running order of the agents and the knowledge they contribute towards the final answers are unknown in advance.

With respect to the construction of software systems, it is essential for the developers to have a profound understanding of the nature of the system. Modelling is the essential means for the understanding of complex systems. Petri-Nets have been established in recent years for the agent modelling; therefore, in this dissertation, Petri-Nets have been chosen for the agent modelling. The content of the MAIP has been defined as "info" field, and the "mode" field identifies whether the information record (data set) is a routine message or a command. The foremost reason for choosing Petri-Nets was that the Petri-Nets allow an agent model to be stated as a set of explicit statements (facts), and they have the ability to refer to low-level contents, such as state, behaviour and function, as well as high-level contents, such as agent goals and plans, at the same time. The second reason was that Petri-Nets allow the representation of concurrent processes which was essential for the modelling and representation of agent and multi-agent systems [Purvis 1996].

The research focus of this thesis has been on how to provide agent modelling and implementation methodology for agent based V&V in permanently changing heterogeneous environments. The research focus has been divided into two main conceptual parts:

- **Layered Abstraction:**
  We analysed the current testing process (see Section 3.3) and formulated for the testing tasks that are to be improved by the new research approach, i.e. how layered abstraction including its knowledge representation may help to improve these tasks. The first conceptual part deals with abstract modelling of systems, their components and their functions in different levels so that agent-based models may be applied thereon. Three layers have been defined in this thesis:

  - Component level layer (equipment layer)
  - System level layer (multi-equipment layer)
  - Multi system layer (functional layer)

- **Context Based learning:**
  In the second part the concept of agent adaptivity is presented. Here, a new approach of context based learning is applied, whereby the agents can observe the environment and its changes in different contexts.

Based on the agent modelling aspect described in Section 2.2 (Agent modelling with Petri-Nets) in this chapter the conceptual and methodical foundations are represented as well as a description of the concept of the present thesis. This new concept has been based on a definition of generic agent model architecture (see Figure 8) that has been specified with respect to the testing tasks to be improved (see Section 3.3). In the next chapter of Part II of this document the two conceptual parts are described in detail.

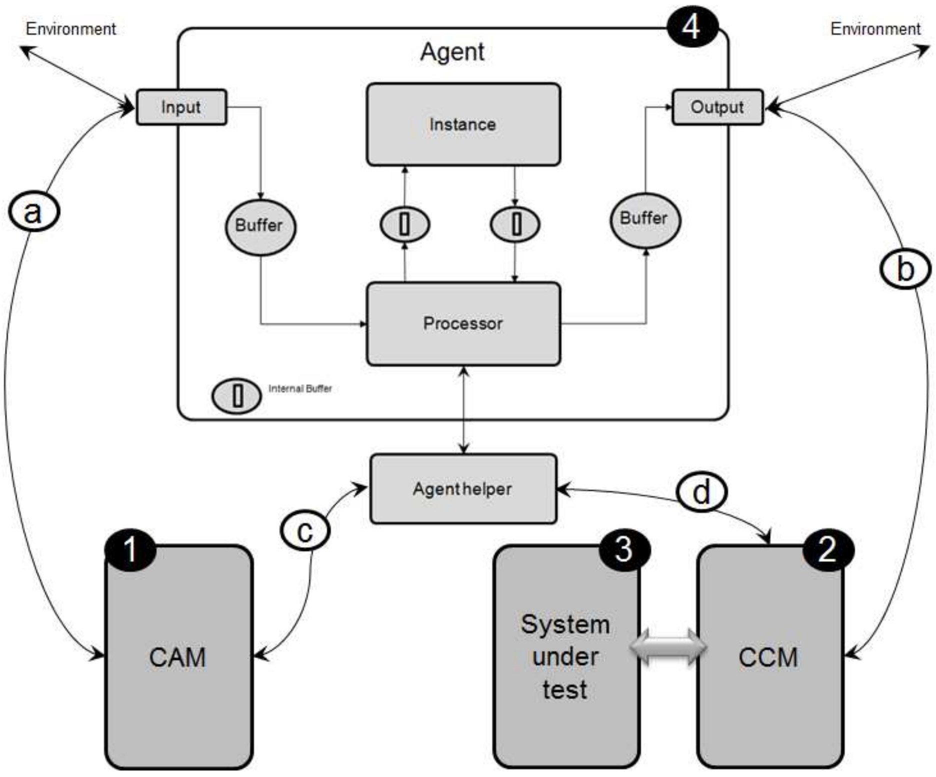


Figure 8 Generic agent model architecture

- **CAM:** Component abstraction model that is used for the representation of the knowledge about the system under test on each abstraction layer and its behaviours. CAM provides agents with all required data to define their initial instance.

- **CCM:** Component control model is an access model that is used for the adaption of the system under test or its hosting test engine.
- **Agent:** A generic agent model as defined in Section 2.2.
- **Agent helper:** Specific software routines to support agents with functionality outside of their messaging context (e.g. writing to file system during CAM updating process, realtime interaction with the test engine during test execution).

3.1. Introduction

The contribution of the present thesis is embedded in more complex questions of the agent oriented approach in the domain of V&V of complex systems. Among them are the questions: What is a real complex system and what is a model of the system? What is the role of verbal logic when implementing system models? The testing model is installed as a formal specification of the workable (real) system by the agent based test. A discussion of this method requires analysing the existing testing process and the methodology used for test case detection intended for the verification of software. Software is created on the basis of requirements. In this context system models, requirements and the term verification play a role.

3.2. Research Methods

The next sections present research methods that can be applied in software development research and how they are applied in the new approach discussed in this thesis.

Software development is a multi-disciplinary and complex research field stretching from technical research like development of various tools, through modelling and languages, to more non-technical research like managing people and changing processes and organisation etc. Software development cannot be performed in the same way as manufacturing, because it involves human-intensive and creative activities. The goal of research within software engineering is to make tools, methods, and models enabling software to be produced more effectively, with better quality, on time, and by expending fewer resources. In this thesis research methods are required to investigate and examine how the MAIP entities and their representations can be modelled and integrated into an appropriate multi-agent environment to ensure the capabilities of performing V&V tasks within the system testing domain. Basili [Basili 1992] has identified three main research approaches that are commonly used for doing experiments in software engineering:

- **The engineering method:** By using the engineering experimental method, engineers build and test a system according to a hypothesis. Based upon the result of the test, they improve the solution until it requires no further improvement. The engineering method is typically used to find better methods for structuring large systems, and software engineering here is viewed as a creative task not to be controlled by anything else than necessary restrictions on the resulting product.

- **The empirical method:** A statistical method is proposed as a means to validate a given hypothesis. Unlike the analytical method, there may not be a formal model or theory describing the hypothesis. Data is collected to falsify the hypothesis. The empirical method can be applied to new technology to determine if this new technology is better or worse than the existing technology for producing software effectively.

- **The mathematical method:** The mathematical method is based on mathematical and formal methods for doing experiments. A formal theory is developed and results derived from that theory can be compared with empirical observations. The mathematical method is usually used to find better formal methods and languages, where software development is viewed as a mathematical transformation process.

"The engineering method and the empirical method can be seen as variations of the scientific method" [Basili 1992]. Note also that a combination of the research methods can be used.

In this thesis the emphasis has been on the use of the engineering method, but we have also conducted some limited empirical studies. In the next sections of this chapter, we described the research methods used in this thesis in more detail.

3.3. System validation and verification

Based on the V-Model (see Section 2.1), on the engineering researched method and with respect to the industrial standard and especially to the test process defined and used or to the testing of aircraft systems (e.g. AIRBUS), a validation or verification process (testing process) generally consists of the following main activities:

- **Define test plan:** Generally a management planning document that shows how the testing approach will be done, which testing instance will do it, which items will be tested, how long it will take and what test coverage will be expected. In addition, the input data validation will be performed to ensure that all data required for testing are available and valid.
- **Test design**: Describes test conditions, the system under test configuration, the expected results as well as the test pass criteria.. Test design describes also how to run each test, including any set-up preconditions and the steps that need to be followed.
- **Test configuration:** Is a result of test design tasks wherein generic test cases will be specified and linked to a valid test configuration (e.g. system standard and interface specification)
- **Define test cases**: With respect to the given test design the test cases are to be defined. This definition represents the test specification in a generic way to ensure that the test cases can be run in different test configurations.
- **Define test procedure:** With respect to a given test design the steps to be followed will be formally defined.
- **Implement test cases:** The generation of test codes (test script) that can be run on the specific test engine.
- **Define test execution:** Depending on the defined test procedure and with respect to a linked test configuration the test execution will be defined including test monitoring and control instances. These instances are responsible for the recording of which test cases were run, who ran them in what order, and whether each test passed or failed, including interaction with the test engine with respect to failure injection and script manipulation.
- **Evaluate and report test:** Describes, for any test that failed, the actual versus the expected result, and other information intended to throw light on why a test has failed.

Figure 9 shows the main testing activities. In the next sections the current limitations and problems of each testing task have been analysed and respectively our research challenges have been formulated by the identification of the requirements which need to be fulfilled in order to improve the whole testing process.

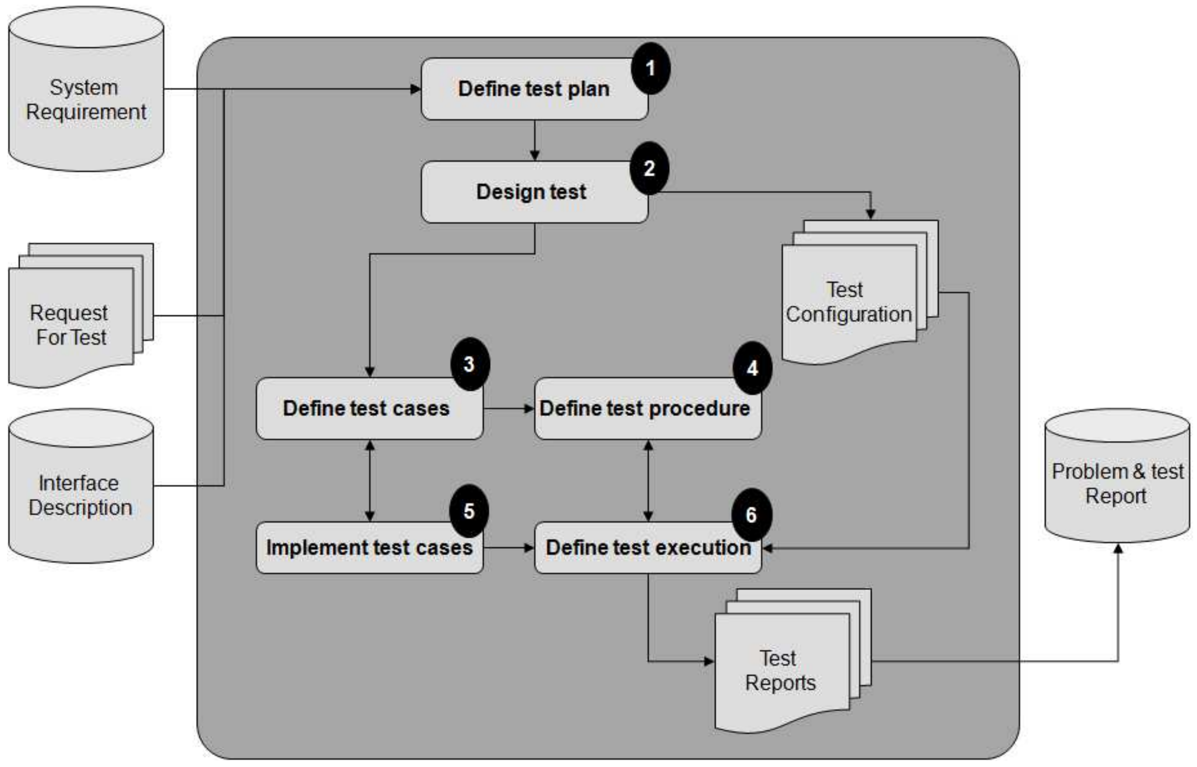


Figure 9 Testing tasks

3.3.1. Define test plan

Based on the IEEE 829 (Standard for Software Test Documentation), a test management system including a tool chain has been developed (in use since 2003) in the scope of a research project so that there is no need to improve this system within this thesis.

A test plan activity is an approach that will be used to validate, verify and ensure that a product or system under test meets its design specifications and other requirements. A test plan is usually prepared by test designers wherein an agreement with the system designer or test requester shall be made to determine the test scope and all features that are not to be tested within the defined scope.

Depending on features of the organization to which the test plan applies and especially for AIRBUS, a test plan should include one or more of the following definitions:

- **Validation of Input Data:** This task is to be performed during the development or approval milestone of the system under test to ensure that the input data are compliant to the test design. This task is typically done manually.
- **Test Scope Definition:** Defines the scope of the system functions to be tested including the parameter range to be used.
- **Test Item/Test Feature Definition:** Identifies the system components or system features to be tested.
- **Test Approach Definition:** Defines the test approach to be used for this testing task.
- **Test Environment Definition:** Defines the required system under test environment, e.g. other connected system, system components or simulation.
- **Test Activity Schedule Definition:** Defines the test schedule for this testing task.

Validation of input data

The most import technical part for using an agent based approach of the test plan activity is the validation of the input data. The following data types are essentially important to enable a test plan activity; the test build-up and the test design cannot be realized without them:

- System interfaces (required for the interaction with the system to be tested)
- System installation (required for the representative instrumentation)
- Delivery plan (required to plan the test resources)
- System requirements (required to define the pass / fail criteria).

In industry and specifically in aircraft manufacturing, system interfaces, installation and delivery plans are established well-defined processes which are not to be improved in our current research. System requirement management, formal requirement description, and requirement traceability were also challenges which have been handled in different research works and are not a part of this thesis:

- Care Marcos (Formal management using excel automation): The results of this research project could not fulfill all requirements because an equitable data model was missing.
- RTM (Requirement Traceability Manager): The results of this research project could fulfill all requirements but it was too complex to be handled by test engineers with typical competencies.
- DOORS (Dynamic Object Oriented Requirements System): Well-defined existing modelling tool which fulfilled all requirements and has been in use since 2007.

3.3.2. Define test design

The test design definition describes the structured way of creating test cases that can be implemented to validate or verify the system under test. A test design can be agreed and accepted by the system designer when the defined techniques that exploit the structure of the system under test are minimised to a number of test cases required to achieve a given level of coverage.

A test design definition activity includes the following definitions:

- **Define test configuration:** the definition of the environment configuration that is required to perform all defined test cases (e.g. the configuration of test engines, simulation, monitoring panels and test script). This task is usually done manually and sometimes supported by configuration scripts.
- **Define test requirements:** the conversion of system requirements to formally described requirements (test requirements) to ensure their testability.
- **Define pass/fail criteria:** Linking or adding the pass/fail criteria to each test requirement defined in the previous task.

This activity is time consuming and highly complex so that the agent-based approach can be used to improve it. The research question is how to model an agent and which entities are required to be capable of supporting these tasks by using the generic agent model architecture (see Figure 8).

Generally, in order to be able to manage testing activities we have distinguished between data, processing and required system under test adaption. In the generic agent model the CAM is intended to represent the data (knowledge), the agent to process data and the CCM to adapt the system under test.

For the CAM the following data entities and their semantics have been identified to be represented to be able to perform test design definition activity and its tasks (see Appendix C):

- Interface information (system under test and test engine interfaces)
- Parameter list (system under test and test engine parameters)

- Operation list (system under test functions)

For CCM an interface to the test system (test engine that hosts the system under test) has been developed based on the given test engine remote API (application programming interface).

For the agent and agent helper the following requirements have been addressed to be considered when modelling agents:

- Based on the system under test requirements and functions test requirements and their pass/ fail criteria shall be created
- With respect to the CAM data test configuration shall be created.
- Test configuration shall be adaptive to environment changes during test run time (modified parameter or modified operation behaviours).

3.3.3. Define test case

Usually, this task covers state-of-the art techniques for the generation of test specification and its specific test engine-oriented implementation. Some system testing tasks, such as extensive low-level interface regression testing, can be time-consuming when done manually. In addition, a manual approach may not be effective in finding certain types of failures. Test automation offers a possibility to effectively perform these types of testing, and agent test approaches can increase the flexibility and the coverage of such tests. For this reason different research projects have been handled in test automation wherein some of them gave encouraging results but were not industrialised because of their complexity.

System modelling by using HybridUML

The first research project was the modelling of a system using HybridUML (a realtime extension for UML) to enable the capabilities of automated generation of test cases and test controls. This project was founded by Airbus-Germany in cooperation with the University of Bremen (2004-2007). The problem with this approach has been, it is highly complex, it's time-costly and it requires fundamental changes in the system development process.

System modelling by using SysML

In this project the use of SysML for the system and test modelling has been researched in cooperation with the University of Hamburg (2009-2011). The problems with this approach have been the complex configuration and insufficient robustness against evolutionary development life cycle.

The test case is a specific executable test that validates or verifies all test requirements including inputs and outputs of a system and then provides a results description of the test case steps that should be performed. Test case steps described in a test case include all details even if they are assumed to be given knowledge specified by the system designer.

A test case definition activity includes the following tasks:

- **Test Case Definition:** Definition of objective and context of the test case in a textual form (meta information of the test case)
- **Test Case Run Initialisation:** Definition of the initial configuration of the test case in a formal way (e.g. input and output parameters)
- **Test Case linking with Signal Scope:** Identification of the signal scope (real physical system parameter) required for its execution.
- **Test Case linking with Test Configuration:** Linking of the test case to a valid configuration defined within the test design definition activity.

For the modelling of the test case definition tasks the same CAM as well as CCM can be used which are modelled during test design activity because no additional data have been identified for these tasks.

But for the agent and agent helper the following additional requirements have been identified that are to be considered when modelling agents:

- Based on the CAM the test specification and its test cases shall be defined including their initial conditions.
- Test cases shall be linked to their signals (parameters) and respectively related signal scope. The definition of the signal scope is important to reduce the number of signals that are irrelevant and not affected by this test case.
- Test cases shall be linked to test configurations to ensure their applicability and validity within a specific configuration.

3.3.4. Define test procedure

The test procedure definition is a specific grouping of test cases within a specific configuration and test setup that control the execution of the implemented test cases. This testing activity is complex rather than time-consuming and needs to be improved. The reason for this complexity is the appropriate selection of test cases that are valid, useful and correctly configured or initialised.

A test procedure definition activity includes the following tasks:

- **Test Case Selection:** Selection of test cases that can be logically grouped with respect to their configuration, initial setup and the function to be tested.
- **Test Procedure Initialisation:** Initialisation of the test cases linked to it including the definition of the point of observation and control. In addition, the signal scope of all linked test cases is to be determined implicitly.

Also here, only agents are affected when defining test procedure, and for this activity the following requirements have been identified:

- Smart selection of test cases with respect to the given test configuration and to the sequence of test case execution.
- Smart conflict management of the initialisation of the test cases with respect to the sequence in which they are to be executed.

3.3.5. Implement test case

The test case implementation is the activity to be performed to convert the defined test case in an executable form that can then be executed during the test execution activity. There are several ways to perform this activity:

- If the defined test case is formally described, the test case implementation will be done automatically supported by certain tools or scripts.
- If the test case is not formally defined, a test instruction document is to be created so that the test can then be performed manually.
- If the test case is to be automatically executed but no formal definition is given, the test case is to be implemented and validated manually.

In this thesis test cases are specified by agents and formally defined so that the implementation can be performed automatically.

Here the CCM and agents are affected and the following requirements have been identified:

- CCM shall support target cross compilation including code injection. Code injection is essentially important to ensure the capability that enables agents to react to environment changes in order to be adaptive to them. This adaptivity could lead to a modification of test cases and their steps as well as of the test procedure.
- Agent models shall be capable of validating implemented test cases with respect to the following aspects:

  - Is the test case executable on the target host?
  - Are all required signals correctly created and linked? If not, then they are to be created and linked by agents during initialisation and every time a code injection has been performed.

3.3.6. Define test execution

In the validation and verification of complex systems there are two aspects to be handled when defining test execution. On the one hand, we need to execute tests while they are developed to check them for proper functionality. On the other hand, there is a need to run tests periodically to ensure the stability of the system under test by using different configurations, without redefining it every time a test configuration needs to be adapted.

A test execution definition activity includes the following tasks:

- **Linking of the test procedure to be executed:** For each test procedure and its linked test cases a test execution entity is to be defined to regulate the test run on a specific test engine host including the initialisation of the test procedure, the selection of the appropriate test case parameters, the instantiation of monitoring objects (e.g. panels) and ensuring the right test flow.
- **Generation of test case result objects**: Result objects are to be created to observe test actions and to record them if required. The test result objects have to have the same granularity as their linked test cases, so that each test case or test case step result can be recorded when a test passed/failed criteria is defined.
- **Execution of tests:** Test runs including interaction capability during test execution.
- **Evaluation and reporting of test results:** After the test execution the test results are to be analysed and evaluated to support further reporting tasks and evidence when the test has been successfully executed, and problem report system failures are detected and identified.

Test execution in a complex system area is linked with system requirements and configured by interface description. The definition of a test execution in a complex environment such as aircraft testing area usually suffers from efforts required to manage the configuration of the control, monitor and evaluation items, which are closely linked to the system requirements and interface specification. The reasons for this difficulty are mostly immaturely defined system requirements and not fully defined or incorrectly defined interfaces. Classical modelling approaches are based on fully defined and mature items, so that a new approach was required to handle these issues.

With respect to this issue only agent modelling is affected, and the following requirements have been identified to handle it:

- With respect to the pass/fail criteria defined by the test requirements and their linked test cases agent models shall be able to create the test result objects that host the results of tests.
- Based on the test results the test monitoring and evaluation task shall be handled by agents; for that, the following capabilities are to be implemented:
  - Real-time evaluation of test results.
  - Provide notification messaging or reporting to notify affected entities to enable them to handle failure events if required. The emergence of failures could lead to changes in the test configuration, modification of executed test case steps, aborting of test execution, adaptation of test specification and modification of knowledge hosted by the CAM.

3.4. Improvement Objectives

By analysing the requirements identified in the previous section the following main objectives have been formulated to be achieved by using agent-based approach, in order to achieve the main goal of this thesis: *“that the only task to be done by the test engineer is to define an appropriate CAM”.*

- Enabling test specification by limited knowledge about systems to be tested. Given that CAM has to provide agents with the knowledge about the system to be tested, it is essential to enable test specification tasks even if the specification of the system to be tested has not been completed. For this challenge a discovery approach has been developed that is supported by agent models, wherein the following aspects are to be considered which are supported by the context-based learning approach:

    - Inputs are unpredictable.
    - Systems or system components are not specified in detail.
    - Multi-system functions are roughly specified.
    - Development testing in an evolutionary system implementation and manufacturing process (sometimes before system design has been completed).

- Enabling test processes in a highly dynamic and rapidly changing environment, especially when the system behaviour deviates from the specified behaviour. For this challenge the agent adaptivity approach has been developed that is supported by agent models, wherein the following aspects are to be considered:

    - Handling of system clusters which consist of complex and dynamic systems.
    - Complex interaction between systems within system clusters.
    - Handling inter-cluster functions.

- Handling a huge number of requirements and interfaces which have permanently increased during the recent years; for that layered abstraction approach has been used. The following aspects have been considered:

    - Handling different levels and contexts of requirements.
    - Handling different parameters’ visibility/accessibility scopes.
    - Enabling early interface validation capability.

3.5. Agent Based Testing

The term “agent based testing” summarizes a plurality of methods and technologies. The following sections illustrate at first the conceptual foundations and thereafter show a classification of the differing methods in order to handle the results of previous researchers (see Section 3.3).

<u>Test Model</u>

A test model comprises a description model of the system to be tested and an explanatory model of the test environment. For the present thesis the test models have been defined as abstract descriptions of the properties to be tested of the test object and of the test environment. A test model is, therefore, understood as an abstract behaviour model which includes the functional aspects required for the test. Modelling of application settings or test sequences is not to be included and is reserved for agents only. The decision with respect to modelling formalism of the test model strongly depends on the character of the system to be modelled. The models which describe system features and test environment are called CAM (Component Abstraction Model, see Section 3.7) in this thesis. The test model has been developed to handle the weakness of the classical test case implementation tasks (see Section 3.3)

Agent Test vs. Agent Based test

The objective of the agent test is to verify an agent model. Thereby, the requirements of the model are formally described and are verified against the model. On the basis of a model checker it can be determined if a model fulfils a property [Bordini 2011]. Modelling of agent tests is not part of this thesis. The agent based test initiates test cases on the basis of a model (in our case test model represented by CAM), for a test object which is different from the model. In this case, the model provides information with respect to properties to be tested of the test object, or activities due to which, the properties to be tested can be observed. For the time being it is not determined in which forms of a model these properties or activities have been stored. The general definition of a “model” allows any abstract description of the test object, even natural-language specifications. In order to allow a clearer delimitation from the classical procedures of test case detection, solely formal and semi-formal notations were considered at this point. As a rule, in the agent based test, test cases are derived automatically from a test case generator. In this thesis agents have assumed the task of specifying and generating tests on the basis of CAM information.

Agent Based Test Process

The agent-based test process can be seen in Figure 10, which is based on the test model generated within the test specification process. With respect to the testing task (see Figure 9) the test specification has been represented by the test model, which is the enabler of the agent test process (see Figure 10 (1)). The function selector (see Figure 10 (2)) is an option to select a subset of the functionality represented by the CAM when required. The agent model (see Figure 10 (3)) is responsible for the transformation of the test specification into implementable format and the linking of agents with the knowledge base represented by the CAM. The test case generation and compilation (see Figure 10 (4-5)) are defined to realize the new test generation task (see Figure 9) improved by agent-based approach. Test execution and evaluation (see Figure 10 (6-7)) are defined to realize the test execution task also improved by agent-based approach (see Figure 9)

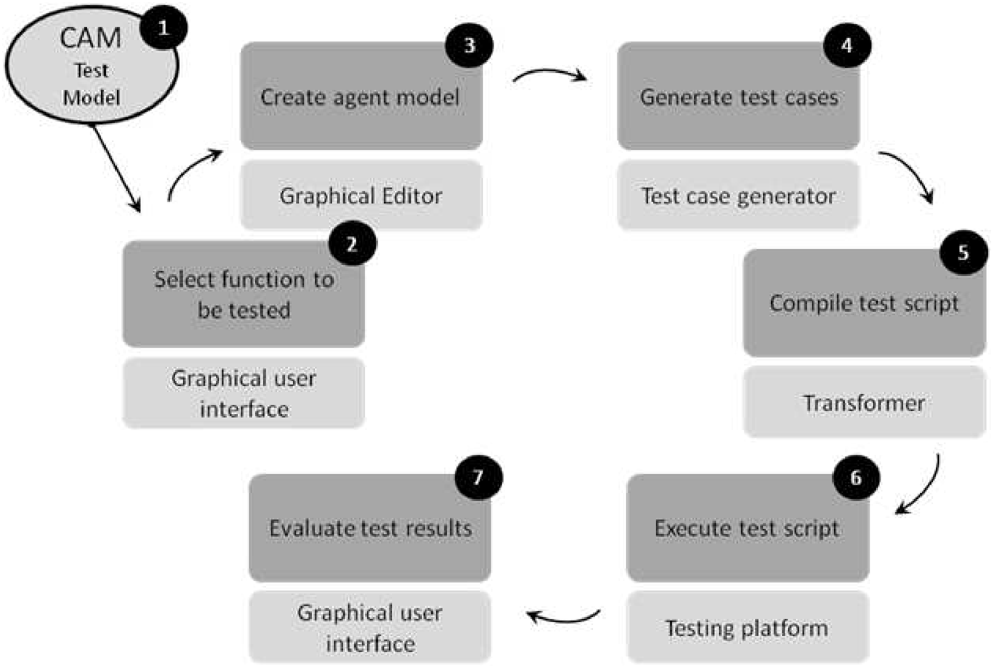


Figure 10 Agent based testing process

A function in this thesis is the interaction between several systems in order to realise an overall functionality. The agent based test derives test cases on the basis of a test selection, depending on the systems which are part of the function; these systems are represented by several system-oriented CAMs. The system under test (SUT) in this case comprises the function in the first place and the corresponding systems in the second place.

During the first step of the agent-based test process the test engineer creates several CAMs in order to describe the systems and their functions.

In the second step the tester defines strategies with respect to the agent configuration. Test strategies can be differently motivated. The necessity to verify the basic functionality of a part system (component) within one hour or to test separately specific security requirements can be test strategies. These strategies will be provided to the test case generator in a formalised manner. Types of possible test selection criteria will be presented further in this section.

On the basis of these test section strategies the test case generator (one or more agents) automatically creates the desired test cases. A test case corresponds to a sequence of state transitions from the point of view of the test case generator. A test case includes system conditions as well as expected system behaviour. The different systems used in test case generation will be considered in more detail in this section.

In the literature, variants are well-known for the model based test; these variants can be adopted for the agent oriented test, since this approach has been established for complex systems and has rarely been researched for agent based tests. The classification of Utting [Utting 2006] divides the different processes on the basis of their test selection criteria, notations, test case generation algorithms and general differences.

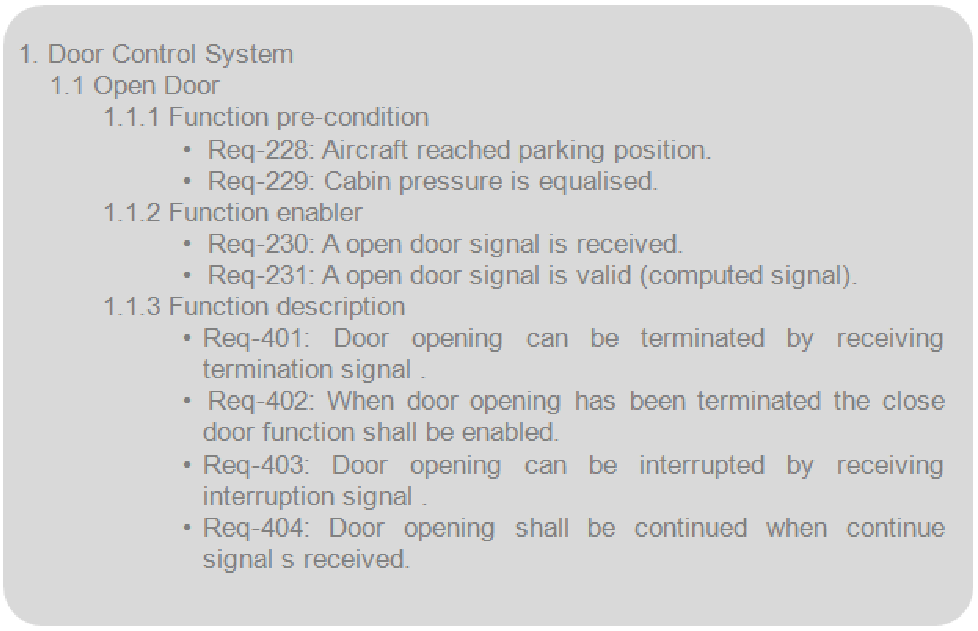
3.6. From Requirements to Agent Model

To be able to communicate descriptively the concept of the thesis within the scope of an application domain, a specific software development process has been selected in the present paragraph. Since this thesis has been accompanied by cooperation with the company Airbus with respect to a project dealing with the improvement of the aircraft cabin (“Agent Based Testing ABT”), the decision for the development of test systems has been taken in the aircraft industry.

At this point I leave it to the creativity of the reader to perform a transition of the presented generic ideas also to other software/hardware development techniques and domains. The time behaviour of distributed systems in an aircraft is a significant component of agent modelling. An agent test model must allow to represent time appropriately.

Frequently, severe real-time requirements to a system exist. The modelling of the system “Aircraft Door Management Control” represented in the following allows the use of so-called timers implemented as variables $t \in R+$ on positive, real numbers. The possibility to use these timers for the modelling of real-time requirements has been secondary only with respect to the approach of the present thesis. Solely the possibility for the modelling of time requirements is part of further considerations.

Figure 11 shows an excerpt from a system requirement specification of the Open Door Control functionality of an aircraft. It includes natural language system requirements. Each single requirement can be recognized by the preceding Req-xxx identifier.



1. Door Control System
    1.1 Open Door
        1.1.1 Function pre-condition
        - Req-228: Aircraft reached parking position.
        - Req-229: Cabin pressure is equalised.

        1.1.2 Function enabler
        - Req-230: A open door signal is received.
        - Req-231: A open door signal is valid (computed signal).

        1.1.3 Function description
        - Req-401: Door opening can be terminated by receiving termination signal .
        - Req-402: When door opening has been terminated the close door function shall be enabled.
        - Req-403: Door opening can be interrupted by receiving interruption signal .
        - Req-404: Door opening shall be continued when continue signal s received.

Figure 11 Door Control Requirement Extract

Requirements are usually defined positively, i.e. it is described when and whereby a function is activated. It is implicitly assumed how the system shall act if activation does not occur. The requirements describe when the aircraft door shall be opened. This occurs as soon as the button “open door” is pressed by the cabin crew. It is not described how the aircraft door shall act if no signal “open door” is received. In this case, it is implicitly assumed that the aircraft door does not open. Formal system specification means that this negative view with respect to the system, contrary to the natural language specification, is automatically enforced.

In an aircraft the interaction of the system components to be tested is a reactive system which includes parallel, usually unscheduled, processes in real-time. The system integration test focuses on the system state at a defined point of time as well as on the system behaviour over time, i.e. on the change of the system caused by a state change. A state change can be described on the basis of the pre-state and the next state of a system [Clarke 2000]. Since reactive systems influence their environment or are influenced by their environment, monitoring of state changes is of major importance. Not only input and expected output are important, but also the active chain, i.e. the observable state changes during function performance.

3.6.1. Timed Abstract Machine (TAM)

Consideration of the characteristics, state charts, time abstract machine of [Alur 1998] and SysML results that state machine serve as a means for system modelling. The agent model as a specification of system behaviour is completed by test information, e. g. tolerances of signal transfer allowed during testing, and by components of the test environment which are especially required during testing. The state charts are the most important part of the CAM. The specification of a CAM is completed by the formal definition of the required functions, their restrictions and signal flow (see Section 3.7).

The approach of the agent test model creation here is exemplary only and can serve as a basis for other system modelling methods. During subsequent test case generation the state charts are imported and processed.

All state charts run in parallel and communicate via shared global variables, wherein the principle for variable declarations “as local as possible, as global as necessary” is valid. The varying system boundaries and system abstraction levels help to facilitate the structure of the visual range of variables. They allow for locally declaring variables for a state chart. The timer variables usually employed are declared on the state chart level. In Figure 12 timer variables are indicated by t. In the state machine represented, the requirement with respect to the transition from one state to another is indicated in squared brackets. Behind the slash of a transition the actions which are performed when the transition is taken are indicated. If actions shall be performed when a state is entered, this is recorded behind the word “entry”. If actions are to be repeated, the term “do” is put in front.

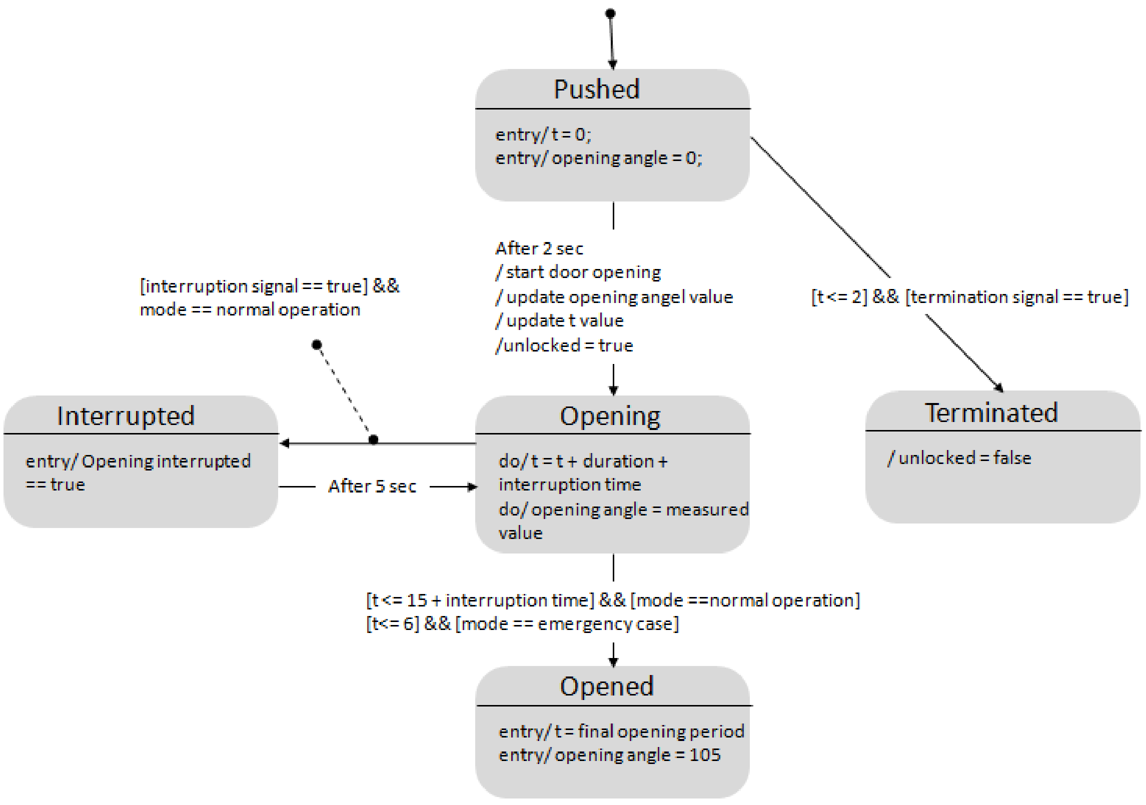

Pushed
entry/ t = 0;
entry/ opening angle = 0;
After 2 sec
/ start door opening
/ update opening angel value
/ update t value
/unlocked = true
[t <= 2] && [termination signal == true]
Terminated
/ unlocked = false
[interruption signal == true] &&
mode == normal operation
Interrupted
entry/ Opening interrupted == true
After 5 sec
Opening
do/ t = t + duration + interruption time
do/ opening angle = measured value
[t <= 15 + interruption time] && [mode ==normal operation]
[t<= 6] && [mode == emergency case]
Opened
entry/ t = final opening period
entry/ opening angle = 105

Figure 12 Simplified Door Control Service State Machine

The state machine “Open Door“ is among the so-called environment machines of the agent test model. Separation of machines which model the system environment from the models of the behaviour to be tested reflect the separation of stimuli and shall-behaviour. Machines of the environment describe acceptable system stimulation as well as faulty operations. Usually these machines are not predictable, since system operation is not predictable. However, this is not always the case. If during testing simulation models are installed, as e. g. a model for simulation of rain, the behaviour of the rain model is modelled within the test system environment so that a shall-behaviour calculation can be made. The circle between environment and system must also remain closed within an agent test model. These models are predictable, since for each system state a definite next state can be calculated. Apart from these models, all machines of the system behaviour are also predictable. The behaviour of systems in the aircraft as well as in the automobile and railway industry must be predictable at any time.

The environment machines curtail the allowed stimuli variations during test case generation. The opening angle in Figure 12 may be changed only within the time space preset on the transitions. By changing the angle and the liberty to choose the duration of the change and of the value assignment of the variable as far as possible arbitrarily, a test case generator can stimulate a state change in the system and can, thus, achieve desired structural model coverages, like the coverage of all transitions. The challenge is to choose the value assignment of the stimuli variables and their time process such that maximum model coverage will be achieved. Since the test case generator is not entirely arbitrarily due to the environment machines, this task is additionally complicated. Completely free data coverage, however, would lead to test cases which on the one hand could be unrealistic and on the other hand could be faulty (by e.g. faulty changing of the opening angle (> 105^{o}) which is in contrast to the mechanical conditions). The behaviour of environment models must be emulated. How this can be made efficient, is presently still part of research. The requirements of system stimulation and its limits is, thus, an important point in test case generation. In this thesis the timed abstract machine has been modified (consolidated with the Computation Sequence Chart (CSC)) and has then been installed to transfer the system states of the system specification in a formal description. The formal state modelling has then been taken as logical terms for the state description into the CAM specification (see Section 3.7). From the logical terms test cases have been generated by using agents (see Section 3.8).

3.6.2. Timed Computation (TC)

In this thesis a new formalism "Timed Computation (TC)" has been introduced which allows to combine functional requirements and agent test models which results in an extension of the Computation Sequence Chart (CSC) of Grosch (see [Grosch 2011]). This extension was necessary to enable combination of the TAMs with the CAM-Operation (see Section 3.7.2).

A TC consists of elements of existing state transition graphs and describes a group of system computations which reflect the specified behaviour of a single requirement. The next paragraph illustrates the graphical notation, the syntax and the semantics of a TC. A TC shall be consulted together with the agent test model for test case generation and for comparison with existing test cases. For this purpose the TC requires a logical representation.

In the case of the agent model based test, system behaviour is represented in state transition graphs. The state transition graph corresponds to the expected system behaviour. The easiest way to identify elementary system behaviour as it is described in an elementary naturally linguistic requirement is to use a state transition graph by choosing a sub-path via the graph which fulfils the requirement best.

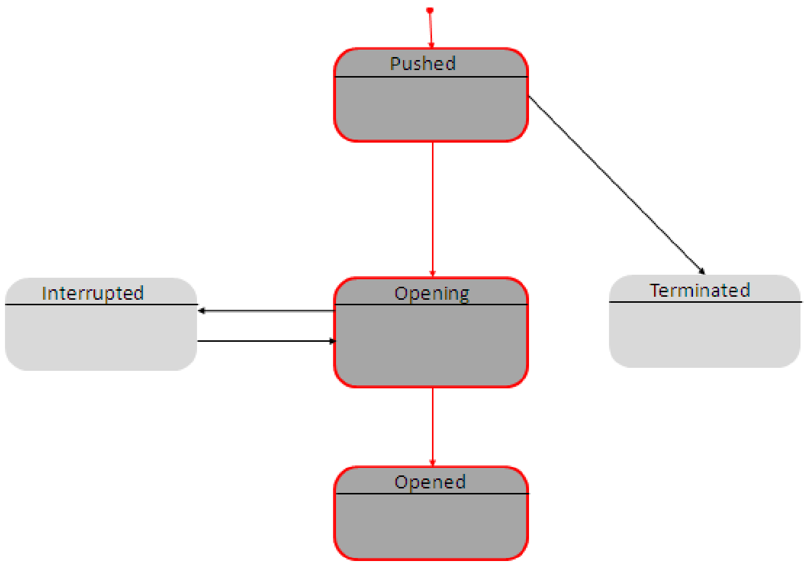


Figure 13 Path of computation tree

From the point of view of the state transition graphs which represent the system behaviour, a requirement also corresponds to one or more paths through the represented computation tree of the graph. Figure 13 shows a computation tree. The path marked coloured through the tree may on the one hand be understood as specific system behaviour and on the other hand fulfils chosen system properties. The computation tree describes all system computations which can be performed from the starting point. A single requirement describes a finite quantity of these computations. Thus, all system requirements, regardless of system behaviour descriptions or system properties, can be described by means of the specification of a quantity of paths through the computation tree. A system property then must apply to any system computation which is in the computation tree of the state transition graph. It is clear that we can perform the requirements tracing on the basis of this path selection. Definition of desired paths by a computation tree can be performed by a system designer. Single system behaviour can be easily represented via a chosen path by the system model, as can be seen in Figure 13.

The coloured paths through the machine in Figure 13 are depending on data assignment, corresponding to alternative paths through the computation tree. The path describes the desired operation sequence based on the involved system component function to be performed.

For the definition of the syntax and the semantics of a TC the following assumption shall be given. An agent test model is a composition of Timed Automata **A** = $A_1 \parallel A_2 \parallel \ldots \parallel A_n$, where A_x is representing Timed Automata corresponding to the system component time constraints.

Definition (Timed Automaton, [Clarke 2000])

Formally, a timed automaton is a tuple $A = (Q,\Sigma,C,E,q_0)$ that consists of the following components:

- Q is a finite set. The elements of Q are called the states of A.
- Σ is a finite set called the alphabet or actions of A.
- C is a finite set called the clocks of A.
- $E \subseteq Q\times\Sigma\times B(C)\times P(C)\times Q$ is a set of edges, called transitions of A, where
    - B(C) is the set of boolean clock constraints involving clocks from C, and
    - P(C) is the power set of C.
- q_0 is an element of Q, called the initial state.
- An edge (q,a,g,r,q') from E is a transition from state q to q' with action a, guard (constraint) g and clock resets r.

3.7. Component Abstraction Model (CAM)

The component abstraction model (CAM) has been defined in this thesis to perform an abstraction of system components which enables us to supply agents with the formal knowledge of the system components (black box consideration). The CAM forms the first layer of the MAIP Layered Abstraction modelling which is required to realise the agent environment. A composition of several CAMs has been used to represent the knowledge in order to form the second and third layer of the layered abstraction as shown in Figure 14 for aircraft systems:

- Equipment level (System component)
- System level (Multi-system component)
- Function level (Multi-system)

Dependent on the kind of knowledge to be presented and on how it is specified, the way to define a CAM can differ, but its purpose allows the same, namely, the representation of the knowledge about the system including the requirements to be fulfilled by it. In this thesis this kind of CAM definition has been developed and evaluated for the testing of system controllers and of network data security. For other test approaches we see no limitation to define a new kind of CAM that can be integrated and used within the MAIP without impacting other entities of it.

For the definition of a CAM two phases were planned. In the first phase the requirements formulated in Section 3.3 were analysed, whereas in the second phase the specification of the CAM was written.

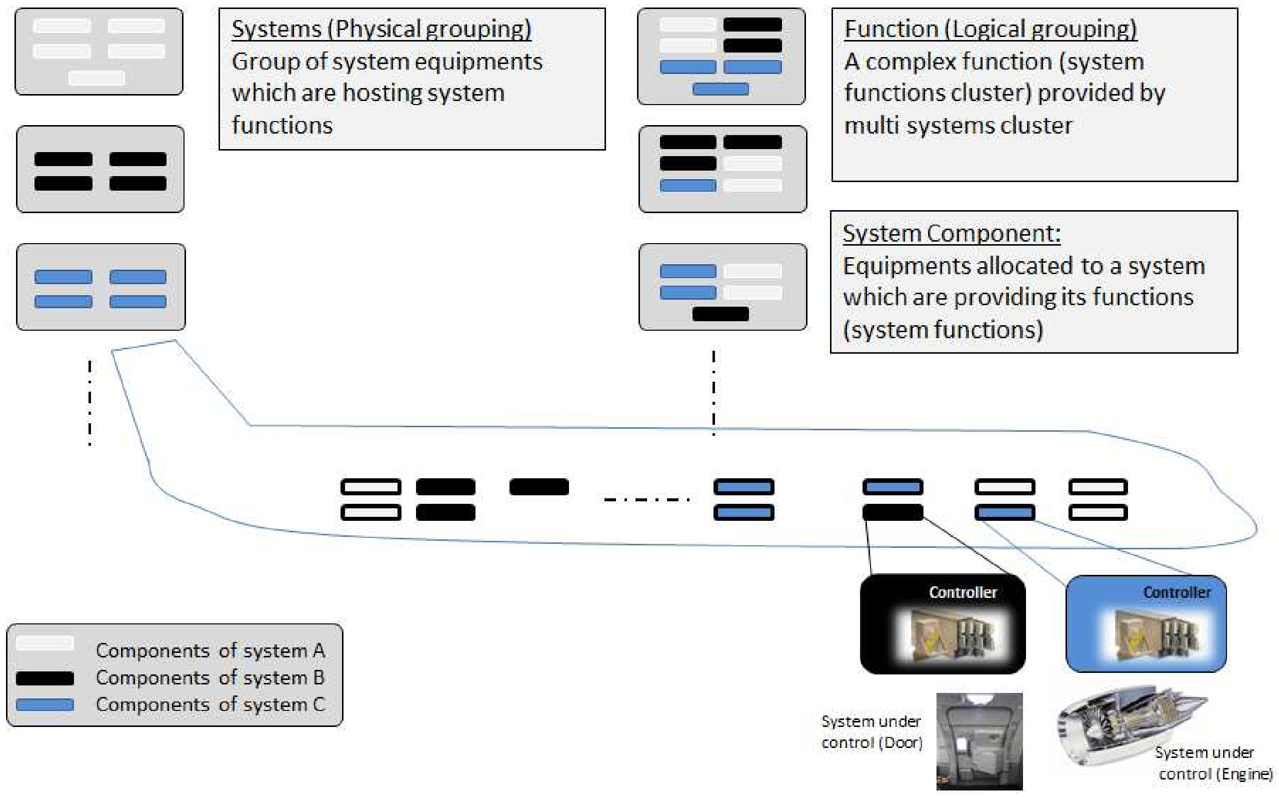

Systems (Physical grouping)
Group of system equipments which are hosting system functions
Function (Logical grouping)
A complex function (system functions cluster) provided by multi systems cluster
System Component:
Equipments allocated to a system which are providing its functions (system functions)
Components of system A
Components of system B
Components of system C
Controller
Controller
System under control (Door)
System under control (Engine)

Figure 14 Layered abstraction for aircraft system

3.7.1. Required capabilities

To be able to provide the agent with the required knowledge (agent's rules) and to interact with the required MAIP services, the following capabilities have been taken into consideration for the design of the CAM:

Durable saving:

The MAIP services and the CAM have been designed to ensure the capability to durably save the established knowledge with a higher confidence factor (see Section 3.8.3). In the nature of system component requirements there was no need for the ability to modify or change them by agent tasks, due to the fact that the changes in this area are done by system component designers by editing the CAM directly for changes or modifications.

Accessibility:

The CAM has been designed to be able to interact with agents and agent helpers during the whole test run time. Depending on the confidence and relevance factors of different access types that have been provided by the MAIP services, the following capabilities have been implemented:

- Knowledge expressions must be clearly interpreted by agents via CCM (Component Control Model (see Section 3.10.).
- Read accessibility for agent helpers including filtering.
- Read/write accessibility for agents.

Classified representation:

With respect to the different requirement classes and the different assessments regarding confidence and relevance, the CAM has been designed to represent different requirements classes (function, interface, state and performance) with the capability to assess each requirement based on the concept described in the knowledge-establishing process (see Section 3.8.3).

Extensibility:

For new knowledge gained by agent tasks the CAM has been extended by the so-called virtual CAM to represent all knowledge and rules with a lower confidence factor <3 (see Section 3.8.3). The virtual CAM has been designed to provide the MAIP services for rule-based and case-based reasoning tasks. When the knowledge represented by virtual CAM has been established and the confidence factor has achieved the highest value, the knowledge data has been moved to the CAM.

Filtering:

Depending on the scope of the V&V tasks, sometimes a subset of requirements is applicable; therefore, the CAM data structure has been modelled to enable the capability of filtering which can be then requested by the MAIP services (see Chapter 4).

Evaluation:

The rules described in the virtual CAM are under validation every time the knowledge establishing process performs until the knowledge data has been moved to the CAM by achieving the highest confidence factor. Therefore, the virtual CAM has been modelled in this thesis to allow different levels of accessibility so that the knowledge-establishing agent can modify or request the following data:

- Modifying of confidence factor (only highly confidential rules or knowledge can be used for verification to ensure that the gained knowledge has been validated that is required for aircraft testing).
- Modifying of relevance factor (only relevant knowledge is to be considered to reduce the amount of information to be managed by agents due to the performance required for the real-time capability).
- Request for rule changes (adding, deleting and modifying, when system behaviours deviate from the CAM).
- Request for test context changes (with respect to the function to be tested or system to be involved).

3.7.2. CAM Design

Given the basic division of the MAIP-design into Layered Abstraction and Context-Based Learning, the design tasks have been defined as a basis for any implementation of the design flow in an incremental way. In a first step, the CAM has been designed.

In this thesis we have also used a pragmatic widely distributed object-oriented technology to implement the beliefs and goals of agents in consideration of the essential requirements which have to be facilitated by the MAIP and have to be represented by the CAM.

This part of the work has been focusing on which system component models and concepts are necessary to model and support MAIP entities that contain abstracted system components representing system component requirements.

Knowledge representation research involves an analysis of how to accurately and effectively reason and how best to use a set of symbols to represent a set of facts within a knowledge domain. The CAM model in this dissertation has been designed to represent knowledge required to define the agent's facts. Rational agent programming has been motivated on several grounds. One of its motivations has been to provide for a high-level specification framework for agent programs based on common sense concepts such as beliefs, goals, actions, and plans. Such a programming framework comes with several benefits, among them that, though the programming framework is abstract, it can be realized computationally, and, that the programming framework is based on common sense intuitive concepts which nevertheless have a well-defined semantics. In this dissertation the knowledge engineering process has been used to identify the elements to be designed and implemented.

Design Principles

Context-based learning is defined by four principles; in this section the four design principles are identified and specified.

Principle 1: (Decomposition)

Motivated by the validation and verification platform for complex systems, context-based learning is designed for domains that are too complex (real-time, collaborative and competitive domain environments) for agents to learn direct mappings from their sensors to actuators. Instead, the context-based learning approach consists of breaking down a problem into several behavioural contexts using machine learning (ML) techniques for each context. Context-based learning uses a bottom-up incremental approach to hierarchical task decomposition. Starting with detailed context behaviours (system component level), the process of creating new ML subtasks continues until global strategic behaviours are reached that deal with the full domain complexity (multi-system, functional and operational level). The detailed level has been broken down in four different sub-contexts to build the function, interface, state and performance critical contexts.

Principle 2: (Context-Based Consideration)

The appropriate behaviour granularity and the aspects of the behaviours to be learned are determined as a function (system component operation or multi system function) of the specific domain. The task decomposition in context-based learning is not automated. Instead, the contexts are defined by the machine learning opportunities in the domain. Context-based learning can, however, be combined with any algorithm for learning abstraction levels. In particular, let A be an algorithm for learning task decompositions within a domain. Suppose that A does not have an objective metric for comparing different decompositions. Applying context-based learning on the task decomposition and quantifying the resulting performance can be used as a measure of the utility of A’s output. Figure 15 illustrates abstract context-based learning task decomposition within a collaborative and competitive domain. Learning can begin with individual behaviours, which facilitate multi-agent collaborative behaviours, and eventually lead to full collaborative and competitive behaviours.

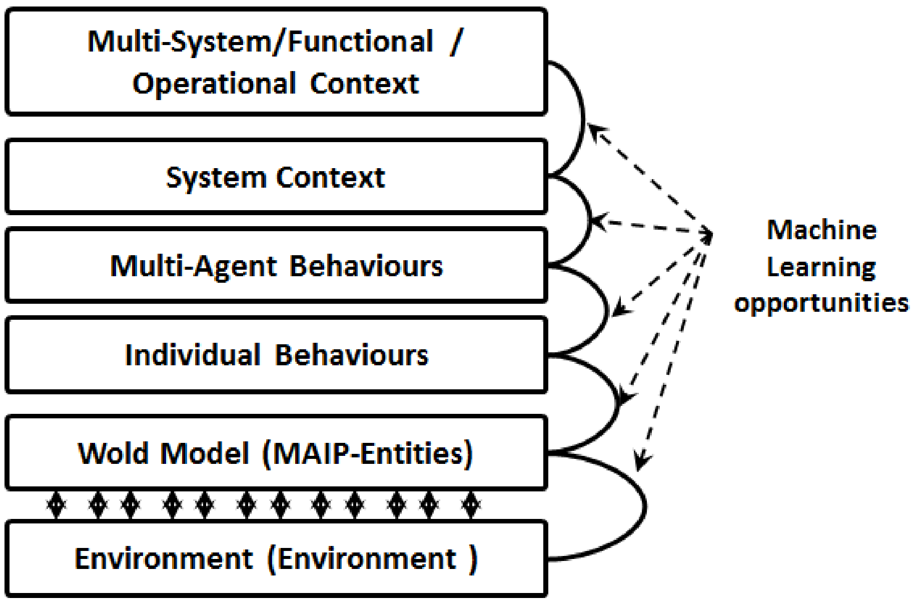

Multi-System/Functional / Operational Context
System Context
Multi-Agent Behaviours
Individual Behaviours
Wold Model (MAIP-Entities)
Environment (Environment)
Machine Learning opportunities

Figure 15 A sample task decomposition within context-based learning

Principle 3: (Machine Learning)

Machine learning is used as a central part of context-based learning to exploit data in order to adapt the overall MAIP environment. ML is useful for behaviours to be adapted that are difficult to fine-tune manually. It is useful for adaptation when the task details are not completely known in advance or when they may change dynamically. In the former case, learning can be done off-line and frozen during actual task execution (see Section 4.3). In the latter case, on-line learning is necessary: since the agent needs to adapt to unexpected situations, it must be able to alter its behaviour even while executing its task. Like the task decomposition itself, the choice of the machine learning method depends on the subtask.

Principle 4: (Hierarchy Traversal)

The key defining characteristic of context-based learning is that each context directly affects the learning at the next higher context. A learned subtask can affect the subsequent context either:

- by providing a portion of the behaviour used during adaption or
- by creating the input representation of the learning algorithm.

In general, machine learning algorithms and decision trees require an input and output representation, a target mapping from input(s) to output(s), and adaption. The goal of learning is to generalize the target mapping from the adaption examples which provide the correct outputs for only a portion of the input space (operation oriented on the system component level).

When using ML for behaviour learning, adaption examples are generated by placing an agent in a situation corresponding to a specific instance of the input representation, allowing it to act; and then giving some indication (for example validated or verified) of the merit (for example test case executed) of the action in the context of the target mapping. Thus, previously learned context can:

- Provide a portion of the behaviour used during adaption by either determining the actions available or affecting the reinforcement received.
- Create the inputs to the learning algorithm by affecting or determining the agent's input representation.

If each learned context in task decomposition directly affects the learning at the higher context, then the system is a context-based learning system, even if the domain does not have identical properties to those considered in this thesis. Without this characteristic, the approach does not fall within the realm of context-based learning.

In general, for the implementation of context-based learning, each MAIP entity as well as a CAM has been modelled by the following design principles:

- Each entity has input and output interfaces linked directly to the real time signals.
- The entity logical core is the representation of the internal operation/function hosted by this entity, where state transitions are also considered as an operation.
- The logical core consists of additional parts linked to the internal operation/function to define the following relationships:
    - The linking between operation and signals.
    - The valid state of the operation.
    - The operation performance constraints.
- The accessibility is assigned on the operation level.

In this and the next section, the CAM design and its affected characteristics are described wherefore the CAM entity model has been used.

For operations which cannot be represented by this definition due to its complexity, the following approach has been applied:

- The operation is to be implemented as a simulation (for example in the programming language C).
- The parameter list (inputs/outputs) of the simulation is to be defined (for example header file).
- The operation state and performance requirements are to be defined (for example additional header files).
- Creation of the CAM specification.

Assumption:

- With respect to the design principle 1, system component behaviours defined by system component requirements are specified by operations representing the behavioural breakdown of the system component.
- With respect to the design principle 2, the operation has been considered as the lowest granularity entity required for the CAM design.
- A system component operation represents one of the transfer or state transition functions allocated to the system component.
- A black box consideration is applied to the representation of transfer function; the internal specific implementation of the transfer function is not part of this representation.
- The definition of the operation is part of the CAM specification, but the parameter assignment applied to an operation is part of the respective CCM specification (see Section 3.9) linked to this CAM and used by an agent (see Section 3.8).
- Operation assignment is linked to multiple states (at least one).
- The performance constraints are part of the operation assignment and are valid for the linked states.
- With respect to design principle 3, the CAM specified by the system component designer (off-line process) is a static entity which cannot be modified by agent's tasks (CAM logical core).
- The CAM extension (virtual CAM) created by agent's tasks (on-line process) is a dynamic entity and is affected by learning and knowledge establishing processes (see Section 3.8.3).

Characteristics affected by learning:

With respect to the design principle 4, the knowledge representation described by the CAM-specification has been designed in order to be capable to represent the knowledge gained by the context-bases learning process. According to the three-stage design processes applied to the aircraft system development process, three respective learning contexts have been defined:

- System context (according to the aircraft system development process)
- Multi-system context (according to the aircraft system interface development process)
- Global context (according to the functional and operational design process).

Section 4.3 describes the context based learning process.

3.8. Agent

With respect to the agent-based testing process (see Section 3.5) the research method used for the representation and modelling of agents and agent helpers was based on three research methods:

- The first one was a lessons learned study from our own V&V testing environment (V&V TE) developed for the A380 and A350 aircraft cabin testing domain to provide the initial requirements for our approach.
- The second method used was a study of the existing agent methodologies developed and established in the software development domain. In Chapter 2 the existing methodologies have been reviewed and analysed.
- The third method used was the definition of multi-agent requirements resulting from workshops performed and reviewed within the agent-based testing project (ABT-Project) which was

managed and accounted by the aircraft cabin and cargo testing team. The ABT-project has been established to improve the testing process with respect to the following aspects:

- Encapsulating the complexity so that test systems are made usable by non-experts.
- Reducing the time required for test preparation and test creation.
- Simplification of test design using environmental modelling and visualization.
- Formalization and auto-validation of interface~~s~~-definition requirements.
- Increasing the test coverage.
- Designing an environment-adaptive and self-learning test platform.
- Enabling re-usability by high degree of modularization (Test Library).

In our approach to modelling agents and agent helpers, we designed a simple basic agent model (see Figure 8) in the first step and improved it in the next steps by considering different aspects derived from our three research methods used. In the next sections these different aspects have been presented and discussed to model the agents and agent helpers that took over the tasks to be performed by the MAIP–agents. In this thesis the development of a generic agent modelling methodology and its application in the V&V domain of system testing have been analysed and assessed.

Consider how the modelling of any complex dynamic system is initially performed in the everyday world. We observe that:

- It is natural to conceptualise the relevant features, i.e. the behavioural components to be modelled, in terms of simple or familiar V&V items.
- The overall model structure and the number of individual items must be kept simple enough so that the entire model can be easily understood, manipulated, and modified, if necessary.

There is considerable interest in agents and, consequently, there is a range of views relating to an agent's characteristics. Some of the essential properties based on the research review presented in Chapter 2:

In our approach, we added an additional component of agents: the processor (agent internal process see Figure 16 (5)). Every modelling element in our MAIP was essentially an agent, but there are two degenerate forms of agents which were given alternative names: instance and processor. Thus, the modelling world consisted of three types of entity: agents, instances, and processors. Agents can request other agents and processors to operate on their own instances which can lead to goal changes. Instances, on the other hand, are acted upon by agents and cannot initiate any action; they essentially store information.

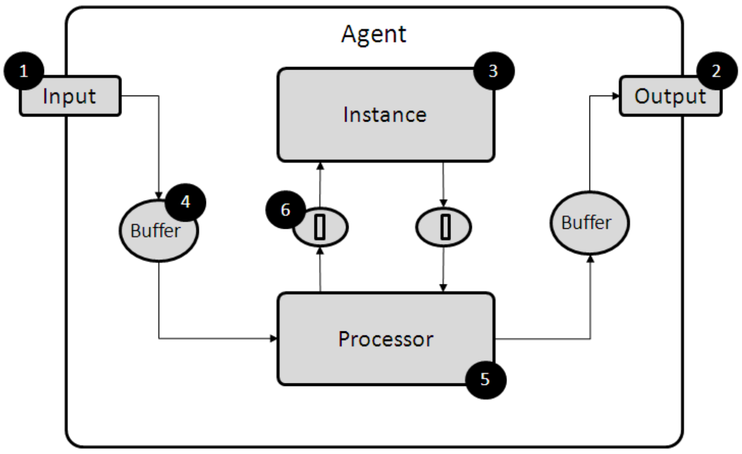


Figure 16 Basic agent model

Processors are agent internal processors that can be commanded to act on an agent instance (and on other agents) by external agents. Figure 16 shows the data exchange lines of action available to be processed. Any received data is forwarded to the processor and supported by a buffer unit. The processor represents the internal utility capability of an agent to process data with respect to the current instance data of this agent. If the data processing is completed and the instance is updated, the resulting data can be buffered and sent to the output. In the next section the basic agent model has been improved and extended in order to be adequate to fulfil the required feature of the desired generic agent. This desired generic agent was the fundamental entity of our MAIP-platform. The agent helpers are modelled to be supporting entities (processor) and are implemented to assist the specification of the generic agents. The solution of the agent specification has been analysed and discussed in the next sections according to different agent features and desired agent models for the V&V tasks to be performed in the MAIP-platform.

3.8.1. Agents' Processes

The basic agent model (depicted in Figure 16) represents one entity "processor" which comprehends all acts to be performed by agents supported by agent helpers which are called agent processes in this thesis. This entity can be easily refined so that the processor entity can handle different tasks depending on different goals and intentions and based on knowledge established by the agent at a certain point of time. Regarding the resulting requirements which were analysed and reviewed in the ABT-project, there were three groups of tasks that an agent processor has to perform:

- V&V tasks: These tasks were directly linked to action to be performed by agents within the context of system and system components V&V wherein the pass/fail criteria (agent facts) are defined and established. These tasks are used to improve the test configuration definition, test case definition and implementation activities (described in Section 3.3).
- Knowledge establishing tasks: These tasks (updating of facts) were responsible for managing agent facts. These tasks are used to improve the test evaluation activity (described in Section 3.3). Three knowledge establishing tasks have been analysed and discussed in the next sections:

    - Knowledge expanding
    - Knowledge assessment regarding confidence and relevance
    - Knowledge transfer to facts and linking to tasks

- Goal definition tasks: These tasks were responsible for determining the next agent goal and for updating the agent instance entity. These tasks are used to improve the test design definition activity (described in Section 3.3).

<u>V&V tasks</u>

Given the basic division of the agent modelling process into computation, communication and V&V task design, design steps have been defined as a guideline for any implementation of the agent's models in an automated way. In a first step, the input and the output of agent model tasks have been defined. More specifically, corresponding abstraction levels as determined by the amount and granularity of implementation detail represented in each model have been specified. Clear definitions of inputs and outputs of each task then serve as a specification for the implementation of each design task. In general, characteristics and features of each input, output and intermediate model have been defined such that interactions between tasks were minimized, automation of design tasks became feasible, and reliable feedback for design space exploration was provided.

Details of agent-task level design steps are listed below in terms of necessary design decisions, resulting implementation details in the form of structure and order, and applicable quality metrics for evaluation and exploration. Generally, for each design step, a new design model has been generated that reflects the design decisions made by introducing corresponding implementation details. However, if steps are dependent on each other, they have been combined into a single model in order to obtain useful feedback. On the other hand, independent steps have been separated into different models in order to break the problem into smaller parts, manage complexity and separate concerns. With each new model, a minimal number of new features has been introduced such that they have not been influenced by later steps and can be evaluated and decided at a high level of abstraction (agent processes). The goal was therefore to organize steps into a minimal, orthogonal set of design models such that each model provides the designer with reliable feedback for design and exploration.

Agent-tasks design steps

- Linking of the process task to the system component entity with respect to the resulting requirements and to the V&V task to be performed.
- Generation of the V&V procedure that can validate or verify the respective system component entity (operation, physical and logical interfaces, states and performance).
- Definition of the tasks channels (input/output interface) based on the knowledge allocated to agent instance and stored in the global agent facts.
- Determination of notification and message receiver entities which may be affected by the result of this task.
- Definition of the time schedule and of the time-out constraints within the V&V task which has to be performed completely.

Based on the ABT-project requirement, the study of system component computation and the reviews performed with aircraft and aircraft cabin test teams, the following items have been taken into account:

- Function requirements (system component operations)
- Interface requirements (system component physical interfaces).
- State requirements (system component states including the global system)
- Performance requirements (system component operation schedules)

The next section explains how a verification process has been developed. The function verification process has been chosen because of its generic characteristics.

3.8.2. Agent Function Verification Process

The aim of the function verification process (responsible for the function verification tasks) was to check that each function represented by the system component operation fulfilled the function requirements. As mentioned above, a formal representation (CAM operation) made it possible to establish appropriate rules for automatic verification.

The first step in the function verification process was the definition of the processing architecture. Our new approach to the function verification process has been based on the lessons learned study and on the requirements of the ABT-project. The aim of this process has been to allow the verification of different points of control (POC) and observation (POO) and to enable the capability to verify system component functions regardless of their states, interfaces and performance. This requirement was essential in the complex system domain, due to the fact that the same system component operation shall be performed differently depending on their different interfaces (system redundancy requirement for highly reliable systems), states (operation validity requirements for operation inhibition for certain states on component and system levels) and performance (required for different operation modes, for example normal and emergency operation in the context of the same functionality). Keeping these requirements in mind and with respect to the agent-tasks level design steps, the function verification task has been designed as a specification of the generic basic agent model supported by different agent helpers. In our function verification process different agent helpers were required to support the verification process to ensure the achievement of the function verification task.

Required agent helper:

- **Operation designator:** This helper is responsible for providing the function verification agent with the operation specification.
- **Operation verifier:** This helper is responsible for providing the function verification agent with the test case specification including the pass / fail criteria and its execution (operation scheduler based).
- **Operation router:** This helper is responsible for providing the function verification agent with the operation interface specification including time constraints.
- **Operation messenger:** This helper is responsible for providing the function verification agent with the message and notification information required to be exchanged with the MAIP-platform.
- **Operation scheduler:** This helper is responsible for providing the function verification agent with the time schedule and time out constraints within the V&V task that has to be performed completely.

The results of the modelling of the specific function verification process (reactive agent with static instance data) on the system component operation level are depicted in Figure 17.

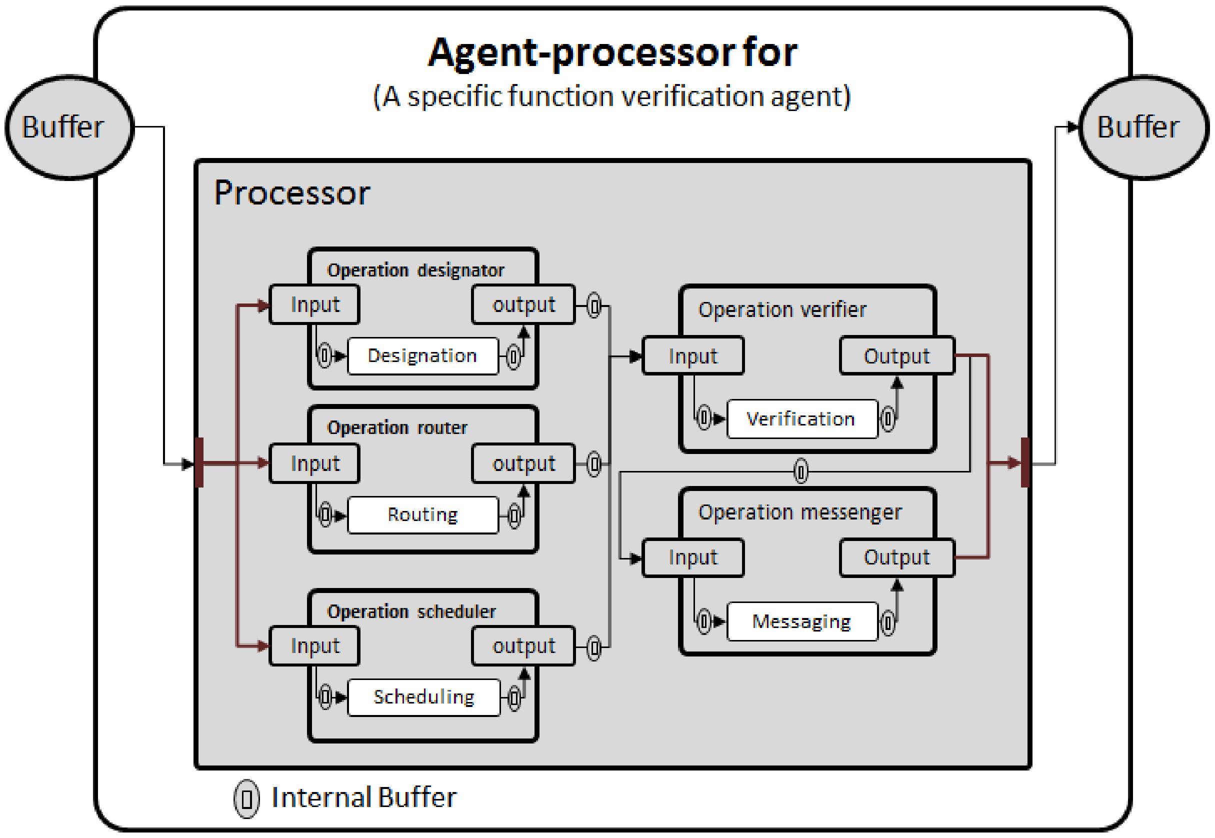


Figure 17 Agent function verification process

3.8.3. Knowledge establishing tasks

This section presents research methods that have been applied in the field of knowledge-based development research, and how this work relates to this thesis.

Software development is a multi-disciplinary and complex research field stretching from information representation, processing and interfacing through modelling and languages, to more philosophical research like rationality and reasoning, etc. In this thesis several aspects of knowledge-based methodology have been analysed, design requirements have been formulated and reviewed by the ABT-project, and the deliberative agents with appropriate central knowledge management units have been designed and presented in the next sections for the definition of the knowledge-establishing tasks as a second agent process.

The CAM designed for the representation of knowledge already established by the system component designer is only a part of the knowledge that is collected and defined during the system component development. Another part of knowledge can be gained during the V&V tasks supported by agents and agent helpers which can be classified as shown below:

- Requirement correcting information as results of invalid or not verifiable requirements. Implementation failure reported by agent helpers cannot be considered as new knowledge, due to the fact that the system component requirements are still valid.
- Interface requirements which are valid on the system component level but not harmonized with other system components so that the interaction between system components is impacted. This type of failure is commonly because of the limited system component responsibility and the missing integration instance in the hierarchal oriented organisation unit in industrial companies.
- New requirements which have not been considered during the system component development due to :
    - Immature system component design.
    - Design changes on other interfaced system components.
    - Complex dependency between system components that has not been considered during the development phase.
    - Global safety and security requirements which have to be fulfilled by each system component.

For this type of failure the agent-based approach is very efficient, and the CAM has been extended to manage them. The CAM extension in this thesis has been called virtual CAM which is subject to the knowledge assessment and publishing process described in the section information identification process.

3.8.4. Knowledge-Based Methodology Aspects

- **Knowledge resources & representation:** Resources to help agents to learn about and generate effective explanations were provided in either a text (e.g. the system shall be initialised within 200 seconds) - or agent-based format (see Section 4.2), depending on conditions (see below). The resources included advice with respect to the characteristics of elaborative explanations and some examples. Although the presentation of these resources varied between groups (text-based or agent-based), the general format and content was the same. The following methods and techniques have been taken into account in this thesis for the knowledge-establishing tasks:
    - Identify information sources: In our system V&V area the information sources were the system requirements and the system design documents.
    - Represent information: Under the assumption that all system requirements and system design documents are formally defined (this is mostly the case in the system development area), our designers have considered the method of information representation to help agents to handle it.
    - Classify information: Based on the four tasks of V&V (function, interface, state and performance), the information has been classified and grouped appropriately.
    - Identify new information: The knowledge-establishing task has defined methods for the identification of new information (with respect to **classification**, **confidence** and **relevance**).
    - Relate new information sources to prior knowledge: Methods have been established to assess new information types before they may be identified as new information.
    - Identify knowledge dependencies.
    - Publish and subscribe information: Methods for the publishing and subscribing (implicitly requesting of information) of information within the agent environment have been defined to be a part of the knowledge-establishing task.

- **Knowledge transfer:** Research in the domain of knowledge transfer focuses on the development and evaluation of theories, modelling languages and instruments for knowledge transfer.

  - Theories: Theories for knowledge transfer aim to satisfy the epistemological need to understand and explain the nature of knowledge transfer itself (such as [M. E. Nissen 2002]). In an organizational context, theories for knowledge transfer aid the understanding of the nature of knowledge relations in and across organizations (in our thesis, agents and their environment) on a conceptual level. Knowledge flow theory [M. E. Nissen 2002] provides a classification of different types of knowledge transfer in organizations and a way for representing the transfer. Knowledge transfer is theorized by distinguishing between situational, source, transfer, relational, recipient and organizational context..
  - Modelling languages: Aid the identification and visualization of concrete knowledge relations among specific organizational entities (such as specific agents, agent roles, agent environments, agent tasks and goals, etc). In this thesis the modelling language used for modelling of knowledge transfer is based on coloured Petri-nets approach (see [Heiko Rölke 2004]).
  - Instruments: Instruments for knowledge transfer aim to improve on and facilitate different aspects of knowledge relations. Technological instruments for knowledge transfer include synchronous and asynchronous communication tools such as wikis, discussion boards, or expert locators. Organizational instruments include for example mentoring, experience factories and job rotation [M. E. Nissen 2002]. For this approach a combination of data base query, notification and messaging techniques (developed in JADE based on FIPA [Bellifemine 2011]) have been used.

- **Knowledge dependencies:** Strategic knowledge dependencies between different actors (such as CAM, CCM, agent environment and agents) have been identified. Questions for identifying knowledge dependencies include:
  - Who do actors turn to for advice and/or expert knowledge?
  - Who turns to them for advice? What kind of knowledge is involved?
  - Integrating the answers to these questions into knowledge-establishing agent’s process models results in a set of identified strategic knowledge dependencies between a set of actors, as described in the results of our knowledge-establishing agent.
  - The supportive means that are utilized by the participants of a specific knowledge dependency are identified. Questions for identifying these means include: How does knowledge transfer take place? And what kind of communication channels and storage objects are involved? (As described in the result of our knowledge establishing agent).
  - The modelling method used in this thesis suggests re-conceptualizing knowledge dependencies and corresponding supportive means as a distinct agent process within the knowledge-establishing agent’s process. In this thesis this is the so-called **Knowledge Transfer Process**.

Having an agent-oriented modelling approach for analysing the effectiveness of knowledge transfer instruments available can be expected to yield the following benefits:

- **Effectiveness** of knowledge transfer instruments in specific situations can be analysed before deploying them in their environments (agents), thereby potentially reducing costs of errors or experimentation and facilitating the “process of design”.
- **Alternatives** to available knowledge transfer instruments can be formally explored rather than randomly identified, thereby potentially facilitating a “process of improvement”.

Approaches for the development of models that reflect relationships between goals, between agents, and between goals and tasks exist (such as goal interdependency graphs, and strategic dependency and strategic relationship diagrams, (see [Muneendra 2012]). In the domain of requirements based engineering (and in this case requirements-based V&V), these approaches have been applied, for example, to justify software designs, to explore and select among design alternatives, and to trace features back to higher level system goals. In the following section, first the agent-oriented modelling requirements have been introduced, and subsequently an approach has been illustrated to analyse the effectiveness of knowledge transfer instruments on different levels of detail with respect to the identified aspects of knowledge resources, representation and dependency.

3.8.5. Agent Information Identification Process

- Definition of the information identification process and identification of affected entities.
- Assessment of the identified information and the classification of information regarding confidence and relevance.
- Updating of the knowledge databases and the definition of the publishing / subscribing policies.
- Definition of the publishing methodology including notification, message and subscribing.

Based on the ABT-project requirements, the study of system component computation and the reviews performed with aircraft and aircraft cabin test teams, the following items have been taken into account:

- Knowledge about function (system component operations represented by the CAM).
- Knowledge about interfaces (system component physical interfaces represented by the CAM).
- Knowledge about state (system component states including the global system state represented by the CAM).
- Knowledge about performance (system component operation schedules represented by the CAM).

The aim of the information identification process was to check that new information, information modification or information invalidity represented by the knowledge database is identified. As mentioned above, formal representation makes it possible to establish appropriate rules for automatic identification.

The first step in the information identification process was the identification of the system component requirements (function, interface, state and performance requirements) of all involved system components. Our new approach to the information identification process was based on the lessons learned study and on the requirements of the ABT-project. The aim of this process was to allow the identification on different levels of information sources to enable identification of the base information that can be changed for the following reasons:

- New requirements have been added to the information base by designer or agent activities.
- An already existing information entity has been modified with regard to content, confidence or relevance.
- An already existing information entity has lost its validity caused by design changes or agent activities.

The information availability and accessibility is essential in the complex system domain, due to the fact that the same information shall be distributed to different entities, maybe leading to re-definition of agent rules, tasks and goals without impacting the agent plan. If the agent plan is impacted, the agent shall be replaced with a new one for re-establishing a new plan to achieve the V&V task. Keeping these objectives in mind and with respect to the agent-tasks level design steps, the information identification task is designed as a specification of the generic basic agent model supported by different agent helpers.

<u>Required agent helpers</u>

- **Information designator:** This helper is responsible for indicating new, modified or invalid information in the MAIP context caused by the system component requirement updating process.
- **Entity designator:** This helper is responsible for determining affected entities within the MAIP context, when information has been updated.

- **Information messenger:** This helper is responsible for providing the MAIP with the message and notification information required to be exchanged with the entities within the MAIP-platform.

Agent's information assessment process:

The second step was the definition of the method (process) which handles the assessment (evaluation) of the new, modified information or invalid information. Three aspects of the information evaluation process have been considered:

- **Classification:** The first process was responsible for the classification of the information. For new information the classification is dependent directly on the source class so that function requirement information can directly be classified to the function knowledge database entities, and the same holds for interface, state and performance requirements. If the information has only been modified, the class of the information will be kept. Information which has become invalid will be marked as deleted or obsolete within the knowledge database. For information originated by agents, the agent that identified the information is responsible for its classification.
- **Confidence:** A confidence factor (1 to 3) has been defined to quantify the confidence of new information or modified information. Information which is directly generated by system designers and stated as a system requirement will be quantified as highly confidential without any assessment (confidence factor = 3). Information which is gained by agent activities will be quantified as lowly confidential (confidence factor = 1) until a second agent has approved the information (confidence factor = 2). When a third agent has approved the information, again the confidence factor will be increased to 3.

  Relevance: A relevance factor (1 to 3) has been defined to quantify the relevance of new information or modified information. Information which is directly generated by system designers and stated as system requirement is already quantified by definition without any assessment (relevance factor = 3). Information which is gained by agent activities will be quantified as low relevance (relevance factor = 1) until a second agent has requested the information (relevance factor = 2). When a third agent has requested the information, again the relevance factor will be increased to 3

Required agent helper:

- **Information Classifier:** This helper is responsible for the classification of new, modified or invalid information in the MAIP context caused by identification of new information.
- **Information Confidence Qualifier:** This helper is responsible for the qualification of the confidence of the new information caused by identification of new information.
- **Information Relevance Qualifier:** This helper is responsible for the qualification of the relevance of the new information caused by identification of new information.
- **Information messenger:** This helper is responsible for providing the MAIP with the message and notification information required to be exchanged with the entities within the MAIP-platform.

Agent's information publishing process:

The third step was the definition of the method (process) which handles the publishing and distribution of the centrally managed information and how this information can be requested by agents periodically or on demand. Information availability and accessibility is essential in the complex system domain. The occurrence of new information shall be notified to each entity which is affected by the **Entity Designator.** When an entity has been informed about the occurrence of new information, the agent which hosts this entity requests the new information by the central knowledge database instance and decides if the agent instance, agent rules, or task have to be updated. If the agent's goal loses its validity and the information is highly confidential and relevant (determined by the respective information qualifier) the agent shall terminate itself with the notification (including user notification per reporting, logging or graphical user interface) "goal not achievable". The subscribing process is not a part of this publishing process and has been designed in the **Instance Synchronizer** agent.

Required agent helper:

- **Knowledge databases synchronizer:** This helper is responsible for the updating of the knowledge database after the information has been assessed with respect to confidence and relevance on the MAIP-platform level.
- **Information adviser:** The information adviser is responsible to notify all entities which are affected (determined by the entity designator) that relevant and confident information (newly identified) is ready to be requested. The requesting process responsibility is allocated to the agent that may request this information due to the autonomy of the agent decision (see Chapter 2).
- **Entity synchronizer:** This helper is responsible for updating affected entity instances when and only when this helper is delegated to perform this task by agents whose instances need to be updated.
- **Publishing messenger:** This helper is responsible for providing the affected entities (determined by the entity designator) with the message and notification information required to be exchanged with the entities within the MAIP-platform.

3.8.6. Goal definition tasks

This section presents research methods that have been applied for the determination of the goal definition tasks and their representation in the agent instance entity. In this section, the goal definition and specification have been taken into consideration for the definition of these tasks:

Goal Definition

Based on the BDI system (see section 2.3.1) four types of goals are supported: Perform, achieve, query, and maintain goals. A perform goal states that something should be done but may not necessarily lead to any specific result. The achieve goal describes an target state to be reached, without specifying how to achieve it. Therefore, an agent can try out different alternatives to reach the goal. The query goal represents a need for information. If the information is not readily available, plans are selected and executed to gather the needed information. The maintain goal specifies a state that should be kept (maintained) once it is achieved. It is the most abstract goal in BDI systems.

Unlike traditional BDI systems, which treat goals merely as a special kind of event, goals which represent the V&V objectives defined by IEEE 829-1998(1) have been identified as a central and centralized concept in this thesis. Based on the defined and specified agent's process there was no need to develop different types of agent goals. To keep the principle of agent autonomy (see Chapter 2) in this research work a new goal definition agent has been developed which is responsible for providing MAIP-agents with the current goal information. Due to the nature of function-oriented testing developed for the V&V testing environment (V&V TE) in the A380 and A350 aircraft cabin testing domain, the V&V objectives are defined as a part of the test design activity. The defined V&V objectives have been analysed with respect to the following aspects.

- Availability of information required to define agent goals
    - Functions to be verified.
    - Interfaces to be validated or verified.
    - States to be validated or verified.
    - Performance requirements to be verified.
- Lifecycle of agent goals:
    - Monitoring duration
    - Stimulation duration
    - Goal lifecycle
    - Goal relevance.

(1) IEEE 829-1998 IEEE Standard for Software Test Documentation

Based on the analysis of the V&V objectives (VVOs), the following VVO-oriented tasks have been identified as goal definition agent processes:

- Determination of the required or involved agents
- Determination of functions, interfaces, states, and performance requirements which have to be validated or verified.
- Establishing of goal information.
- Refreshing goal information
- Publishing / Re-publishing of goal information

Goal Specification

Based on the requirements derived from the tasks to be performed by the goal definition agent, there was a need to model a reactive agent (with dynamically changing instance data) which shall permanently determine the goal changes by considering the test design-oriented execution steps with the following required agent helpers:

- **Agent designator:** This helper is responsible for indicating agents which have to be activated to perform the VVO-oriented test design. The information provided by the test design, the respective system component definition and the respective configured CAM (see Chapter 3.7.2) has been the source of this identification capability.
- **Requirement designator:** This helper is responsible for determining functions, interfaces, states, and performance requirements to be validated or verified. The information provided by the test design and the respective configured CAM has been the source of this determination capability.
- **Goal establisher:** This helper is responsible for the definition and refreshing of the VVO-oriented global goal and the respective goal tasks based on the information provided by agent and requirement designators.
- **Goal Publisher:** This helper is responsible for providing the affected entities (determined by the agent designator) with the message and notification information required to be exchanged with the entities within the MAIP-platform.

3.9. Component Control Model (CCM)

With respect to the agent-based testing process (see Section 3.5) there was a need to adapt the agent context derived from the test model (see Section 3.3.2 to the environment. For this adaption a specific control model has been developed to perform specific testing tasks taking into account in the characteristics of the thesis question (see Section 1.4):

To distinguish between inter-system functions and component internal functions the component functions have been called **operations** in this thesis. For the operation a database model has been designed including input, output, control parameters and operational constraints. In order to validate individual elements of a system component or the system as a whole in the V&V domain, some of the system components or systems will be physically connected to the test system; others will need to be simulated. Additionally, the environment in which the system component operates needs to be simulated. In this thesis the interface of the system component to other entities of the test system has been defined by a set of test system signals in at least one signal group.

The CAM has been defined for a representation of the system component as an entity that the other test system entities can deal with. The CAM is responsible for any interaction with the system component, which is physically connected to the test system. The following capabilities have been implemented within the test system to equip them with the ability to abstract the system component:

- Physical connector to adapt the system component.

- Hardware abstraction layer (HAL) to enable the interaction with the system component. The HAL is an environment that provides a simple device driver interface (I/O Modules) for programs to connect to underlying hardware.
- Configuration layer for the configuration of the HAL by implementing the system component requirements (I/O characteristics and I/O constraints).

The CAM signal definition serves to allocate physical parameters of the system component to an appropriate memory space that can be accessed by other entities of the test system. Also in this thesis solution signals represent the system component parameters in the test system memory. The accessibility of a system component for all test system entities' parameters has been ensured by this signal definition (parameter memory representation). For other test system entities the system component parameters have been represented by a flat list of signals.

3.9.1. HAL

The driver of the I/O modules have been extended with a virtual I/O API which hides driver low level interfaces from the application programmers. This virtual API has generic behaviours so that all I/O modules can be handled via a common interface. The required I/O driver architecture is depicted in Figure 18.

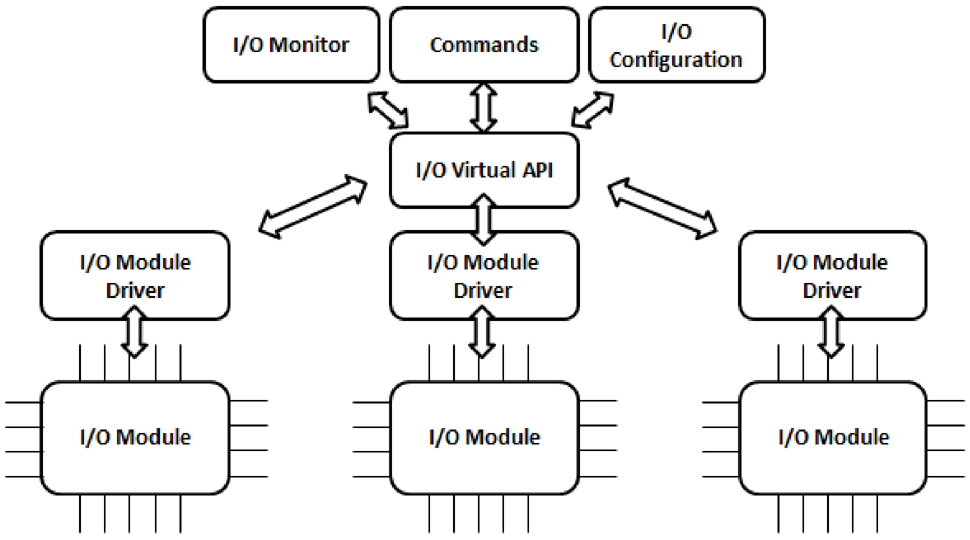


Figure 18 I/O driver architecture

<u>I/O CONFIGURATION</u>

The I/O configuration is a description of all integrated I/O modules in the test systems which will be managed by an I/O configuration file. For more details see Appendix B.

<u>HAL-COMMANDS</u>

A set of HAL-commands has been defined so that other MAIP entities can use them. This set of commands was used to allow an interaction with the I/O modules which are connected to the system component to be validated or verified. For more details about attributes to be set, signatures and description of the set of HAL–commands, see Appendix B.

3.9.2. CAM Physical Configuration Definition

The aim of the CAM entity configuration has been the generation of the CAM configuration file which specifies the CAM with respect to the following aspects (for more details see Appendix C):

- Definition of the system component interfaces to be implemented in the MAIP environment.
- Specification of the system component interfaces characteristics required for the interface instrumentation.
- Configuration of the system component interfaces to be instrumented in the MAIP environment by using the HAL-commands.
- Definition of the underlying layer of the CCM entity.

3.9.3. Signal Definition

Signals which represent system component parameters have been called “real time signals” to distinguish them from other signals defined by the test system or other entities.

A signal group is a data structure which is basically a table of records with the required parameters describing one table entry plus a data storage (memory space) holding the actual data associated with a signal. All test system software entities have been built upon the signal group, which serves as a distributed, real-time database holding all dynamic data of a test system application.

For the generation of a real signal group the system component parameters stored in the database (see Section 3.7.2) have been used. In the MAIP implementation the generated signal group (one group for each system component) has been saved locally on the test system and has been managed by using an appropriate versioning control system.

Signals for the monitoring and controlling of test system entities have been called configuration signals. These configuration signals have been delivered by test system suppliers and have been used to configure the test system and its I/O. From the software and test application view the configuration signals have been processed in the same way as the real signals. Signals for the monitoring and controlling of test system platform entities have been called platform signals. These platform signals depend on the platform which hosts the test system, have been delivered by test system suppliers and have been used to configure the test system platform. From the software and test application view the configuration signals were processed in the same way as the real signals (for more details see Appendix B).

The CAM has been designed to act as a local instance for the definition and modelling of system components and their requirements. For the V&V tasks in a complex domain there was a need to design control and monitor entities based on the CAM with the following features:

- Generation of test engine configuration files.
- Generation of configurable real time test cases (have been called **test simulations** in this thesis) based on the CAM requirements models.
- Building a remote access capability (has been called **agent adapter** in this thesis) for the reconfiguration of the test engine and the test simulations at run time.

To handle these tasks the CCM has been modelled to enwrap the CAM as a control and monitor unit (at start) that can interact with the CAM, its requirement models, satisfaction rules, prioritisations and interdependencies. The real time capability of the CCM interaction with the test simulation has led to the decision that two types of CCM adapters were required. The first one was the control instance running on the real time test engine which has been called CCM controller, and the second one was the delegation instance running on the agent platform which has been called CCM delegator and has acted as agent adapter.

3.9.4. CCM Models

The defined CAM has provided the capability of representing knowledge to the system component to be validated or verified. This representation has provided the opportunity to perform the following tasks:

- Definition of CCM units as the second abstraction of the Layered Abstraction for the generation, interacting, controlling and monitoring of test system entities:
    - CCM delegation unit (CCM Delegator): is responsible for the generation and instantiation of test system entities required for the execution of test procedures consisting of reconfigurable test cases. The re-configurability of test cases has been gained by the variability of test case steps based on the CAM-models' definition.
    - CCM control unit (CCM Controller): is responsible for controlling the execution flow of the test procedures and for the intercommunication between CCM Delegator and CCM Processor.
    - CCM process unit (CCM Processor): is responsible for the interaction with test system entities (system component under test, system simulation, control panel, data recorder and data logger) and is controlled by the CCM-Controller.
- The decoupling of system component knowledge represented by CAM models and test system entities by modelling of the CCM units.
- Based on the re-configurability of CAM, knowledge can be extended, modified, deleted or revalidated. This flexible knowledge handling managed by the CAM has provided the agent models with the required up-to-date information to be provided to the agent helper and consequently to the CCM delegation unit.

CCM Concept

For the execution of agent process tasks (see Section 3.7.2) the CCM units have been designed. The CCM delegation unit as a non-real time unit is responsible for the following tasks:

- Creation of a reconfigurable CCM Controller as a real time simulation to control and monitor the execution flow of the CCM Processor.
- Creation of a reconfigurable CCM Processor as a real time simulation to execute the test procedures and their test cases in the real time context.
- Reconfiguration of the CCM Controller and CCM Processor with respect to tasks redefined by the agent and the agent helper.
- Establishing of interaction between agent helper and CCM Controller.

Determinism and reproducibility

Tests in the aircraft system validation and verification are to be deterministic and reproducible. To achieve these requirements the CCM process has been designed as an elementary unit, i. e. a sequence of test steps which have to be persistently recorded. Any new or modified task created by agents has yielded to a new creation of CCM processes. The number and the order of CCM processes to be executed have been defined by the agents.

Real time capability

Tests in the aircraft system validation and verification are to be executed in real time. To achieve these requirements, the CCM process has been implemented as a real time test simulation where their parameters have been handled as real time signals managed by the operating system as a shared memory pointer.

- Test simulation signals which have represented the test and system component parameters.

- Test case step instructions for the reconfiguration of CCM process tasks (test case steps). The real time signals have stored the test case instruction instead of single parameters. Due to the fact that real time signals can be exchanged without any real time interruption, the reconfiguration has been ensured by exchanging the test case steps in the same way.
- Test simulation messages which have been used to inform the CCM control unit about the test case steps results.

An overview of the CCM model including an architectural overview is shown in Figure 19 and the knowledge management and the definition of virtual CAM are described in Section 3.8.3.

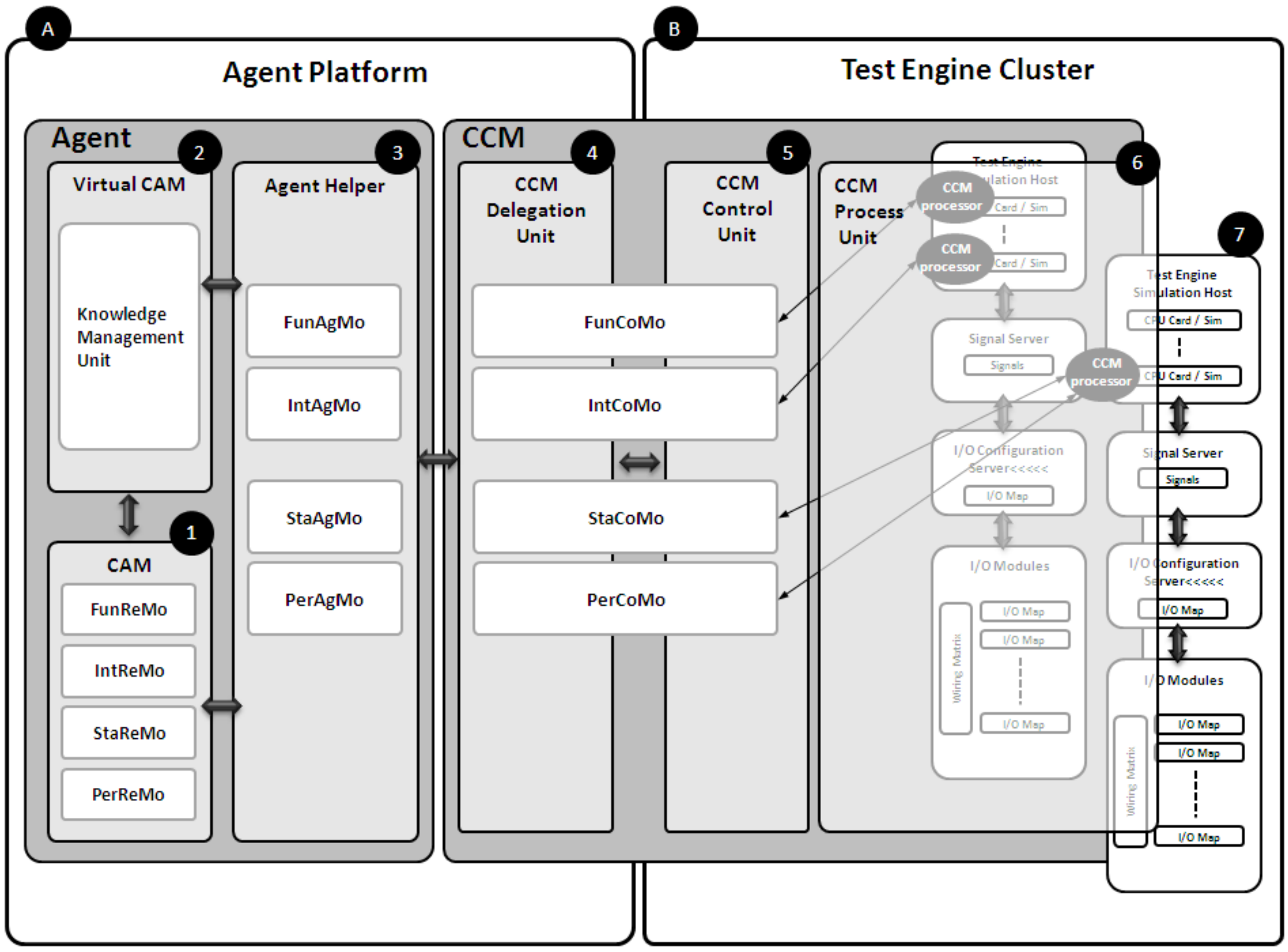


Figure 19 CCM Model

- CCM Model A: a platform on which agents are running (test specification process see Section 3.3).
- CCM Model B: an environment adapter consisting of a generic and a specific part. The specific part is closely dependent on the test engine connected with the agent platform.
- CCM Model 1: the CAM instance which is responsible for the knowledge representation of the functional, interface, state and performance requirements.
- CCM Model 2: the Virtual CAM instance which is responsible for the representation of the knowledge gained during the test and is to be established by knowledge establishing tasks (see Section 3.8.3).
- CCM Model 3: the instance of the agent helper (generic agent for deterministic tasks) which is responsible for supporting the agent instance when predefined and deterministic tasks are to be performed.

- CCM Model 4: the generic part of the CCM instance which is responsible for the transformation of agents' command into test engine specific commands.
- CCM Model 5: a test engine specific part of the CCM instance which is responsible for the control of all tasks being performed on the test engine.
- CCM Model 6: a test engine specific part of the CCM instance which is responsible for the processing of all tasks delegated by the CCM delegation unit.
- CCM Model 7: the test engine on which all real-time commands are being executed.

3.10.Conclusion

On the basis of the analysis which was done in the context of the ABT-project, we have assembled the fundamental requirements which were necessary to enable agent-based testing for complex systems. These requirements were supplemented by the objective to carry out minimal amendments in the existing development processes, since fundamental changes need reconstruction of the design processes which have been established over the years.

We have established that testing of system components, systems and multi-system functions which are subject to different development processes must be ensured. These different development processes are reproduced in the manner how these elements to be tested are specified, produced, commissioned/mandated and tested. Additionally, they differ in the manner in which they interact in different contexts.

For these different elements three different abstraction layers have been developed. These abstraction layers enabled us to handle different types of knowledge, representation of knowledge and processing of knowledge. This layered abstraction represents the first contribution of this thesis, since it allowed us to differently specify our generic entities in each layer. A generic entity (CAM) which can be specified in each layer and represents the element to be tested in the test platform has been defined for the elements to be tested. To represent a system component a cCAM has been defined (see Section 3.7). For the system and function levels a sysCAM and funCAM for functions have been modeled.

For inter-communication between the CAM entities and the element to be tested, CCM has been developed; it consists of two parts, a generic, test system-independent part and a specific part which must be adapted to the different test engines. For the development of CCM classical methods have been employed.

Since the elements to be tested are specified differently and must interact in different contexts, a knowledge-representation entity has been added for the CAM entities which had been created on the basis of the TAM (Timed Abstraction Machine) developed in this thesis. The necessity of this new development resulted from the requirements that system states and state transitions should be defined depending on time and time-oriented incidents. Additionally, the operations should be linked to the states and state transitions. The TAM is an extension of ASM (see [Alur 1998]) and represents the second input of this thesis.

CCM (Component Control Model) and CAM (Component Abstraction Model) are the foundations which have been provided for the agent approach. Since no suitable implementation was known for system verification, a generic agent model has been developed on the basis of the BDI model (see Section 2.3.1) which allows specific implementation of further specific agents. For implementation the classical object-oriented implementation has been applied. For specification of the agent-based model different, task-oriented approaches have been developed (see Section 3.8), inter alia the knowledge establishing process with its different tasks.

The fundamental modelling of agents of the knowledge establishing process has been employed to implement the different context-learning processes (see Chapter 4) which represents the second contribution of this thesis.

To be able to integrate all of these three main entities (CAM, CCM and agents), the MAIP (Multi Agent Integration) platform has been developed. MAIP has the task to supply a platform which allows inter-communication of different entities and platforms. For implementation of the MAIP classical methods of the implementation of distributed and realtime objects have been employed. The MAIP provides additional services (messaging, notification, reporting, etc.) by the means of which inter-action between the entities is secured.

CHAPTER FOUR MAIP CONCEPT

4. MAIP Concept

This chapter of the thesis focuses on which models and concepts are necessary to model and support V&V processes that contain both distributed, cooperative processes on the multi-system level, and local, individual processes on the system level. We found that it was necessary to divide between cooperative and individual processes. It was therefore required to specify the interaction between cooperative processes, individual processes and the entities of each process. Our research in models and concepts has resulted in three fundamental sub-processes which can be integrated in an overall concept of the MAIP. The first sub-process of the system layered abstraction has been developed to break down the complex system into three layers so that the agent platform can handle them efficiently. The reason for this breaking down is based on the nature of the three phases of specification for complex systems and the contracted suppliers; equipment, system and functional (multi-system) specifications. The second sub-process has been developed to ensure the capabilities of representing knowledge on each layer defined by the system abstraction and required by the building of agents' rules. On each abstracted layer different agents are acting differently to achieve different goals within different contexts. It was therefore required to develop different learning strategies with respect to the different contexts in which agents are acting to gain knowledge. In the next three sections the three fundamental sub-process and their aspects are described.

4.1. System Layered Abstraction

Before a system can be validated or verified it is important to know how this system is specified. In our own analysis of the development of complex systems and specifically in the aircraft industry there are three main phases for the specification. The first phase is the specification of the function required by the customers. The core activity of this phase is the formalisation of customer needs and the specification of the functions documented as the so-called function requirements which are naturally multi-system based. In the second phase the function requirements are to be allocated to different systems which are responsible for providing the system functionality required to fulfil the related function requirement. In the third phase the system components required for the manufacturing of the system are to be specified. The system components manufacturing is a very critical aspect due to the fact that external partners are contracted for the manufacturing of this component, they may have different points of view or understanding with respect to the tasks of the system component. Usually in the third phase cooperation between the industry and its suppliers will be established that leads to a detailed specification of the system component. In the first phase of the system integration of different system component suppliers the validity of the system component specification will be tested including the validation of the system and function requirements from which they are derived. The maturity of the system component specification is usually higher than the maturity of the parent requirements (original requirements which the current requirements are derived from) with respect to maturity, testability and feasibility. Based on these three development phases with different specification maturity, there is a need to handle each phase differently in our MAIP. In this thesis three layers are defined to perform the system abstraction. In each layer different entities have been defined to be able to perform the overall validation and verification activities. This layered abstraction has been represented by the CAM definition.

The CAM (Component Abstraction Model) is a representation of a component (subset of a system under test).This representation describes the interface characteristics (signals with their data properties), behaviour constraints and functional requirements (different behaviours regarding different system states, the transitions between the states and the performance requirements regarding the states and transitions). Signals model the physical I/Os available to a component. A signal is identified by a unique identifier by which it is identified on the test engine. The signal definition includes additional properties that are required for its configuration regarding the target test engine (a defined data type, a size (for variable length types), a default value and a defined value range).

The next sections describe the representation of this model, as well as abstraction layers to be used by the other entities (agents, CCM-component control model). The system layered abstraction is a logical definition that needs not to be matched to the physical topological definition resulting from the subdivision into multi-system functions done due to the installation requirements, systems and system components. Essential for system layered abstraction are the characteristics of the system to be tested and how this system under test has been specified. For the representation of the system under test regarding the modelling of its states, behaviours, physical architectures and functional requirements, three different test objectives have been analysed that cover the most important testing scope in the world of laboratory tests within the domain of aircraft testing activity:

- Simulation verification
- Controller verification
- Network data security testing.

With respect to these three test objectives and their testing requirements different representation methods have been developed to be able to provide agents with sufficient knowledge so that they are able to perform tests by defining test cases derived from this formally represented knowledge.

4.1.1. Expressions

Expressions are formally defined relations of signals, pre-defined constants (symbols) and a set of built-in functions, for the purpose of being validated and verified at runtime to check constraints or to calculate desired values. In the agent testing approach, each expression has a textual representation (e.g. “A + COS(C) + 12”). In this thesis a generic expression graphical editor has been developed for the generation of expressions including their time-constraints (time-constraints are linked to operators) that can be easily extended when new signal relations are required. In the scope of our agent-based approach two kinds of expressions for logical and mathematical operations have been developed, as shown in Figure 20 and Figure 21.

These expressions have been used to represent the functional requirements to be fulfilled by the system under test. If the functional requirements cannot be represented by an expression operator because of their complexity (rarely the case), an operator can be replaced by linking the external library (e.g. written in C) where the time-constraints can be defined, as well. Generally a constraint represents the relationship between a parameter and an expression, which can be evaluated at runtime to be validated or not. Time-constraints have been defined to represent the performance requirements of the system under test. Other constraints like linking the operations to the valid states or transitions have been defined within the state and transition context and linked to the respective operations, as described in the next section.

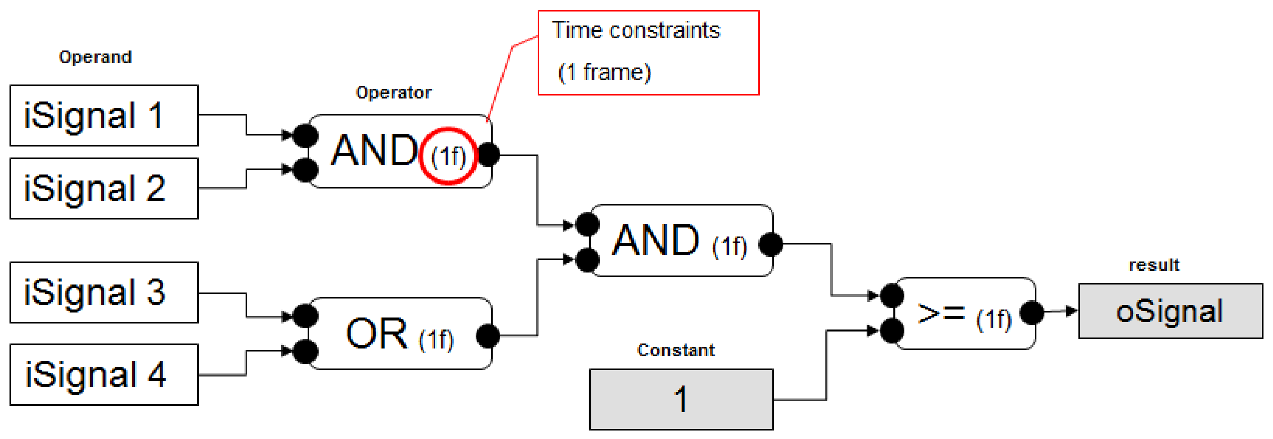


Figure 20 Logical Operation

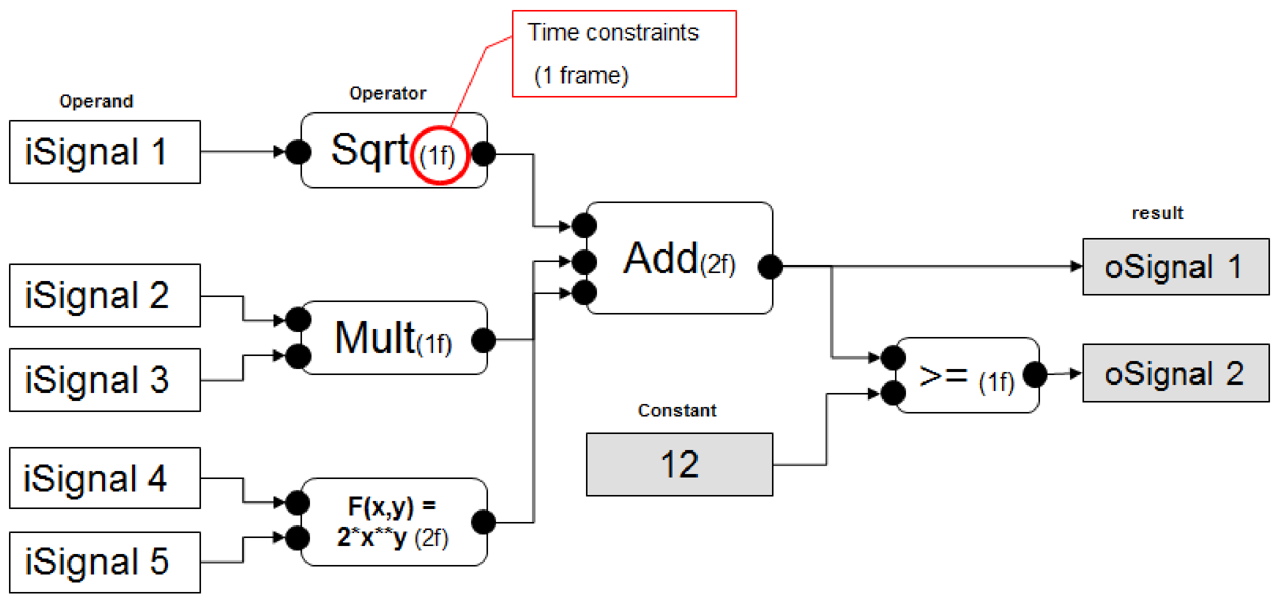


Figure 21 Mathematical Operation

4.1.2. States and Transitions

Some of the system and data security requirements are to be verified during certain system states or protocol activities. To enable agents to consider this, it is necessary to design a model wherein the test designer can represent the intended states of the system or network (in case of data security test). For this representation the following aspects are to be considered:

- The system component state is represented by a set of physical signals with their particular values so that any signal value change mostly leads to having different states (may be every frame).
- It is important to understand that identifying states logically with appropriate names and sets of physical signals is a logical design process done abstractly to physical states. While physical states always look the same, the reflection of a real system with logical state models always depends on the test intention, purpose and approach. To emphasize this, logical states are not directly based on physical signals, but on abstract signals (directly mapped to physical signals

or to a set of physical signals). In the next section the term signal has been used as logical signals.

- A State is defined by the following entities, as shown in Figure 22.

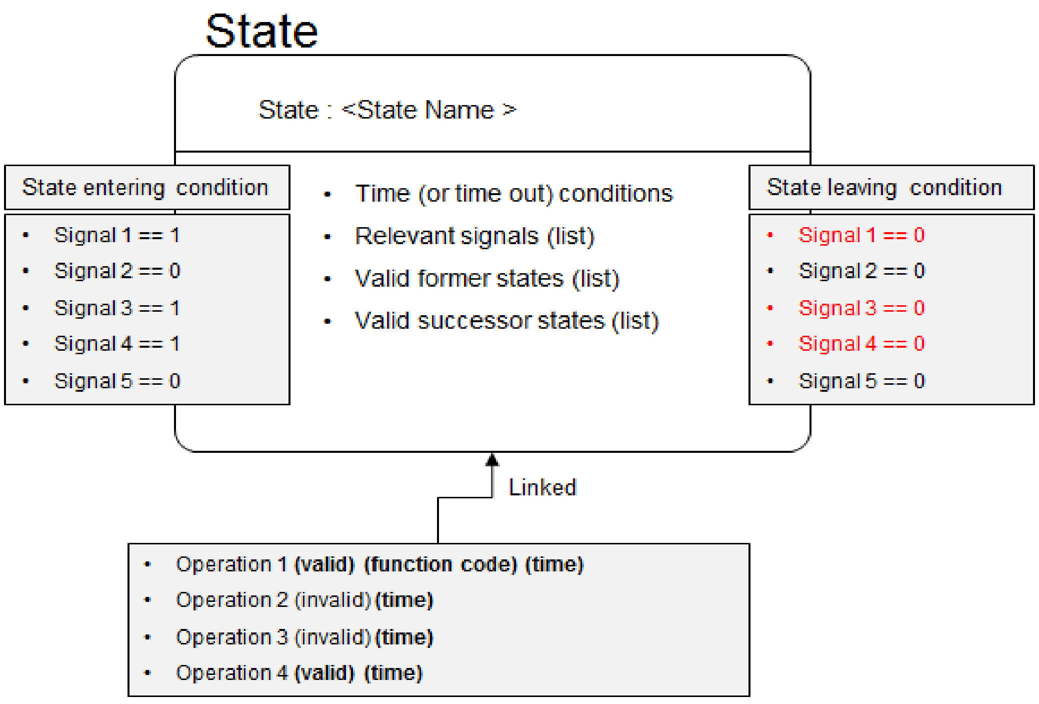


Figure 22 State

- State entering condition: a list of signals with their particular values (signal patterns) that define the entering condition of a certain state.
- State leaving condition: a list of signals with their particular values (signal patterns) that define the leaving condition of a certain state (for example in Figure 22 a deviation to the entering state relating to value changes of signals 1,3 or 4 lead to leaving of the state)
- The list of the system or component operations is linked to the state where three properties specify the type of this linking:
  - Valid: to define if the operation can be performed within this state (valid for yes and invalid for no).
  - Function code: to identify the involvement of a certain operation in a global system function (all involved operations shall possess the same function code).
  - Time: to define the time-constraints of the related operation.
- State Time: to define the time-constraints directly linked to the state.
- Relevant signals: a list of signals that affect this state to determine its signal context.
- Valid former states: a list of valid states from which this state can be achieved.
- Valid successor states: a list of valid states that can be achieved from this state.

When designing a test scenario, usually a certain sequence of states is in mind, where states may be valid in a defined sequence only. So it is required to define a new entity to represent the relation between states. In this thesis this relation has been called “transition”. A transition is a context where a state cannot be identified, but recognized to be the state to be reached later, and a former state has been left or not given anymore. Sometimes a state’s end condition is met at the same time as a new state’s begin condition; thus an empty transition should be modelled or acceptable.

Usually a state could have more than one valid successor state. To be able to model this, a transition should lead to one desired state regardless whichever the successor state was, where states could be followed by different transitions. In contrary to states, occurrence of transitions can only be identified when the state desired by the system is achieved because of the logical nature of transitions as links between states, containing no physical information so that their occurrence could be detected.

.

A transition is defined by the following entities, as shown in Figure 23.

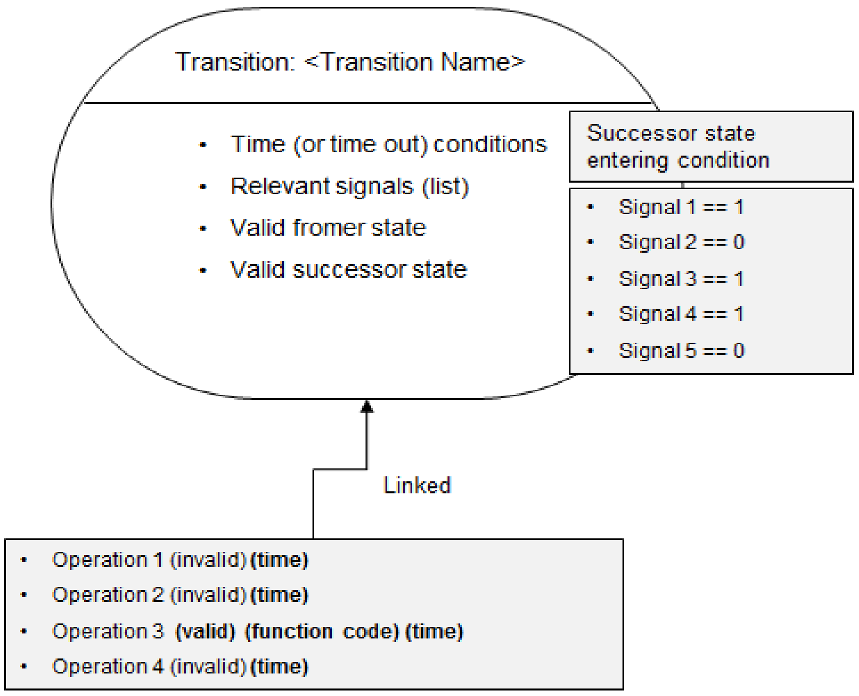

Transition: <Transition Name>
Time (or time out) conditions
Relevant signals (list)
Valid fromer state
Valid successor state
Successor state entering condition
Signal 1 == 1
Signal 2 == 0
Signal 3 == 1
Signal 4 == 1
Signal 5 == 0
Linked
Operation 1 (invalid) (time)
Operation 2 (invalid) (time)
Operation 3 (valid) (function code) (time)
Operation 4 (invalid) (time)

Figure 23 Transition

- Successor state entering condition: a list of signals with their particular values (signal patterns) that define the entering condition of the successor state.
- The list of the system or component operations is linked to the transition in the same way as is done with respect to the state entity.
- Transition time: to define the time-constraints directly linked to the transition.
- Relevant signals: a list of signals that are to be set by this transition and are depending on the intended successor state.
- Valid former state: a valid state from which this transition has been started.
- Valid successor state: a valid state that can be achieved by this transition.

4.2. Knowledge Representation

Based on the entities defined in Section 4.1 in this thesis, a new modelling language has been developed, the so-called “Coded Timed Abstract State Machine (CTASM)” for the representation of system requirements specifications to build the knowledge base of agents used for system validation and verification. The CTASM is one of the possibilities to specify a CAM and has been used in our work because of its generic characteristics and its capability to model complex behaviours for the system under test. Although there is no limitation to specify a CAM by using other modelling languages, in this thesis we have decided to use CTASM that covers all capabilities required for the testing activities considered in this thesis (see Section 4.1).

The CTASM modelling language is an extension of the Abstract State Machines ASM developed by [Rosenzweig 2000] that includes facilities for specifying non-functional behaviour, namely time and resource consumption. In the complex system development, the verification of systems is based on the accuracy and correctness of four implemented system entities- function, interfaces, state and performance. The objective of the CTASM modelling language and its graphical representation is to provide a framework for the modelling of agent knowledge required for the Agent Based System Verification where these four key entities can be represented and analysed. In this section we begin the description of the modelling language with an analytical survey of the use of ASM in verifying complex systems. The core difference between the CTASM and the ASM is that states (see Section 4.1.2) in the CTASM are signal pattern based and time dependent instead of being instantaneous and time triggered. This paradigm captures the realistic behaviour of system components to be verified where actions are never instantaneous. The semantics of the CTASM is the behavioral description of system component functions with respect to the given instantaneous signal pattern (system component input-output value set).

Requirements-based engineering is an approach to the engineering of hardware and software systems where engineering is conducted with the help of structured requirements documents. Requirements documents represent high-level abstractions that are used to represent and analyse system designs throughout the engineering lifecycle of the system based on the customer needs. More specifically, system requirements-based designs are solution independent and can be analysed and validated during the early stages of the lifecycle, before the system is implemented. The philosophy of requirements-based engineering relies on the economics of shifting the implementation responsibility to the supplier where they are free to select the implementation methods in case all requirements have been fulfilled. Relying on this, economics demands new testing and integration approaches to ensure that the right product with the correct functionality has been realised. In the regular requirements-based engineering model driven designs or prototypes are usually not used; the product to be manufactured cannot be fully validated or verified before it has been implemented. If the requirements are formally specified, the analysis, in order to validate or verify them, can be automated. For example, consistency, testability, feasibility and completeness were identified as useful properties of specification that can be validated or verified automatically. Furthermore, the use of a formal requirements specification method can automate engineering testing activities such as test preparation, test case generation and executions [Lundqvist 2008].

Computing systems are usually real-time and frame based systems which are permanently interacting with the environment. The correctness of such systems is defined as the system demonstrating correct behaviour in its continued interaction with the environment and with respect to the real-time capability within an acceptably bounded amount of time. Controllers for aircraft systems are examples of frame based, real-time and embedded systems, due to the fact that embedded systems are typically limited in the amounts of resources that they can utilise and in the time given to perform certain functionalities. The verification of an aircraft controller depends on three key factors – the functional correctness according to defined operation states, appropriate response times and adequately bounded resource utilisation. Aircraft systems are typically of the safety critical nature, so that incorrect behaviour could lead to a serious problem. For such systems, optimised and smart test methodology can provide desired benefits in terms of improving the implementation and getting mature systems.

Abstract State Machines (ASM) have been used to specify, analyse and verify hardware and software systems at different levels of abstraction. Abstract State Machines can also be used to automate engineering activities, including model validation, test case generation and test execution [Grieskamp 2002]. The Timed Abstract State Machine (TASM) language is an extension of the

ASM language that includes facilities for specifying non-functional properties, namely time and resource consumption [Ouimet 2007]. The Coded Timed Abstract State Machine (CTASM) is another extension of the ASM model that includes capabilities for modelling functional properties whose behaviors depend on the state and time context in which they are operating. The goal of the CTASM is to provide a state-based modelling approach to engineer controller models, where the four key aspects of system behaviour can be specified, analysed, and traced from the initial design stage all the way through model maintenance. This state based model can be used to provide software agents [Zambonelli, Jennings, and Wooldridge 2003], designed for performing system validation and verification activities, with the required knowledge about the system to be validated or verified.

4.2.1. System component knowledge representation

For the representation of system components as the first abstraction layer a simplified controller of the aircraft door management system (DSC) has been selected to explain how system components can be represented by using CTASM for the CAM specification. The DSC controller is responsible for the management (monitor and controlling) of the real aircraft door. The following engineering steps (*the only engineering activity that is not supported by agent approach in this thesis*) have been performed for the representation of the DSC CTASM:

<u>Assumption</u>

- The physical sensors of the doors are connected to the test system I/Os and correctly configured (including the definition of their mapping signals). The signal mapping definition is required to be able to interact with the I/Os of the test system during testing.
- The DSC controller is physically connected to the test system I/O and correctly configured.

<u>Step 1</u>

Identification of the signal scope (sub-set of the test system signals) that is to be linked to the CTASM.

<u>Step 2</u>

Definition of system component operations by using the expression editor (including the linking to the external library, see Section 4.1.1). An operation is an elementary function of the system component which can be defined to represent a relationship between values (usually input and output parameters). Each of its input values gives back exactly one output value. The validity of operation is state oriented and depends on the states in which the operation is performable. For operations which are globally valid no state dependency definition is required.

<u>Step 3</u>

Definition of system component states:

- Definition of the state entering condition by identifying the sub-set of affected signals and their particular values.
- Definition of the state leaving condition.
- Linking of the system component operations including the definition of the linked attributes (valid. function-code and time, see Section 4.1.1).
- Definition of the state time-constraints.
- Identification of the state relevant signals (sub-set of the CTASM single scope) that are affected by a certain state.

In this thesis, the calculation of responses has been defined by the operations. A state can be related to the system or to its state representation by its control system. At each instant of time, a dynamic system is in a specific state represented by its signal pattern. State_systems can have one or many state variables: at any time, the system's state is defined as the unique value for each of the state variables. State machine systems can be modelled with one or many concurrent state machines: at any time, each of the concurrent state machines must be in one and only one state. A state is a unique snapshot that is specified by values for a set of variables, characterises the system for a period of time and is different from other states. Each state is different from other states in the inputs it responds to, the outputs it produces or the transitions it takes. A transition is a response to an input that may cause a change of state.

Step 4

Definition of system component transitions:

- Definition of the successor state entering condition by identifying the sub-set of affected signals and their particular values, whereby the relevant signal list is defined implicitly.
- Linking the transition to the former and successor states.
- Linking of the system component operations including the definition of the linked attributes (valid function-code and time, see Section 4.1.1).

A transition is a progression from one state to another and will be triggered by an event that is either internal or external to the entity being modelled. In this thesis events are triggered by signal value changes caused by operation or external stimulation. For software agents which handle system validation and verification tasks, transitions are typically the result of the invocation of an operation (stimulated by software agents) that causes an important change in state, although it is important to understand that not all method invocations will result in transitions. Specifically for controllers, the reflected state of the system under control can be changed, i.e. a system component operation triggered by software agents must not lead to a change of its own state. A graphical representation of the CTASM for the door opening function of the DCS controller is shown in Figure 24.

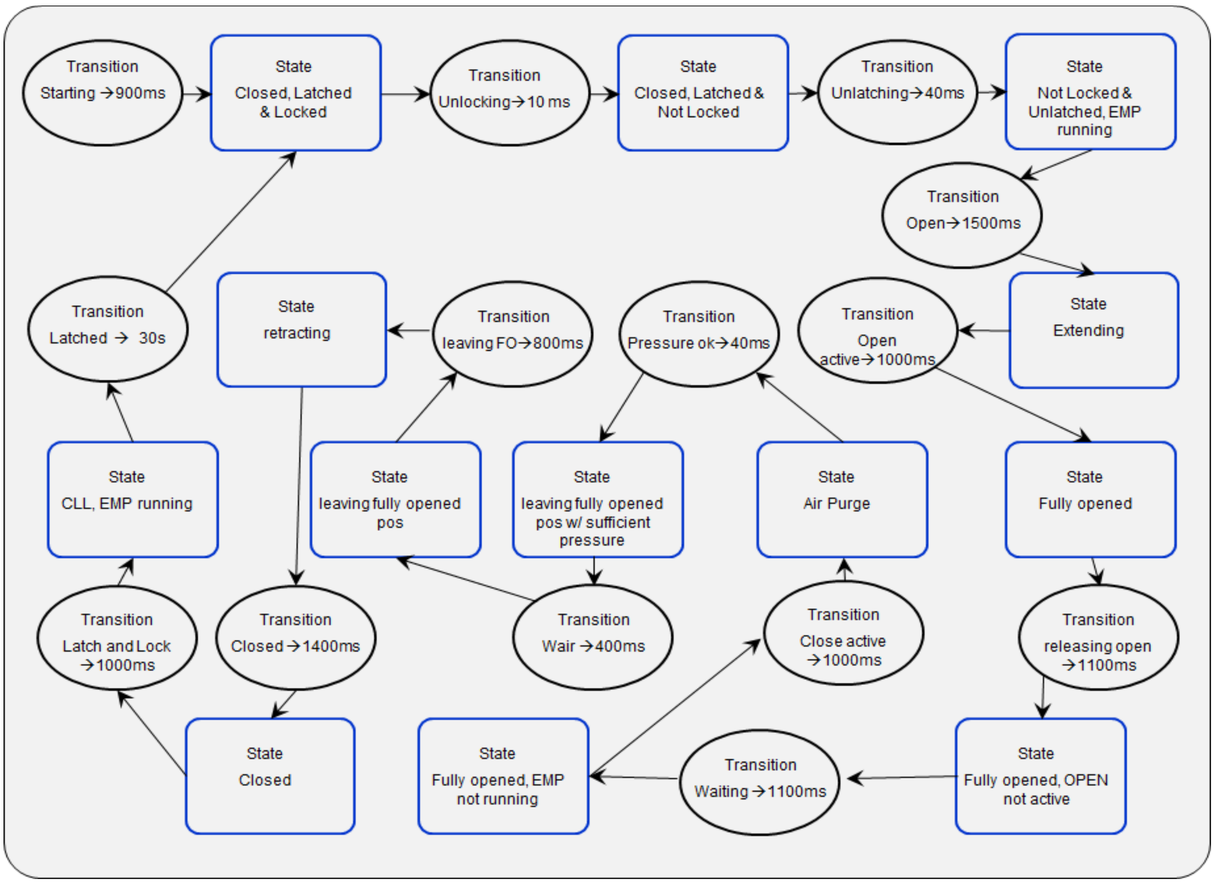

Transition Starting →900ms
State Closed, Latched & Locked
Transition Unlocking→10 ms
State Closed, Latched & Not Locked
Transition Unlatching→40ms
State Not Locked & Unlatched, EMP running
Transition Open→1500ms
State Extending
Transition Open active→1000ms
Transition Pressure ok→40ms
Transition leaving FO→800ms
State retracting
Transition Latched → 30s
State CLL, EMP running
State leaving fully opened pos
State leaving fully opened pos w/ sufficient pressure
State Air Purge
State Fully opened
Transition Latch and Lock →1000ms
Transition Closed→1400ms
Transition Wair→400ms
Transition Close active →1000ms
Transition releasing open →1100ms
State Closed
State Fully opened, EMP not running
Transition Waiting→1100ms
State Fully opened, OPEN not active

Figure 24 CTASM

Using CTASM within an agent based testing approach

With respect to the generic agent model architecture (see Figure 8), the following steps are to be performed to define the agent that can use the defined CTASM:

Step 1

Identification of the agent helpers required to perform the test. In the case of the door opening function, four agent helpers have been identified:

- Operation verifier: a generic agent helper with the capabilities to verify a system component operation based on the their defined expressions.
- Transition trigger: a generic agent helper with the capabilities to set a list of signals based on the transition defined in a certain CTASM.
- State monitor: a generic agent helper with the capabilities to check if the intended state has been reached and all validity and time-constraints are fulfilled.
- CCM trigger (component control model, a specific test engine interface): a specific agent helper responsible for establishing the remote control connection to a certain test engine platform.

Within the CTASM an operation that represents the relationship between a set of inputs and a set of outputs is a coded state or transition (function-code, see Section 4.1.2). The operation coding is the definition of the validity and visibility context of this operation. Within the CTASM the operation can be considered as globally valid and can be verified by software agent states and transitions independently, if no links are defined to an operation.

Step 2

For the configuration of agents (one agent in the case of the door opening function), the following steps are to be performed:

- Linking of the agent instance (see Figure 8) to the defined CTASM
- Linking of the agent processor to the three defined agent helpers.

Step 3

Preparations required for agent activity are complete so that the test can be executed and monitored by using it.

The Coded Timed Abstract State Machine Mark-up Language

The key improvement of the TASM compared to the ASM is the duration of steps. In the ASM, machine steps are instantaneous. The key difference between the Coded Timed Abstract State Machine (CTASM) and the TASM is that the timed steps are divided into three time constraints managed by three different schedulers. All schedulers are managed by test agents and are defined with respect to the given time constraints regarding operations, states and transitions. With respect to the given time constraints the agents are able to define the testing scenarios and the required time oriented passed / failed criteria. Furthermore, in the CTASM, lasting and differently scheduled steps consume a limited amount of resources. In the case of single agent specifications, the scheduled steps of the agent dictate the progression of time in the specification. In the case of multi-agent specifications, the scheduled steps are used to synchronise agents with respect to one another and with respect to the underlying operations, states, and transitions to be tested. The specification of time constraints of operations, states and transitions is achieved through annotations of individual rules. The concrete syntax of the CTASM presented in Appendix D is an extension for time constraints and resource annotations defined by the ASM.

4.2.2. System knowledge representation

A system is a logical composition of system components whose design and development is managed by system designers and whose represented entities are executed on a test engine. For the definition of system models a specific CTASM has been designed which has been called sysCTASM in this thesis. A sysCTASM is a logical group of system components represented by their CTASMs with their own states, transitions, operations and time constraints.

System State Signals

A system state signal is an entity that describes a state or a transition on the system component level via CTASM interface description so that each state or transition on the CTASM level can be represented as a system state signal (sysCTASM signals) and mapped to the sysCTASM signal scope, as shown in Figure 25 for a simplified DSC-Panel (Doors System Control).

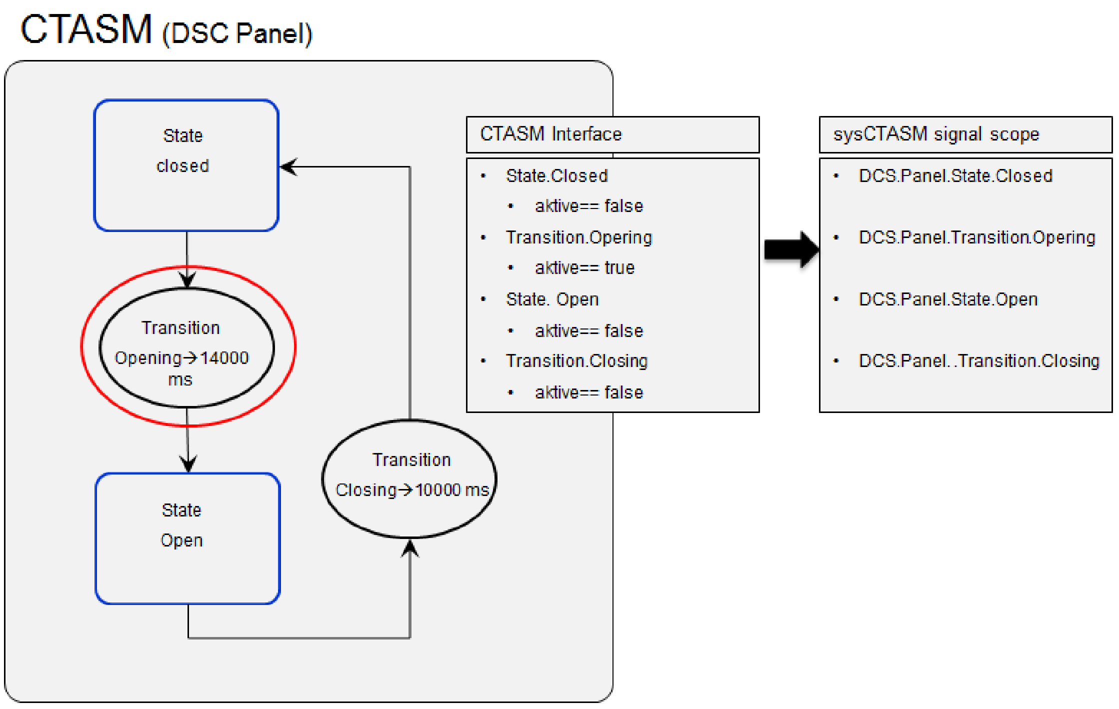


Figure 25 System state signal

System Functions

Based on the definition of the system component operation and its link to a function-code (see Section 4.2.1), a system function has been defined as a sequence of system component operations. System functions in this thesis have been defined by using the same expression methodology as used on the system component level (see Section 4.1.1) with the following customisations:

- System component operators are to be replaced by system component operations (the so-called system operator in this thesis is an abstract definition of the system component operators. Figure 26 shows this abstraction based on the mathematical operation shown in Figure 21), i.e. a black-box consideration of the system component operations.
- The time-constraints are to be redefined for the operation level (based on the time-constraints of the operators, e.g. in the most simple form the sum of the operator time-constraints).

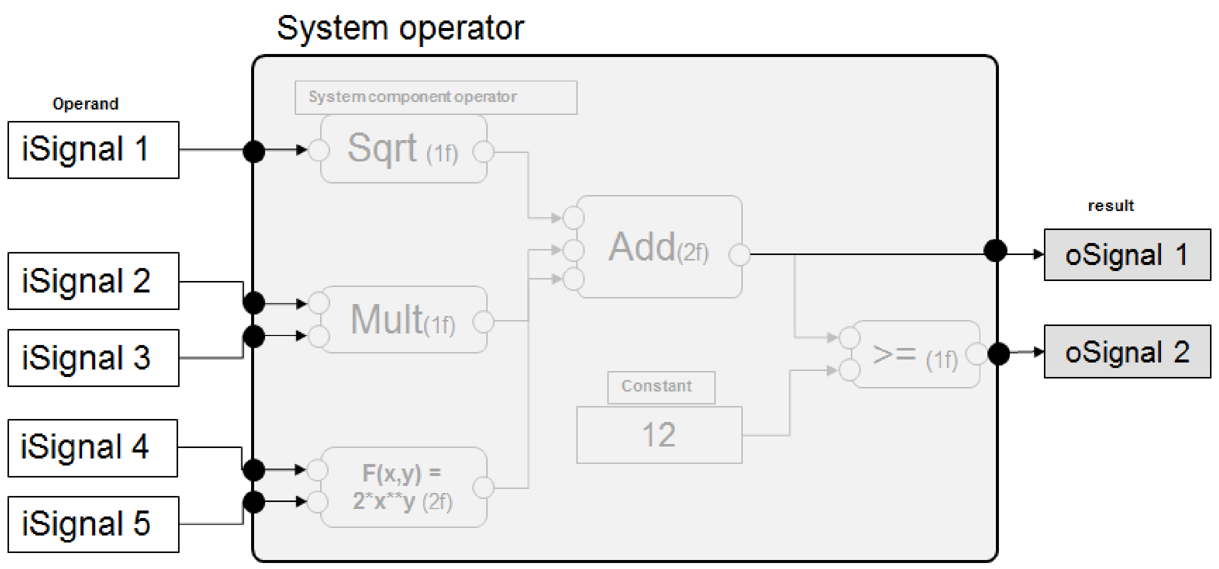


Figure 26 System operator

<u>System State</u>

Unlike a system component state which is represented by a certain signal values pattern, a system state is represented by a certain state pattern of the system components hosted by this system. For this representation the following aspects are to be considered:

- The system state is represented by a set of system component states and transitions (system signals) with their particular entities.
- A state is defined by the following entities, as shown in Figure 27.

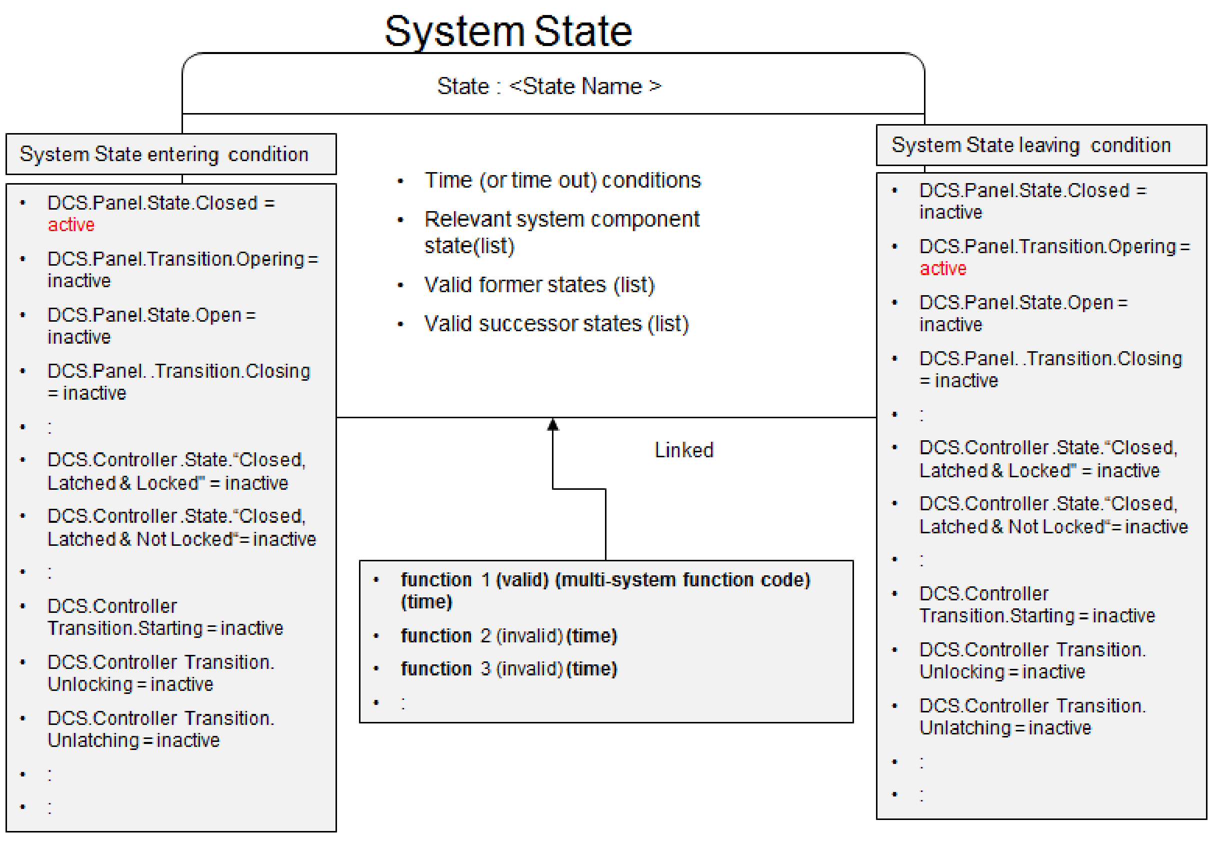

System State
State : <State Name >
System State entering condition
DCS.Panel.State.Closed = active
DCS.Panel.Transition.Opering = inactive
DCS.Panel.State.Open = inactive
DCS.Panel. .Transition.Closing = inactive
:
DCS.Controller .State.“Closed, Latched & Locked” = inactive
DCS.Controller .State.“Closed, Latched & Not Locked”= inactive
:
DCS.Controller Transition.Starting = inactive
DCS.Controller Transition. Unlocking = inactive
DCS.Controller Transition. Unlatching = inactive
:
:
Time (or time out) conditions
Relevant system component state(list)
Valid former states (list)
Valid successor states (list)
Linked
function 1 (valid) (multi-system function code) (time)
function 2 (invalid) (time)
function 3 (invalid) (time)
:
System State leaving condition
DCS.Panel.State.Closed = inactive
DCS.Panel.Transition.Opering = active
DCS.Panel.State.Open = inactive
DCS.Panel. .Transition.Closing = inactive
:
DCS.Controller .State.“Closed, Latched & Locked” = inactive
DCS.Controller .State.“Closed, Latched & Not Locked”= inactive
:
DCS.Controller Transition.Starting = inactive
DCS.Controller Transition. Unlocking = inactive
DCS.Controller Transition. Unlatching = inactive
:
:

Figure 27 System state

- System state entering condition: a list of system component states with their particular entities (system component state/transition patterns) that define the entering condition of a certain system state.
- State leaving condition: a list of system component states that lead to leaving this state)
- The list of the system functions that is linked to the state where three properties specify the type of this linking:
  - Valid: to define if the system function can be performed within this state.
  - Multi-system function code: to identify the involvement of a certain system function in a global multi-system function (all involved operations shall possess the same multi-system function code).
  - Time: to define the time-constraints of the related system functions.
- Relevant signals: a list of signals that affect this state to determine its system component state context.

4.2.3. Function knowledge representation

A function is a logical composition of systems whose design and development is managed by function designers and whose entities are executed on the MAIP. The reason for the execution on the MAIP rather than on the test engine is based on the nature of the function which possesses no physical resources of different system components but allocates them. The allocation of the physical resources is defined by the function (function CCM) and the configuration is delegated to the hosting systems (system CCM). For the definition of function models a specific CTASM has been designed which has been called funCTASM in this thesis. A funCTASM is a group of systems represented by their sysCTASMs with their own states, transitions, operations and time constraints. Analogous to the system, the function entities can be defined:

Like the system state which is represented by a certain system component state pattern, a function state is represented by a certain state pattern of the system allocated to this function. The allocated systems are called function involved systems in this thesis.

For some validation and verification activities the co-existence of the funCTASM, involved sysCTASMs and related CTASMs is essentially required to allow concurrent testing on these three layers. To be able to manage this co-existence three pre-processing units have been defined. The pre-processing units are responsible for the preparation of the function, system and system component representations to provide the CCM-entities (see Section 3.9) with the required information to configure the test engine and test environment.

4.3. Context Based Learning

In this thesis it has been proposed that learning agents be incorporated into MAIP environments in order to generate or modify the CTASMs, sysCTASMs and funCTASMs to model the adaptive behaviour of agents. These learning agents adapt to specific environment and events during the test run.

They would select tasks to be accomplished among a given set of tasks as the testing progresses, or synthesize tasks for themselves based on their observations of the environment and on information they may receive from other agents. We investigate an approach in which agents are assigned goals when the test run starts, and then pursue these goals autonomously and adaptively. During the test, agents progressively improve their ability to accomplish their goals effectively and safely. Agents learn from their own observations and from the experience of other agents with whom they exchange information. Each learning agent starts with a given representation (CTASMs, sysCTASMs and funCTASMs, see Section 4.2) of the testing environment from which it progressively constructs its own internal representation and uses it to make decisions. This section describes how learning methods can support this approach, and shows that context-based learning may be effectively used in this scope. Different approaches have been used with respect to the linked CTASM entities.

Methods of learning and their limitations

Based on the learning methods analysed by [Althoff 2011], Figure 28 shows the inductive interference methods to be discussed in this thesis:

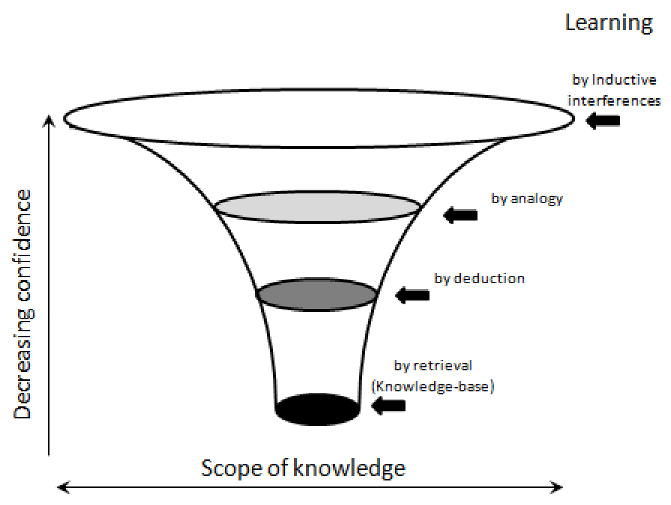


Figure 28 Agent learning methods from Althoff

Knowledge-based retrieval

Specifications and requirements created by system designers are represented in this thesis by using CTASM and its derivates (sysCTASM and funCTASM) to provide agents with the required knowledge during their tasks performed in order to achieve their goals. Like [Althoff 2011], this knowledge is classified in this thesis as confident and does not need to be validated during the agent-based testing.

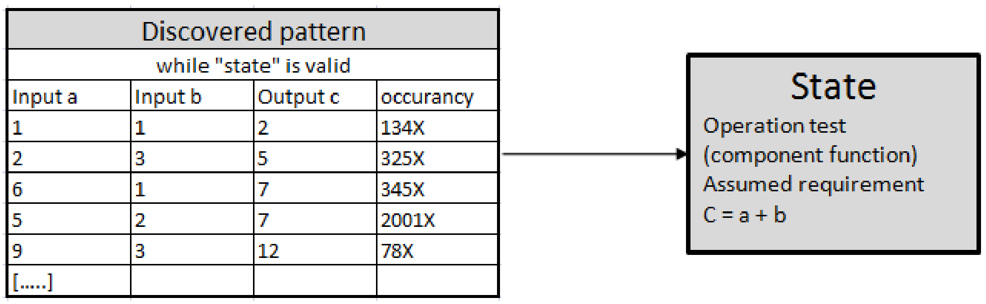
Deduction

This method has been used for pattern discovery (black box consideration) within system component context and involved system discovery within function context:

- If P ∊ Q → R is a valid knowledge, P and Q are discovered, and then R is gathered knowledge (analytical learning, no new domain specific knowledge is gathered).
- If discovered knowledge by observation is (1,2) (4,8) (7,14) (47,94), then (X, 2*X) is learned knowledge (synthetic learning new knowledge has been advised).

This knowledge is classified in this thesis as non-confident and must be validated by knowledge establishing processes before it can be added to database (see Section 3.8.3).

The deduction method has been used in this thesis to define an expression based on recorded or monitored data during the test run, as shown in Figure 29. The basic mathematical and logical expressions are pre-defined and represent the persistent operation pattern that can be used to compare the detected behaviours with the pre-defined one. Generally depending on the required complexity, agents can be defined to combine different operation patterns to be able to detect more complex behaviours. In this thesis more complex behaviours are represented by expressions defined by users or linked to external libraries.


Discovered pattern
while "state" is valid

Input a	Input b	Output c	occurancy
1	1	2	134X
2	3	5	325X
6	1	7	345X
5	2	7	2001X
9	3	12	78X
[.....]			

Figure 29 Simplified function discovery

Analogy

This method has been used for function procedure discovery within function context to detect similar scenarios which lead to the same results. Analogy has been used to reduce the test case number to be performed by the V&V activities; similar procedures with same points of control and points of observation are declared as irrelevant tests and do not need to be performed again:

- If P → R is a valid knowledge, P' is discovered and is similar to P, then R is learned knowledge (Learning by analogy).

This knowledge is classified in this thesis as non-confident and must be validated by knowledge establishing processes (see Section 3.8.3).

Pattern discovery

By observing a system over a sufficient period of time (during testing or using recorded data) a signal pattern can be identified periodically that could be compared with a given state entry condition, as shown in Figure 30, to identify that a certain state has been reached.

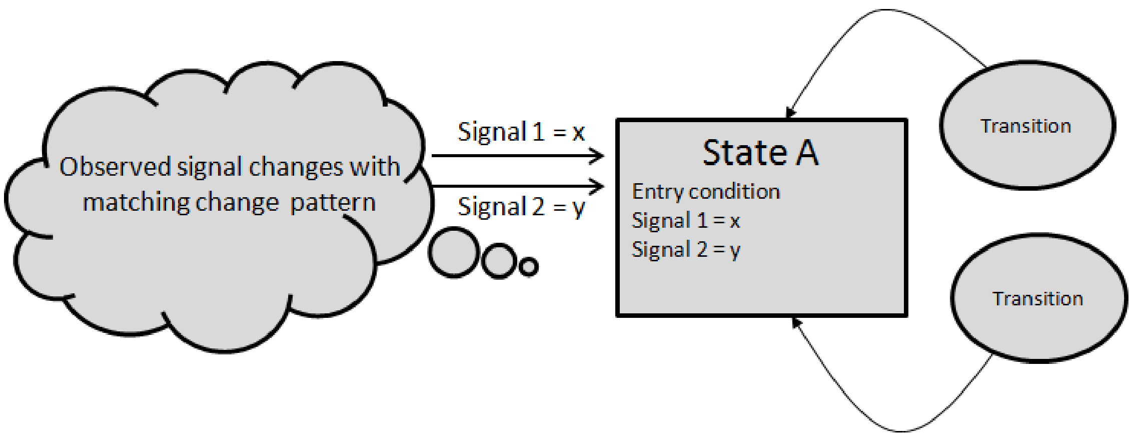


Figure 30 Simplified pattern discovery

State completion

Based on the pattern discovery method, an agent could find the transitions between states, including all particular signal values occurring that have led to this transition, as shown in Figure 31.

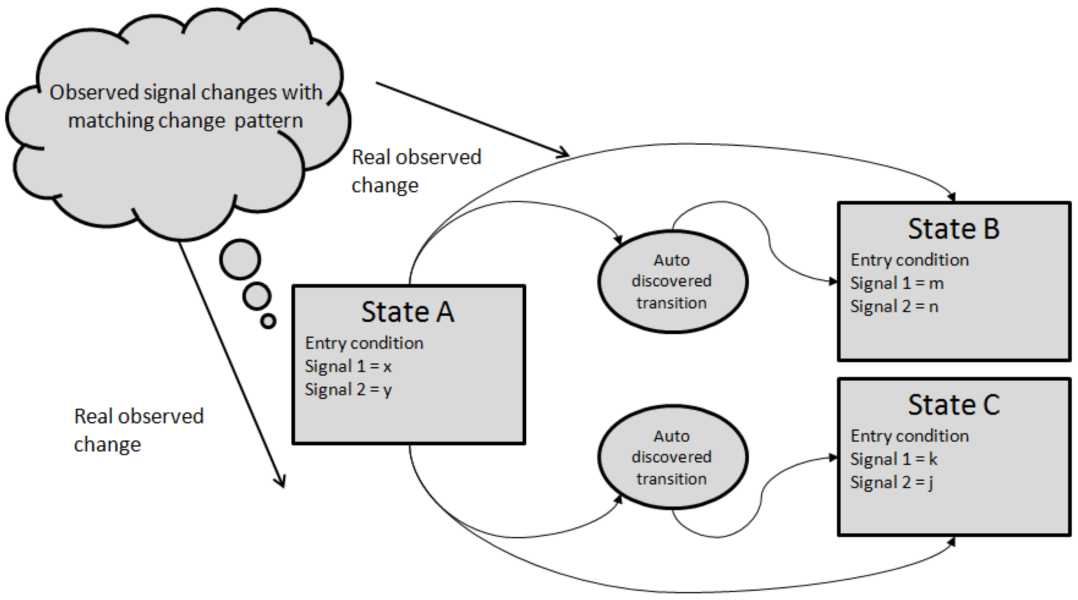


Figure 31 Simplified state completion

For the illustration of some aspects of the learning methods, the DSC Controller example (defined in Section 4.2.1) has been used; therefor the following steps have been performed:

Step 1:

Based on the specification of the system a CTASM has been created, as shown in Figure 32, that represents the expected and specified behaviours of the DCS system.

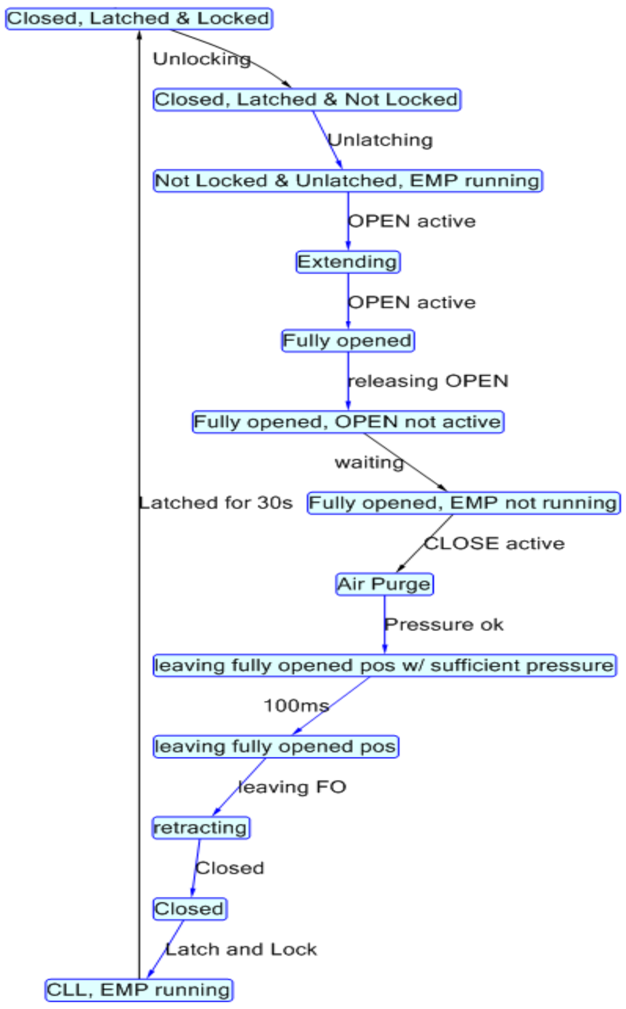


Figure 32 expected CTASM

<u>Step 2:</u>

Recording of real data during an open door scenario in the real aircraft:

- Triggering of the open door function has been done manually by using the open door panel.
- The sensor data have been recorded.
- The DCS controller data have been recorded as well.

<u>Step 3:</u>

Comparing the recorded data (real behaviours) with the expected one defined by the CTASM, as shown in Figure 33. The following observations have been identified (amongst other things):

- There is a transition between the state “Air Purge” and the state “Closed, Latched & Locked” that is not specified and represented within the given CTASM.
- This identified transition possesses five different signal patterns, before the entry condition of the state “Closed, Latched & Locked” is identified.

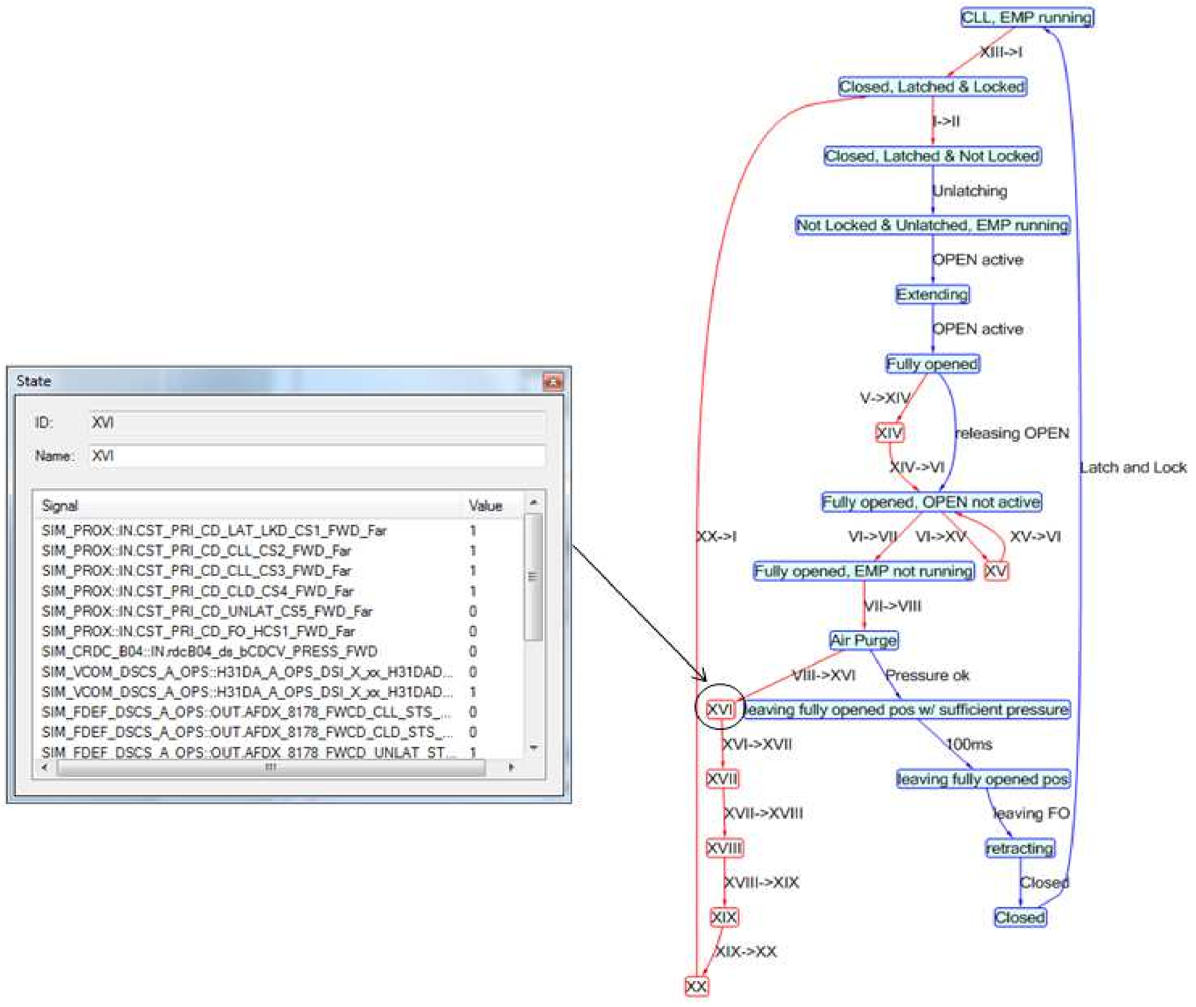
<u>Step 4:</u>

Evaluating the given results has yielded the following statements.:

- At least one transition has not been specified.
- Four signal patterns are not linked to any transition or state although they were required for having the fifth signal pattern that has led to entering a new state.
- This deviating behaviour is systematic and reproducible.

<u>Step 5:</u>

The validation of the given results with the system designers to modify the CTASM or to correct the specification has been done. This validation task is an engineering task and will be performed by reviews usually.



State

ID: XVI

Name: XVI

Signal	Value
SIM_PROX::IN.CST_PRI_CD_LAT_LKD_CS1_FWD_Far	1
SIM_PROX::IN.CST_PRI_CD_CLL_CS2_FWD_Far	1
SIM_PROX::IN.CST_PRI_CD_CLL_CS3_FWD_Far	1
SIM_PROX::IN.CST_PRI_CD_CLD_CS4_FWD_Far	1
SIM_PROX::IN.CST_PRI_CD_UNLAT_CS5_FWD_Far	0
SIM_PROX::IN.CST_PRI_CD_FO_HCS1_FWD_Far	0
SIM_CRDC_B04::IN.rdcB04_ds_bCDCV_PRESS_FWD	0
SIM_VCOM_DSCS_A_OPS::H31DA_A_OPS_DSI_X_xx_H31DAD...	0
SIM_VCOM_DSCS_A_OPS::H31DA_A_OPS_DSI_X_xx_H31DAD...	1
SIM_FDEF_DSCS_A_OPS::OUT.AFDX_8178_FWCD_CLL_STS_...	0
SIM_FDEF_DSCS_A_OPS::OUT.AFDX_8178_FWCD_CLD_STS_...	0
SIM_FDEF_DSCS_A_OPS::OUT.AFDX_8178_FWCD_UNLAT_ST...	1

Figure 33 Real DCS behaviour

Usually by the transition of a system specification to its implementation, models provide a high-level abstraction of the system without the complexity of the implementation details. Models are a powerful tool that can be used for reasoning about the system and models can be checked for satisfiability of given properties, and for estimation of the performance of the implemented system. Thus, for it to be of any use, it is important that the model correctly reflects the system specification.

4.4. Conclusion

On the basis of the generic definition in Chapter 3 the concepts of the specific implementation of the respective entities have been defined in this Chapter.

For the system abstraction layer it was necessary to transfer the test engines as well as the used agents into the layer definition, in addition to the components to be tested. This implementation of all entities involved (system under test, test system and agents) in the definition allows to provide the layer edges with an agent interface so that the communication between the respective layers can be integrated in the agent model. The agent interfaces are developed so that they can assume the tasks of controlling and monitoring the entire data traffic as well, whereby the provision of the information encapsulated in the layer can be secured. The generic CAM defined in Chapter 3 is enhanced by the data of the other entities within its layer to enable it to encapsulate and to represent its layer per interface.

For the system component model the CTASM (Coded Timed Abstraction State Machine) has been defined as a specific implementation for the TAM to represent the behaviour of the system components to be tested as component knowledge (see Section 4.2.1). The component knowledge has been reproduced by states and state transitions which are dependent on time and time-oriented incidents (timed). During a certain state or state transition certain operations of the system components are valid and can be executed. The operations of the system components are provided with a code, whereby they are linked to the state and its transition (coded).

The CTASM as an extension of the ASM enables the capability of agents to verify additionally the context-dependent coded operation of the system components during the testing of states and state transitions. For systems it was necessary to develop a specific solution (sysCTASM), since the composition of all comprised system components was not sufficient (see Section 4.2.2). The reason therefore was that the sysCTASM and its preserving sysCAM (hosting sysCAM) have to fulfill additional tasks to sufficiently represent the system layer. New states, operations and interfaces are defined on the system layer so that agents can interact within this layer and can communicate with the underlying layer of the system components via the interfaces. By these interfaces controlling of the inter-communication was transferred to the agents whereby they can guide the whole procedure and its processes during verification activity.

For the functions it was required to specifically implement the CTASM, since their entities cannot be executed on the test engine, but in the MAIP environment only (see Section 4.2.3). Since functions represent the interaction between different system components of different systems, a composition of systems – represented by its sysCTASMs – cannot simply lead to its representation. For functions the funCTASM has been defined which represents the function layer with its hosting funCAM.

Since all three entities are separated and the inter-communication with the agents is ensured, the knowledge can be exchanged and modified under the agents' control via all layers.

For adaptivity and learning, different learning processes have been developed for the different layers. For the system components a process has been defined which has allowed us to complete the CTASM definition by pattern discovery and context observation as well as to create it from scratch. With this capability the verification process can be started even though the CTASM definition is not complete or mature. Thereby it is presumed that the interfaces of the system components are known.

For systems it is a precondition that learning on the system components level is activated so that the compositions of the states and of the operations can be calculated again and again.

For multi-system functions a new learning process has been defined. By the scenario-oriented definition of the function – where the involved systems and system components can be configured in manifold compositions – learning can be employed effectively to define a new function constellation. By flexible communication of the agents and the layered abstraction including the layer interfaces, learning can be controlled via all three layers by agents.

PART III
IMPLEMENTATION

PART THREE - IMPLEMENTATION

Part Three of the thesis describes the implementation of the MAIP as a complete solution for the agent based system validation and verification approach. Three different use cases are described to validate this approach by using it in different V&V domains.

This final chapter provides a summary of the major contributions of this dissertation. While this dissertation provides contributions to the field, it also raises new questions. Therefore, this chapter concludes with recommendations for future work.

CHAPTER FIVE CASE STUDY

5. Case study evaluation

For the approach evaluation discussed in this thesis and evaluated by ABT-Project the following other approaches have been chosen, as they comprise reference approaches:

- Requirement-based testing (RBT): Based on the system requirements, the test designer defines the test requirements with their pass / fail - criteria. For each test requirement a test case has to be implemented and executed to validate or to verify it. One of the major challenges in test case design is the simple fact that any system, except the most trivial ones, has more possible combinations and permutations of the inputs/states than can be handled by this approach. Another major challenge is the ensurement of the re-usability with respect to the different maturity levels and configurations of the system under test.
- Model-based testing (MBT): The use of a model to describe the behaviour of a system is a proven major advantage for test development teams. Models can be utilised in many ways throughout the product life-cycle, including improved quality of specifications, code generation, reliability analysis, and test generation. The prerequisite for using model-based testing is the availability of mature system models, which is the major problem and challenge of this approach.
- Script-based testing (SBT): is the classical way to perform test cases implemented to automate the generation of test cases, test configurations, test execution and evaluation. The main challenge here is the need for reduction of time, cost and enhancement of test coverage.

Research Question Evaluation Items:

To be able to answer the research question, the following evaluation items have been defined and agreed with the APT-project:

- Relevance of test cases: a test case is relevant if it is linked to at least one system requirement listed in the request for test.
- Applicability of test cases: a test case is applicable if and only if its signals are interfaced and known within the context of the test system (the test engine ADS from the company TechSat in this case).
- Real-time capability: a test case is real-time capable if no real-time violation is indicated by the test system.
- Environment: in the nature of aircraft multi-system testing the environment from the point of view of the system (to be tested) is highly mutable, collaborative and competitive.

Study scope:

The use cases have been implemented based on three requests for tests defined for different systems within the system controller domain. In this thesis other domains are not taken into consideration. The requests for tests selected for this study describe all test objectives required for the release for “entry into service” maturity gates defined for the Airbus cabin development (A350 cabin development). The test objectives defined in the test requests are used to set up the test coverage factor based on the following:

- CF = Coverage factor.
- ET = Number of successfully executed and feasible (testable objectives within a given time and cost) tests. A successfully executed test fulfills **all evaluation items** defined in the previous section except the relevance of test case.
- RT = Number of requested tests. Only requested tests are **relevant** and considered for the evaluation.

Then CF = ET / RT.

For the validation of the MAIP approach the following aspects have been evaluated with respect to the main improvement driver defined by the ABT-project by implementation of our use cases:

- Test coverage: the test coverage with respect to the capability to perform requested test objectives within a given time and limited budget.
- Test preparation: the indication of the required time and complexity for the preparation of test configuration, implementation and validation including test environment.
- Test generation: the complexity required to port the implemented test cases into a certain test engine platform.
- Test execution: the complexity and required know-how to be able to perform the test.
- Test evaluation: the capability to assess and adapt test results during the test and the offline test results analyses.
- Test re-configurability: the capability to use implemented test cases in different test configurations and environments without requiring re-implementation of them.

5.1. Requirements Based Testing

5.1.1. General Description

All test approaches used for the evaluation of the case studies are generally based on the requirements based testing. We have considered the classical way which has been called RBT and its extension supported by scripting for the automation of the testing tasks (called SBT in this thesis), by system modelling (MBT) and by agent based testing (= the current research subject) (ABT). In the following section the generic requirements-based testing process and how this process has been supported by the respective extension has been described.

Based on the definition of [Bender 2009], the requirements-based testing (RBT) process addresses two major issues: firstly, validating that the requirements are correct, complete, unambiguous, and logically consistent; and secondly, designing a necessary and sufficient (from a black box perspective) set of test cases from those requirements to ensure that the design and code fully meet those requirements. The second issue is called verification in the RBT process. In designing tests two topics need to be taken into account: reducing the immensely large number of potential tests down to a manageable size set and ensuring that the tests receive the right answers for the relevant reasons.

The overall RBT process serves to integrate test activities throughout the development life cycle and to focus on the maturity and quality of the Requirements Specification. This leads to early detection of design and specification errors which are very expensive when found during integration testing or later. The RBT process also ensures defect prevention, not only defect detection. To enable a correct implementation of the RBT process, verification activities can be divided into the following main activities based on the definition of [Bender 2009], which are usually manually performed in the classical RBT and are managed by CATEGA that is implemented by a former project (see Section 1.2):

- **Define Test Completion Criteria:** Under the assumption that 100 % testing can never be achieved for highly complex systems, the test scope and its efforts must have specific, quantitative and qualitative goals, so that testing can be considered as completed when the goals have been achieved, e.g. testing is complete (Test Coverage Criteria in this thesis) when:
    - all functional variations have been verified,
    - the verification is fully sensitised for the detection of defects (including robustness tests),
    - all requirements have been fulfilled and verified,
    - a set of runs have been executed successfully with no implementation changes in between.

- **Design Test Cases:** The specification of the test is usually characterized by a five step-approach (Test Specification in this thesis):
    - definition of the initial state,
    - identification of the required data,
    - test case inputs,
    - test case expected outputs (pass/fail criteria) and
    - the final system state.

- **Build Test Cases:** The implementation of the test for the specific test environment (Test Generation).

- **Execute Tests:** Execute the test case steps against the system under test and the documentation of the results (Test Execution).
- **Verify Test Results:** Evaluation of the test results to verify that the test results are as expected (part of the test evolution in this thesis).
- **Verify Test Coverage:** Evaluation of the test results to track the amount of functional coverage achieved by the successful execution of the set of tests (part of the test evolution in this thesis).
- **Problem Reporting Management:** Any defects or problems detected during the testing phase are managed including the interface to the supplier to manage the solving process as well (only defect detection has been considered in this thesis).
- **Manage the Test Library:** All implemented test cases are to be managed to maintain the relationships between them and the configuration of the system being tested (Test Configuration in this thesis).

The characteristics of a test process usually used - which have been taken into account in this thesis to ensure the applicability of this process for the MAIP implementation and to ensure the generic feature of this thesis - are as follows:

- **Testing must be timely:** Testing activities are a sub-set of the system development activities and represent the basement of the system and software development processes to ensure that the development process is well implemented and its activities are well performed. Usually the time needed for testing is underestimated and the relevance and the priority are not sufficiently highly graded because of the limited resources, which could mostly lead to a reduction of the test scope. A reduction of the test scope mostly impacts the achievements of the development objectives and impacts the maturity of the system being tested. Optimizing the test process by using smart methods such as software agents is a conceivable compromise to compensate this situation.
- **Testing must be effective:** The test design should not rely on individual skills and experiences. Instead, it should be based on a repeatable test process that is deterministic and should encapsulate the complexity of environmental needs, so that the test cases derived from this test design can be performed also by persons who are not experts. The encapsulation of the complexity of the test engine has been realized by the implementation of the CCM in our thesis (see Section 3.9).

- **Testing must be efficient:** To ensure that test cases can also be executed quickly when the test configuration has been changed, a high degree of automation is required. Usually, because of the close dependency of test cases and the respective test configuration, it is very difficult to realise the automation of the code generation without immense efforts. In our thesis a solution to this situation has been implemented by using adaptive software agents which are able to adapt test cases automatically with respect to the test configuration changes.
- **Testing must be manageable:** To manage test activities, the test process must provide sufficient metrics to quantitatively identify the status of testing at any time. Test management and documentation are not considered in this thesis, but we have relied on the former project implementation of CATEGA (see Section 1.2) that has been implemented for these objectives.

The RBT methodology is a multi-step process. The main steps of this process are described below:

- **Specify and validate system requirements:** Derived from the objectives for which the project is being initiated, the system requirements are to be specified and validated. For the validation of the requirements different methods can be used:

  - Review (mostly applied)
  - Analysis
  - Prototyping and simulation

Software-agents can be used to support the validation activities especially for the formal validation if the requirements are formally managed by requirements management systems. Two main criteria during the validation phase are to be analysed:

Correctness

The correctness of a requirement statement means that the requirement is verifiable and there is no ambiguity or error in the requirement statement and its attributes. A set of requirements can be said to be correct if there are no inconsistencies or contradictions between requirements in the set. The correctness check can be supported by software agents but it is not the focus in this thesis.

Completeness

Completeness of a requirement statement or a set of requirements means that all attributes have been considered and that those stated are essential and sufficient to allow the design to be initiated and carried out. A complete set of requirements defines the behaviour of the system or system components in all operating conditions. If and only if the requirements are defined for all operating conditions and are linked to all operating contexts and their requirements, the requirements can be assessed as traceable. A completely defined requirement must ensure traceability.

- **Design system:** With respect to the validated system requirements the next engineering steps is to start to design the system, followed by the design verification. The aim of design verification is to provide evidence that the design is compliant with the requirements. This compliance evidence is usually presented during design reviews and it is a key input to decisions on the choice of design solutions and whether to proceed to the next stage of development. Design verification can be supported by software agents as well, especially if the design is realised by using formal methods or modelling techniques. In the ideal world the test model represented by CAM can be derived automatically from a system model when the test requirements and pass/fail criteria are defined in the system model. This aspect is not a part of this thesis but is an important issue for further research.

- **Product Verification and Integration:** The goal of product verification is to ensure that the product fulfils the requirements and the respective design. The product may be the system component or system and shared simulation models, and verification/integration is needed at each level. This will be achieved by defining and carrying out activities in order to provide evidence that the requirements are met. The main scope of this thesis is optimisation of this verification process by using software agents embedded in a multi-system integration platform (MAIP).

- **Product Validation:** Product validation activities aim to demonstrate that the product meets the needs of the customer (development objectives), i.e. the product can be used to perform the required operational functions over a wide range of operational or in-service scenarios with a higher level of maturity. The product validation activities shall be based on the definition of operational scenarios. At system level, product validation activities (main scope of this thesis) have been used to show that the system handles the widest possible range of operational and software agents and have been used to increase the test coverage by executing test cases in different operational modes and within different test configurations.

  <u>Characteristics of Testable Requirements</u>

The testability of requirements is an essential prerequisite for the use of software agents. When test requirements and their associated pass/fail criteria are not clearly defined, the CTASM- entities and their expression cannot be built and consequently software agents cannot be used. To ensure the testability of requirements, the requirements ideally should have all of the following characteristics (based on the definition of [Bender 2009]):

- **Deterministic:** Given an initial system state and a set of inputs (particular signal value set, see Section 4.2), it must be able to predict (based on pass/fail criteria) exactly what the outputs will be.
- **Unambiguous:** All involved entities and project members must receive the same clear meaning from the requirements, leaving no room for interpretation.
- **Correct:** The relationships between causes and effects are described clearly and correctly. This relationship shall also be retained during realisation, so that the verification activities can rely on it.
- **Complete:** All requirements are considered and included. The checking of this completeness shall be provided by the requirements management system. There was no need identified to support this activity with software agents within this thesis.
- **Non-redundant:** Requirements should provide a non-redundant set of functions and events. Usually requirements management systems provide a functionality to support avoidance of redundant requirements, but this task is mostly done by reviews. Also, there was no need to support this activity with software agents.
- **Change control:** Requirements, like all other developing entities of a project, should be placed under change control.
- **Traceable:** Requirements must be traceable to each other, to the needs, to the design, to the test cases and to the implemented code, including linking to the origin of these requirements (parent requirements). Derived requirements, especially implementation requirements, are also to be considered.
- **Feasible:** If the system design is not capable of delivering correct and unique requirements, then it is not feasible to implement and test the requirements.

The key characteristics in the above list are deterministic and unambiguous. Testing, by nature, is comparing the expected results to the observed results. Unfortunately, one usually never finds requirements specifications written in sufficient detail to be able to implement and test them without additional efforts with respect to formalisation and clarification. In this thesis the definition of the CTASM (see Section 4.2), ideally supported by the system designer, can be considered as a requirement formalising task. This formalising task results in a formal model which thereafter is available to be used by software agents to perform their testing tasks.

System development life cycle with integrated testing

In most system and software development life cycles, the bulk of testing occurs only when implemented product becomes available. With the RBT process, testing is integrated throughout the life cycle for all of the reasons stated above as shown in Figure 34.

Note: Actually, in our life cycle we use a water fall approach; to describe a validation or verification task of the V-Process described in Section 2.1 and to simplify the V-process.

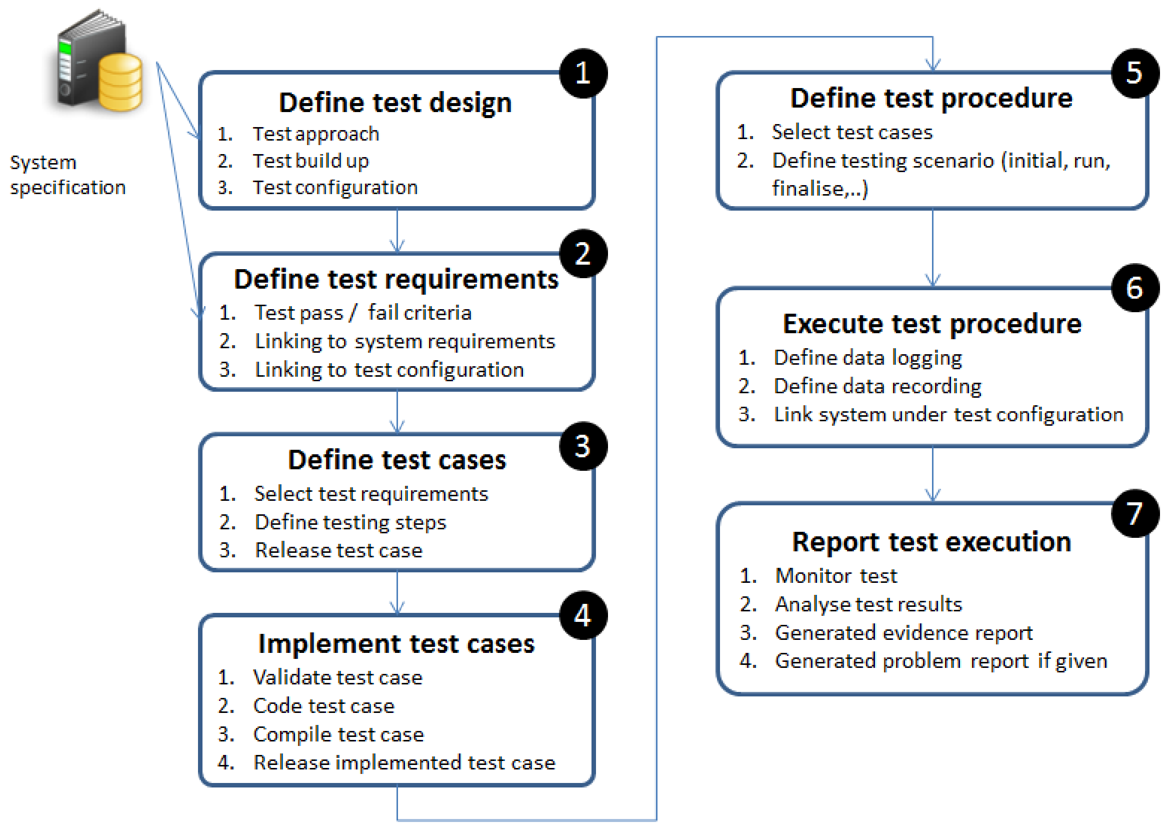


Figure 34 Life cycle of a testing task

(1) Define test design:

Test processes determine whether the development products of a given activity conform to the requirements of that activity and whether the system and/or software satisfy its intended use and user needs. The scope of testing encompasses software-based systems, computer software, hardware, and their interfaces.

The test design is the first stage in developing the tests for system and software testing projects. It records what needs to be tested, and is derived from the documents that come into the testing stage, such as requirements and designs. It records which features of a test item are to be tested, and how a successful test of these features would be recognized. Test design usually includes the definition of the test approach, the test build up definition and the respective test configuration, in which this test design is valid and applicable. The research question with respect to the test case applicability is to be addressed here. Only test cases for which the test is designed can be performed. The agent checks if all signals linked to states and transitions are instrumented and configured within this test design, otherwise the test case is not applicable. To be able to perform this check a test design must be formally defined (it is the case here).

(2) Define test requirements

The process to convert system requirements to testable requirements by formalising them includes the definition of the pass/fail criteria, the linking of system requirements to be tested and the linking of the valid test configuration. Test cases are linked to test requirements in order to identify if the corresponding system requirements are tested.

(3) Define test case

The Test Case Specification is developed in the Development Phase by the organisation responsible for the formal testing of the application. A Test Case Specification describes the purpose of a specific test, identifies the required inputs and expected results, provides step-by-step procedures for executing the test, and outlines the pass/fail criteria for determining acceptance. Test cases can be determined based on the test cases that are linked to the test requirements and are chosen in the request for test.

(4) Implement test cases

The implementation of test cases is the process task which is handling the coding of test cases with respect to the target test engine, to which the system under test is physically connected. This process task consists of four sub tasks. Before a test case can be implemented, it shall be validated regarding correctness und implementability. The second task is the coding of the test cases which can be automated by supporting them with code generation tools. The third step handles the compilation with respect to the target platform which can be supported by compilation tools as well. The last task is the releasing of the test cases to allow using them for any test execution.

(5) Define test procedure

Details with respect to how the tester will physically run the test, the physical set-up required, and the procedure steps that need to be followed including the linking of test cases involved and the specific environmental needs if given.

(6) Define test execution

For the defined test procedure the test execution definition task can be started, for which the data elements for logging and recording shall be selected and a valid test configuration logging is to be linked. The test configuration is a set of configuration items which are needed to be defined for a test execution (e.g. test engine version, complier version used, initial configuration file required …).

(7) Report test execution

For the monitoring of tests specific monitor and injection panels (GUI controls) are to be generated. The test logging describes in detail which tests cases were run, who ran the tests, in what order they were run, and whether or not individual tests were passed or failed. The test problem report is a detail of the actual versus expected results of a test, when a test has failed, and any indication why the test failed. The test report including evidence report describes in detail all the important information coming out of the testing procedure, including an assessment of how well the testing was performed, an assessment of the quality of the system, any incidents that occurred, and a record of which testing was done and how long it took to be used in future test planning. This final document is used to determine if the software being tested is viable enough to proceed to the next stage of development.

Test System

Generally a test system (test bench, TB) is a virtual environment used to verify the correctness or soundness of a design, a real component or a simulation model (e.g. embedded systems). For this thesis the test system has been represented in a generic way with respect to the test engines used in the validation and verification of aircraft cabins. Test systems are generic and consist of the following test system components:

Physical hardware:

- Computing unit (CPU-card). See Figure 35 (1)
- Input / Output cards (I/O) to simulate the component I/O if needed. See Figure 35 (5)
- Work station (configuration host). See Figure 35 (6)
- Simulation host. See Figure 35 (2)

Real time core (RT-core or test engine):

- Distributed real time software platform. See Figure 35 (1)
- Hardware abstraction layer (HAL, Driver, and Services). See Figure 35 (1)
- Signal server which is responsible for the management and accessing of the component parameters. See Figure 35 (3)
- I/O Configuration server which is responsible for the mapping of physical interfaces to the respective parameters. See Figure 35 (4)

Real time tool chain (RT-environment or RT host):

- User interface (Configuration GUI). See Figure 35 (6)
- Integrated development environment (IDE). See Figure 35 (6)
- Debugger. See Figure 35 (6)

System plug (unit under test adapter or UUT plug):

- Discrete / analogue adapter (regarding UUT). See Figure 35 (7)
- Bus adapter (regarding UUT e.g. CAN Arinc429...). See Figure 35 (7)

Plugged Components:

- Component or unit under test “UUT”. See Figure 35 (8)

Components are connected on the test system by using specific system plugs with respect to the required interfaces supported by the components to be validated or verified.

Figure 35 illustrates a schematic overview of the components needed in the test system as a generic definition of the well-established test benches.

<u>General test build up and configuration</u>

The following steps are generally required for the preparation of the test engine before any test can be performed:

- Physical connection of the system under test:
  The system is to be physically connected to the test bench by using a specific adapter that connects the system under test connectors to the test system I/O cards. These connections allow the stimulation and perception of the system under test connectors. This task allows agents to determine if the physical connector is instrumented. When a physical connector is required but not instrumented for a specific interaction with the system under test, the corresponding test case is not applicable then.
- Mapping of physical I/O to logical I/O Model:
  One of the key features of the test engine is its ability to store and transport data. At first glance, this seems to be a trivial task. Nevertheless, it has some deep impact on how test engines implement this task. All data must be represented in a known and agreed upon manner. Since there are different kinds of data, there are also different ways (some of which are well known) of storing data. But even one the same kind of data can be stored and organized in a multitude of different ways. And several of these possibilities are actually used in this thesis. The same holds for the way in which data can be transported. Sometimes the methods of transporting data are directly derived or even dictated by the way the data is physically stored and represented. Sometimes the data undergoes considerable conversion or encoding before it is actually transported. The essence is always the same: we have more than one possibility to implement the transport of data.
  Test engines take a simple approach to solve this problem. All data in them are stored in a unique manner, which is independent on how the data was produced, transported or acquired. For the moment, we call this the internal format. For each external format, a test engine implements two methods: one “in” method to convert from external into internal, and one “out” method to convert from internal to external format. This work is accomplished and implemented in a piece of software called test engine I/O driver.
  The test engine used for the evaluation provides a standardized interface for such drivers. The important part of this interface is two procedures: the driver’s frame-in and frame-out procedures. The first procedure implements the transport of data from the external format into an internal signal table (real time signal “RTS”) the second implements the transport in the opposite direction, i.e. from the RTS to the external format.
  Other parts of this interface include the definition of how the driver interoperates with the test engine and how the behaviour of the driver is configured. This driver-specific, configurable behaviour also contains a standardised description of how to transport data from the internal format to the external format, and vice versa. Another part of this description defines how to convert data from and to engineering units (in this thesis this description is called an I/O map). This task allows agents to determine if a signal is defined in order to ensure the applicability of a test case.
- Integration of system simulations:
  The typical test application on a test engine consists of a number of simulations simulating the dynamic behaviour of components needed to stimulate the system under test, a number of panels visualising the state of the system under test and of simulated components and provides interactive elements to control the test scenario. The integration of these simulations and panels is the process that consists of the following tasks:

  - Definition of the models’ input and output variables and creation of the RTSs containing the corresponding channels as required for the interaction with these signals.
  - Compilation of the models for the corresponding test engine.
  - Validation of the generated code.

- Integration of the simulation in the test procedures.
- Generation of the test procedures:

The test procedure and the incorporated test cases shall be implemented in an appropriate way to be able to run it on the test engine (usually c-code). The test procedure has to consider the I/O mapping and the integrated simulations. Simulations are a mutable, collaborative and competitive part of the environment an agent has to handle. To be able to interact with simulation, simulation publishes its signals submitted by agents so that the behaviour of the simulation can be detected by agents each time it is required.

- Generation of the test configuration:
  To be ready for a test execution a test configuration shall be defined, in which the test procedure is valid and applicable. For the test configuration the following items are to be recorded:

  - The version of the system to be tested (the so-called system standard in the industry)
  - The version of the I/O mapping file
  - The version of the realtime signal definition
  - The versions of the integrated simulations
  - The version of the test procedure
  - The version of the test engine tool chain
  - The version of external environmental needs if given.

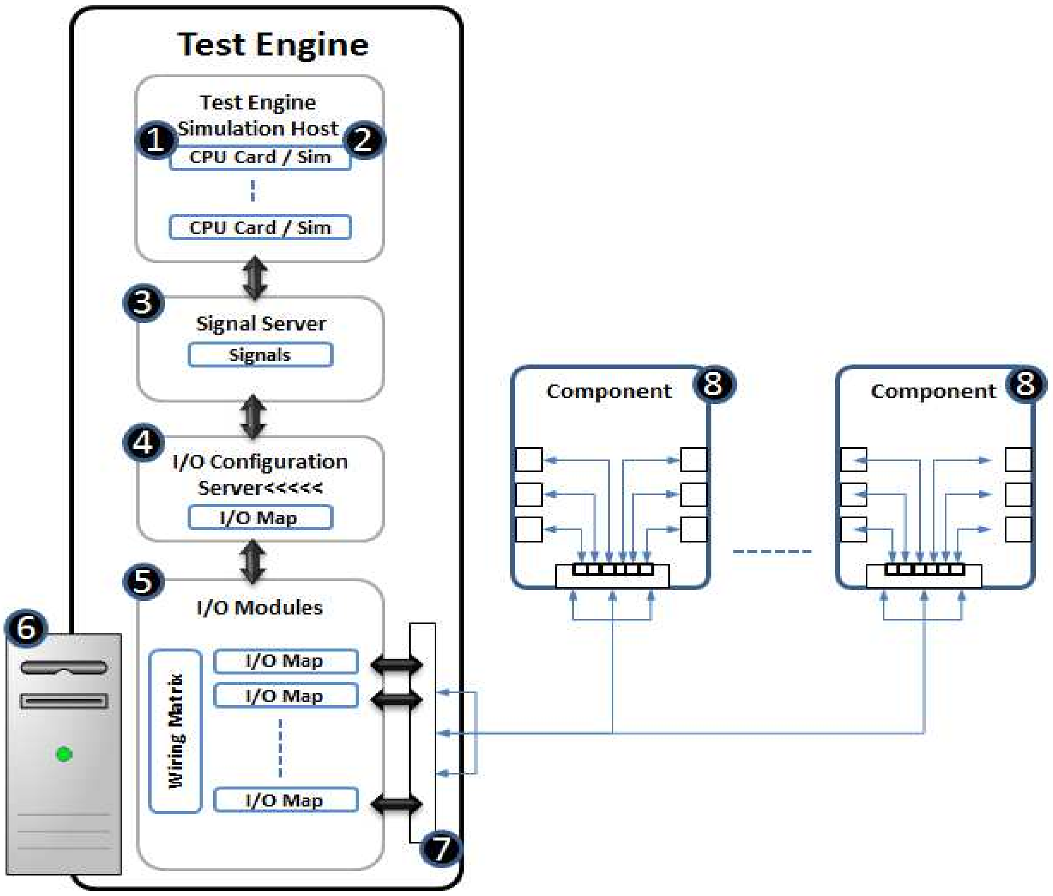


Figure 35 Test system

5.2. Implementation of agent-based testing

For the implementation and evaluation of agent-based system validation and verification, a generic implementation design has been defined with respect to the main testing task, as shown in Figure 36:

- Test specification.
- Test run.
- Test monitoring and evaluation.

The generic implementation has been specified and used for three different application domains in order to validate our approach therein. These three domains of system validation and verification have been selected by the ABT-project that is accompanied by this thesis. The three main domains (simulation validation, controller verification and data security testing) are described and discussed in the next sections.

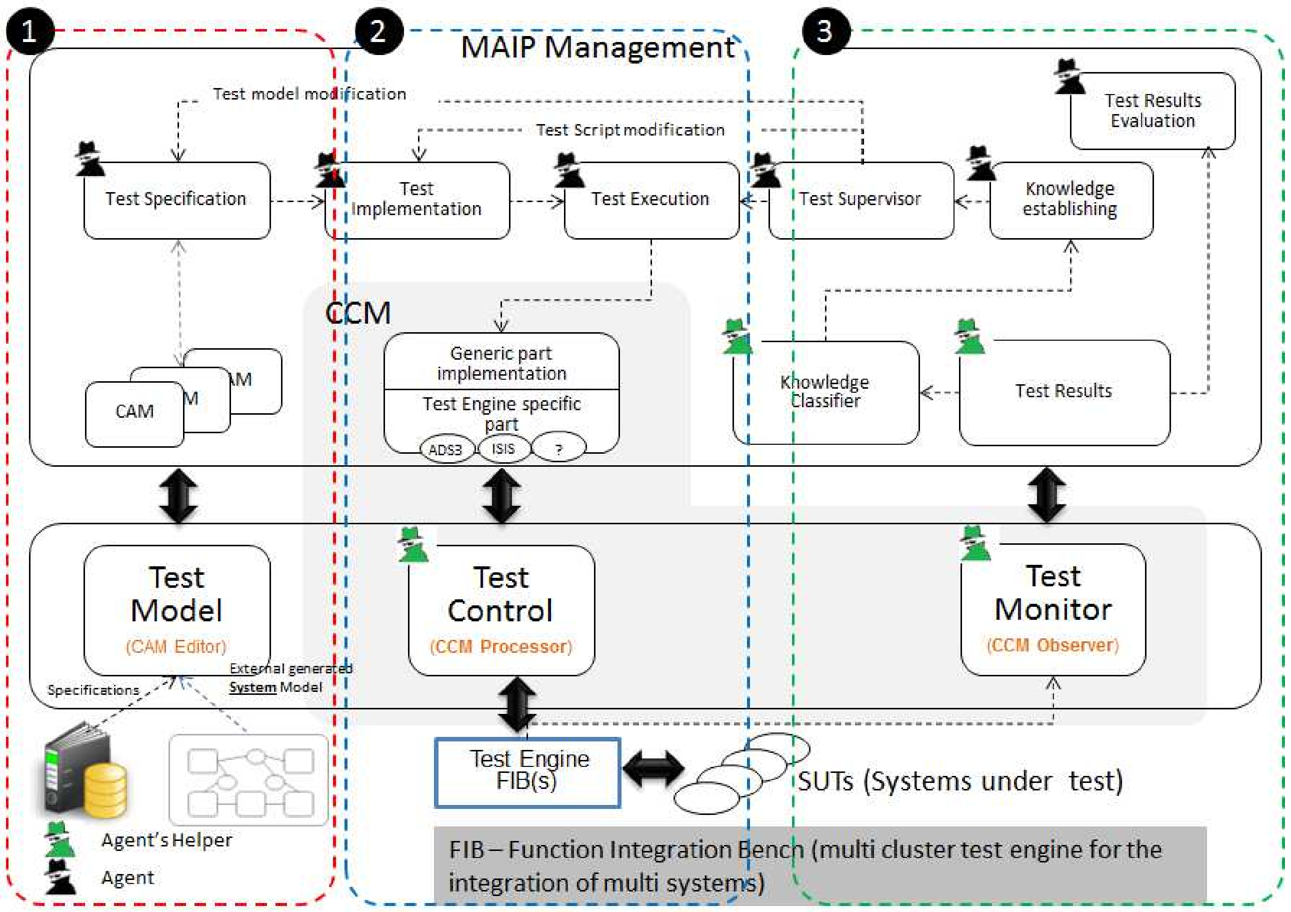


Figure 36 Agent based testing implementation

Test specification (main task 1)

The first main ABT testing task is the development of the test specification that is divided into three sub tasks supported by a tool developed within the accompanying ABT-project:

- Test Model: Based on the system model developed by system designers or on the system specification, a test model has been developed by using the CAM Editor (graphical user interface developed by the ABT-project) in order to define an appropriate CTASM (see Section 4.2). The modelling of a CTASM is an engineering task that determines the following items.

- Which features (operation, state, transition and constraints) of the system under test are to be tested?
- Which interfaces are required?
- Which signals are to be monitored, stimulated and recorded?

In the case of system modelling an agent supported by agent helpers generates the test model automatically. In the case of using the system specification, the tester generates the test model by using the CAM editor manually (specific editor to define a CAM).

- CAM: The CAM is the representation of the test model persistent in XML-format and can be used by agents any time when a test code is to be generated. By using different types of agents, different test procedures can be created depending on the type and depth of the test required to be developed. Within this evaluation task the Cam creation has been demonstrated by the tester teams as easy to perform (2-4 hours)

- Test Specification: Based on a given CAM, different test case generation scenarios can be applied to specify a certain test. For the definition of test case generation scenarios different agent implementations and algorithms can be used, based on the generic approaches defined in Section 4.3. For the evaluation of this thesis three approaches have been implemented represented by the following test case generation modes, wherein any other approach is also allowed and possible:

  - Relaxed mode: The test case scenario is defined by achieving each state at least once caused by any arbitrary transition. This mode has been chosen for this evaluation task. The reason for this selection is maintaining the comparability to other testing approaches that are not able to perform more complex modes.
  - Strict mode: The test case scenario is defined by achieving each state caused by all defined transitions leading to it and over all particular signal value ranges. The iteration over a range can be configured by the user manually.
  - Configured Mode: The test scenario is specified by the user manually.

<u>Test run (main task 2)</u>

- Test Implementation: The specified test is cross-compiled with respect to the target test engine (a specific test engine called ADS II of the company TechSat has been selected as target test engine).
- Test Execution: The test procedure is initialised, simulation is loaded, required monitoring panels are displayed and CCM instances are created.
- CCM Delegator: The CCM Delegator instance is created which is responsible for establishing a connection to the test entities running on the test engine (see Section 3.9.4).
- CCM-Test Control: Is created and is responsible for the realtime processing of signals of the test engine which are connected to the realtime signals already mapped to the I/O of the system to be tested (see Section 3.9.4).
- Three instances (only two are part of this task) of the CCM are created which are responsible for the test execution test control (realtime capability is ensured).

<u>Test monitoring and evaluation (main task 3)</u>

- CCM-Test Monitor: is created and is responsible for the generation of test results based on the pass / fail criteria defined in the test procedure (see Section 3.9.4).
- Knowledge classifier: is created and responsible for the classification of new knowledge if given, based on the test procedure aspects of knowledge discovery (see Section 4.3).
- Knowledge establishing: If new knowledge is approved by a specific level of occurrence, the next instances are to be informed by this process task.

- Test supervisor: The instance which is hosting all agents and is responsible for the configuration and test case generation required for the validation or verification of the system under test. This task is also responsible for the modification of the CAM if required, and for the code injection which is used to define the next testing steps when deviating from the test procedure. This deviation may be required when new knowledge requires test procedure changes.
- Test results evaluation: Based on the pass / fail criteria defined in the test specification and on the information exchanged with the CCM-Monitor instance, the test is evaluated on runtime to reduce the evaluation time after testing which is required for other test approaches.

5.3. Evaluation Criteria

5.3.1. Test Coverage

The testing process for safety-critical systems as for aircraft systems is usually evaluated with code coverage criteria such as MC/DC (Modified Condition/Decision Coverage) defined in the standard DO-178B, Software Considerations in Airborne Systems and Equipment Certification (a de-facto standard for certifying software in the civil aviation domain). For requirements-based and script-based testing techniques we worked on coverage metrics that are defined on the requirements and that are independent of a specific implementation. The same techniques have been used in this thesis for the model-based and agent-based approach, duo to the fact that the requirements traceability is defined in standard DO-178B.

For that purpose we analysed the relationship between existing definitions for structural requirement-coverage metrics, functional-coverage and structural code-coverage metrics.

Test cases are generated from the requirements until all requirements are covered. This task is to be performed by system designers who are responsible for the requirements definition and the definition of the validation and verification activities. The requirements to be tested and the test depth required shall be defined in the request for test documents that is to be committed by the tester with respect to test resources, test capabilities and test build up.

The tester has to define his test by describing the test, approach, methodologies, test build up and how he manages the test and the test configuration. This test definition is accepted by the system designer, who evaluates this test approach and ensures that the test coverage can be achieved therewith. If the coverage is assessed as insufficient by the system designer, additional test cases are generated. The aim is to achieve full requirement coverage, to verify that all requirements are correct and fully cover the intended system behaviour.

In order to formalise system requirements the so-called test requirements are to be implemented, which enable the automated test case implementation. The requirements are also used to derive the test requirements, which can then be used in the test suite for testing purposes.

We considered formal requirements to be another implementation of the system specification. For the model-based and agent-based approach, test models derived from the system requirements are to be designed and generated so that the logical conjunction of all the formal requirements should summarise the software behaviour which is represented in the test model. Treating the formal requirements as a model implementation, we worked on defining requirement-coverage metrics as structural coverage criteria. These structural coverage metrics are used to guide the test-data generation to derive a test suite.

Structural test coverage is a class of coverage metrics used to reason about the sufficiency of a given test suite. A variety of metrics exists; including statement coverage, decision coverage etc. In the safety-critical domain, one established structural coverage metric is the modified condition/decision coverage (MC/DC). The basic idea of MC/DC is to test whether each condition of a decision can independently control the outcome of the decision (without changing the outcome of the other decisions' conditions).

The terms condition and decision refer to the structure of the testing code. At higher abstraction levels, notions like model coverage are used. At model level, structural coverage is used in a rather ad hoc fashion, without having established definitions as they are common for test code level.

Within the evaluation process, we worked on the mapping of structural test coverage between the different representation layers (system components, systems and functions). The implementation is done in a domain-specific modelling environment (MAIP), from which a code generator produces testing code that is then transformed into a test engine running code (usually in C). The evaluation approach was to define the properties of code transformations such that the chosen structural code-coverage metrics are preserved by the code transformation (into test engine code). With this approach one can use the testing source code or the model to generate test cases automatically and independently of the hardware platform.

We assumed in our evaluation that a 100 % test coverage is given, if all requirements defined for tests and assigned as to be tested are covered in test cases and accepted by the system designer, regardless of which testing approaches are to be used.

5.3.2. Test preparation effort

System test preparation effort estimation has always been an on-going challenge to system and software engineers, as test preparation is one of the critical activities of system development.

For the test preparation effort estimation we used a 5-step process consisting of the following stages:

- Identify requirements to be tested
- Classify requirements into the complexity classes with respect to their complexity (e.g. number of parameters, test case steps and number of test cases) ; simple, low complexity and high complexity
- Define for each complexity class the realistic required preparation time
- Calculate the preparation time required to perform all tests defined in the test request
- Define 100 % test preparation time; the shortest preparation time of all testing approaches is representing the 100 % reference for the results assessment.

5.3.3. Test Generation

Based on the formally specified test requirements and their respective test cases an analysis of the test generation task has been performed regarding the following aspects:

- Is the test specification sufficient to perform test generation automatically? (50 points can be achieved).
- Is the generated code runnable without any modification? (20 points can be achieved)
- Is the generated code test configuration-independent? (30 points can be achieved).

The number of points correlates to the average effort suggested by the ABT-project. These suggestions are based on measurements made during earlier test campaigns. The sum of the points that can be achieved is used for the evaluation of test approaches.

5.3.4. Test Execution Monitoring

Monitoring test execution is difficult to do and harder to do well. In this thesis the following elements which can be used for test execution monitoring have been identified to measure the quality of test execution monitoring capabilities:

- Low-level debugging including bug advocacy (40 points can be achieved)
- Manipulation panels including error injection generated by the test generation process (30 points can be achieved)
- Logging and recording of test data (30 points can be achieved).

The sum of the points that can be achieved is used for the evaluation of test approaches.

5.3.5. Test Evaluation

Several criteria must be defined before measures with respect to the quality of the test evaluation can be proposed. If any of the criteria are not fulfilled, the quality of the test evaluation will be impaired. For the evaluation of the test approach the following criteria have been defined:

- Performance and stress testing tool: is integrated with the GUI testing tool (10 points can be achieved)
- Supports stress, load, and performance testing: Allows for simulation of users without requiring use of physical work stations (10 points can be achieved)
- Ability to support configuration testing: Tests can be run on different hardware and software configurations (10 points can be achieved)
- Support resource monitoring: Memory, disk space, system resources (10 points can be achieved)
- Synchronization ability: A script can access a record in a database at the same time to determine locking, deadlock conditions and concurrency control problems (10 points can be achieved)
- Ability to detect when events have been completed in a reliable fashion (10 points can be achieved)
- Ability to provide client to server response times (10 points can be achieved)
- Ability to provide graphical results (10 points can be achieved)
- Ability to provide performance measurements of data loading (20 points can be achieved).

The sum of the points that can be achieved is used for the evaluation of test approaches. The tests have been performed by the ABT-project team supported by aircraft cabin testing teams.

5.4. Case study: verification of system simulation

Modelling and simulation play increasingly important roles in modern system testing. They contribute to our understanding of how entities function and are essential to the effective and efficient design, evaluation and operation of new systems. Verification and validation (V&V) within this field are processes that help to ensure that models and simulations are correct and reliable. Although significant advances in V&V of complex systems have occurred in the past decades, the validation and verification of models and simulations is still highly complex and limit the use of the full potential of modelling and simulation, where agent test approach has shown its strength.

For our case studies a cabin pressure control system (CPCS) simulation has been used. The cabin pressure control system continuously monitors the airplane's ground and flight modes, altitude, climb, cruise, or descent modes as well as the holding patterns at various altitudes. The system uses this information to regulate pressure by controlling the outflow of cabin air by opening or closing the pressure regulation valves located on the exterior of the fuselage. The CPCS system is an application hosted by a specific controller. In this use case the CPCS application has been simulated on a generic platform to perform verification activities for the clarification of it. The verification of the simulated CPCS supports time reduction due to the fact that it can be performed before the development of the hosting controller is completed in order to reduce the testing time.

In this evaluation case study, verification of system simulation has been considered for the evaluation of our test approaches with respect to the evaluation criteria defined in Section 5.3 with the following assumptions that correlate to the planned testing time resources allocated in the testing master plan and suggested by the ABT-project:

- 4 weeks preparation
- 8 test objectives
- 56 test cases.

5.4.1. General Test Approach

For the verification of the simulation generally the following five tasks were required regardless of which test approach is to be used:

1. Integration of the system under test (in this case the system simulation) in the test engine environment including the mapping of the system under test parameters or I/Os to the internal signal representation of the test engine.
2. Test specification including development of test design, test cases, test procedure and test configuration.
3. Implementation of test specification.
4. Execution of implemented tests.
5. Evaluation of test results.

Figure 37 shows the general test approach for the verification of the system simulation.

Requirements based testing (RBT)

For the classical RBT all test approach tasks are performed manually using the tool chain provided by the test engine supplier. Several teams are involved which are handling these different tasks. This approach is complex and requires different specialists with specific skills. The main problems of this approach are:

- The complexity of the implementation which is time consuming as well.
- The needs of specialists with different skills for each task.
- Susceptibility to changes or modification. All these tasks are to be performed again when system or implementation requirements are changed.

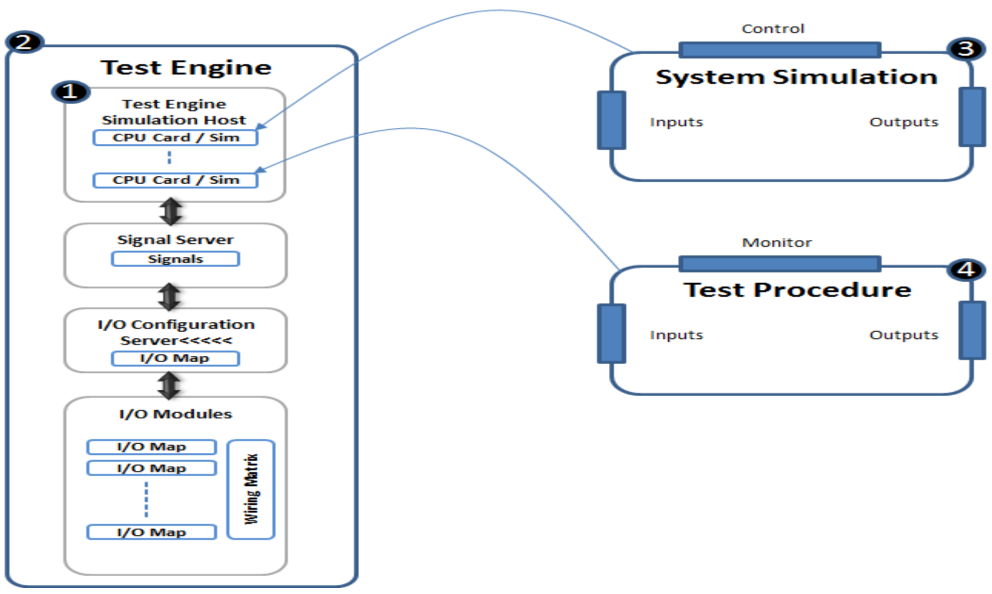


Figure 37 Test approach for the verification of system simulation

<u>Script-based testing (SBT)</u>

In this approach the five general test approach tasks are improved by using different scripts for the automation of these tasks.

For the integration of system simulation a kind of simulation import function has been implemented to simplify the integration task including the automated mapping of the simulation parameters to the internal test engine signals. The test specification is defined by using appropriate formal graphical editors that allow automation of the test procedure generation including the test configuration. Test execution and evaluation of test results remain unchanged.

<u>Criteria-based analysis</u>

By using SBT the test coverage has been increased and the time required for the test preparation has been reduced, as shown in Figure 38, because of time saved by automation which allows that more tests can be implemented in the a certain given time. Code generation and test execution have been improved because they can be performed fully automated. With respect to the automation of test implementation, a difficulty of test result analysing has been detected. The reason for this difficulty is the missing integration of failure correlation and smart failure location that cannot be implemented easily without modelling of the system under test behaviours.

<u>Model-based testing (MBT)</u>

The use of a model to describe the behaviour of a system is a proven and major advantage to test development teams. Models can be utilised in many ways throughout the product lifecycle, including: Improved quality of specifications, code generation, reliability analysis and test generation. In this thesis SysML based models have been used. In this evaluation task we focus on the testing benefits provided by model based testing (automatically derived from the system model) and present the solutions that overcame these challenges. In addition, the benefits of a model are demonstrated. The following test data are gained automatically by the model parsing process:

- Input / Output parameter of the system to be tested.
- State / Transition condition that describes the system behaviours.

- Time and performance constraints that are to be fulfilled by the system to be tested.
- Functional behaviours of modelled systems.

With respect to the general test approach task defined in Section 5.4.1 the following test activities are performed:

- For the integration of the system simulation the approach used for SBT was used for MBT as well.
- For the test specification an additional implementation for the automated test code generation was used based on the information gained by the model description exchange document, which represents the model specification (written in XML).The results of the model interpretation by using this test code generator is a formal test specification.
- Up to now the same process as used for the SBT approach can be used.

<u>Criteria based analysis</u>

By using MBT test coverage and preparation have been radically increased, as shown in Figure 38, since the test specification and the system simulation are based on the same model so that each entity, parameter and function has been tested. For the test generation and execution no benefits are identified, as the same approach as used for SBT was used. The test evaluation has been noticeably improved because of the direct link between system requirements represented in the model and the test specification developed based on it.

<u>ABT for simulation verification</u>

The agent approach implementation design described in Section 5.2 (see also Figure 36) has been used for the evaluation of the simulation verification.

A CTASM has been generated for the representation of the CPCS simulation which is very similar to the real application with the exception that the real I/O behaviour is not simulated in this use case because of the complexity of the realisation with respect to the representativity of I/O interfaces. The main scope of the simulation verification is focused on the functional behaviours of the application simulation.

Based on the generic case study implementation (see Figure 36) and with respect to the general test approach task defined in Section 5.4.1 the following test activities are performed to define the specific implementation for the simulation verification:

- For the simulation integration the same approach has been used which was used for MBT.
- Based on the simulation requirements a CTASM has been created for the CAM representation to develop the test specification and respective test configuration.
- For the implementation of test procedures different agents have been developed based on the CTASM to perform the following testing activities:

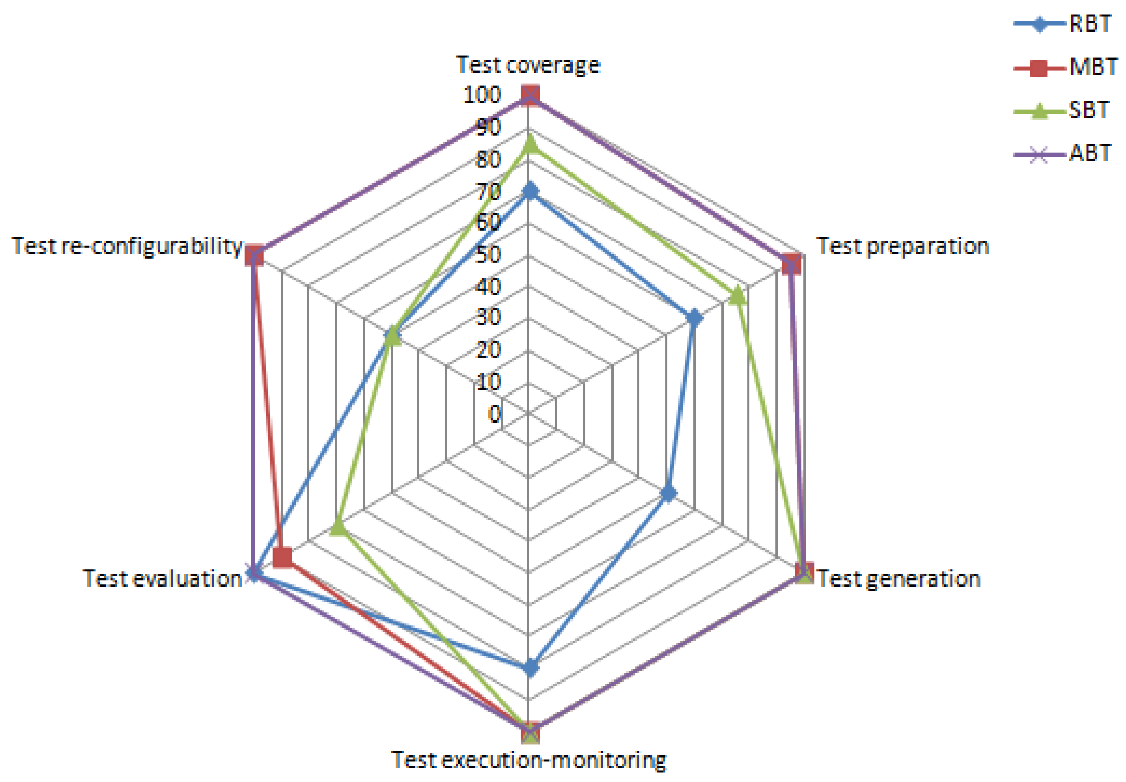
    - State and transition tests
    - Function behaviour tests
    - Performance and time constraints tests.

- For the test execution an agent has been developed to define and to configure the CCM instances required for the intercommunication with the test engine.
- For the test evaluation the CCM-Observer process has been configured to monitor and to report the test activities including failure detection, localisation and logging.

Criteria-based analysis

By using ABT the test coverage and preparation has been radically increased like MBT, as shown in Figure 38.

For the test generation and execution no benefits are identified because of the use of the same approach as used for SBT. The test evaluation has been radically improved because of the monitoring of testing steps directly by using the CCM-Observer instance (see Figure 19).



	RBT	MBT	SBT	ABT
Test coverage	70	100	85	100
Test preparation	60	95	75	95
Test generation	50	100	100	100
Test execution	80	100	100	100
Test evaluation	100	90	70	100
Test re-configurability	50	100	50	100

Figure 38 Evaluation of simulation verification

Discussion

The evaluation tables have been provided by the ABT-project teams (8 test teams). The basis of this evaluation were the test activities under the assumption mentioned in Section 5.4. The evaluation data have been measured and assessed during the whole test campaign and have been published after an internal review done within the scope of test approach evaluation of the ABT-project. In this thesis the radar chart (MS-Excel) has been chosen for the presentation of the evaluation results.

Test coverage

The model-based testing approach was based on the CPCS system model generated for the specification of the CPCS system. The generation of test objectives and, respectively, test cases has been performed automatically. For the agent-based approach the model has been transformed to represent the CTASM required for the agent-supported test performing. From the modelled CTASM an agent can derive test cases so that when the modelled CTASM represents the correct behaviours of the system under test the agents can generate test cases in order to verify the system behaviours. When a generated test case covers one or many system requirements linked in the request for test, then this test case can be considered as relevant. During the evaluation phase a CTASM has been modelled to cover all system behaviours for which the test request was written so that the test cases generated by agents were implicitly linked to the system requirements and are consequently relevant. Certainly agents can be configured to generate more test cases that are not linked to system requirements, but in this thesis this option was not accounted for the evaluation. When a CTASM represents the system behaviours correctly (each system requirement has been modelled), the generated test cases are relevant and cover all system requirements chosen in the request for test. In case of deviation between generated test cases and the request for test, the CTASM has to be improved until all system requirements are covered by the generated test cases.

The coverage factor of 100% (8 test objectives and 56 test cases) has been achieved for these two approaches. For the SBT the generation of these cases was limited by the complexity of several test cases which required strict synchronised intercommunication between test entities (concurrent computation). The RBT was the weakest approach due to the complexity of requirements formalisation to define the test requirements and their pass / fail criteria which lead to the limitation to achieve the goal of covering all test objectives derived from the test requests.

Test preparation

The condition was to perform the test preparation in a certain time of 4 weeks. The ABT and MBT have achieved this goal, but a few tasks (5%) required specific knowhow of modelling (experts) so that 95 % were achieved. The SBT has achieved this test preparation task where 25 % of task-specific knowhow of programming was required so that 95 % was achieved. The RBT approach has not achieved the required objectives with respect to the given time so that only 60 % of test cases were implemented (additional specific knowhow was required, tester and test engine knowhow was sufficient).

Test generation

Except for RBT, the whole test case generation has been performed automatically without any additional activities. The RBT has achieved 50 % only.

Test execution

The MBT, SBT and ABT test case execution tasks have been performed automatically without any additional activities. For the RBT 20% of the test execution demanded an additional modification to handle environmental needs not covered by the RBT test generation.

Test evaluation

The MBT was impacted by complex correlated error messages across different controller entities which were not sufficiently modelled before so that the ABT was the better solution, due to the fact that the evaluation tasks were decoupled from the CTASM and extended by learning. The SBT required additional programming tasks to perform test evaluation so that only 70% of the test cases led directly to evaluation results. Due to the fact that each requirement is to be verified and directly linked to a pass / fail criteria, the evaluation of the test result was directly assessed. If and only if the test case is implemented, the result evaluation was available.

Test re-configurability

The re-configurability of the test cases and their execution were performed automatically by ABT and MBT without requiring modification. For the RBT and SBT a new implementation or modification was required to handle a new system under test configuration or new environmental needs.

Recommendation

For the simulation verification task the MBT and ABT approaches are preferred, especially if a basic model of the system under test is given. For complex systems with compacted dependency the evaluation task by ABT can be performed efficiently by using decoupled agent tasks for results evaluation where more than one CTASM can be used together to evaluate the test results directly. This results evaluation can be done only because of the modelling of system functions and their corresponding signal patterns so that the potential failure sources are isolated. For that it is very essential to model the system function correctly so that agents can be provided with the correct data to be able to determine the failure source.

In this validation process two models were required to verify the CPCS system simulation by using them; the first one was the CPCS system model and the second one was the connected systems model (for example Doors Controller) required to represent the environment of the CPCS system model. The problem of using MBT was that a third model was required to model the interaction of the two models to be able to trace the failure source. For example, the door opening function can only be triggered when the CPCS system sends the signal “cabin pressure is adequate”. In the ABT approach the CTASM of the Doors Controller and the CTASM of the CPCS are implicitly connected due to the nature of agent communication capability, so that no third model was required. Due to this fact, many CTASMs can be connected to represent complex multi-system architecture without requiring additional models to realize the interaction between them.

5.5. Case study: verification of system controller

For this use case a doors control system (DCS) has been used. The doors control system continuously monitors the door’s states in ground and flight modes. The system uses this information to regulate the management of doors opening and closing with respect to the flight phase, cabin pressure, smoke and fire detection functions. The verification of the DCS is required for the achievement of the next integration level (doors integration into aircraft cabin) that can be started when the DCS is verified. A sysCTASM (multi CTASMs composed to represent the whole system behaviours) has been generated for the representation of the DCS which is very similar to the real application including I/O behaviour. The main scope of the DCS verification is focused on the functional behaviours and time constraints required to be integrable into the aircraft cabin environment.

In this evaluation case study verification of system controllers has been considered for the evaluation of our test approaches with respect to the evaluation criteria defined in Section 5.3 with the following assumptions:

- 4 weeks preparation
- 26 test objectives
- 84 test cases.

Requirements-based testing (RBT)

For the classical RBT of controller verification the same approach as used for the simulation verification was used with the following different test build up as shown in Figure 39:

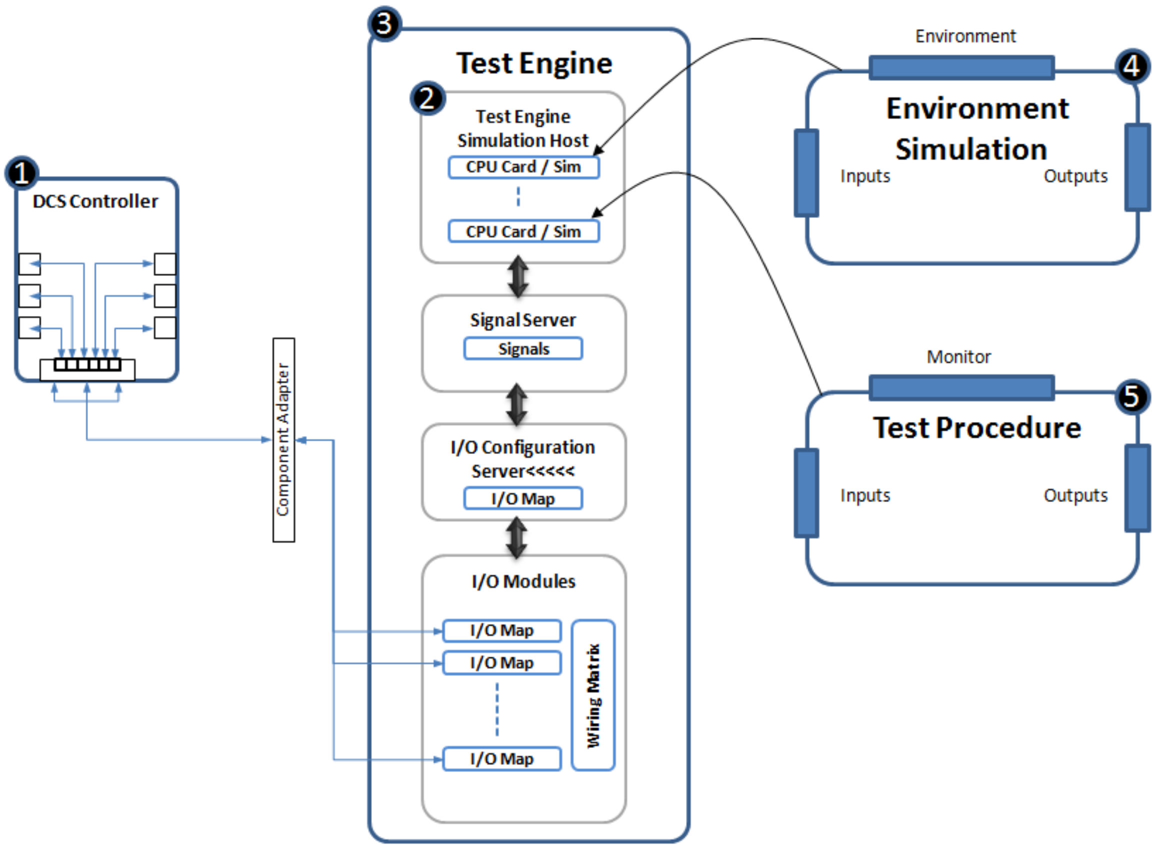


Figure 39 Evaluation of system controller verification

(1) Connected real DCS controller.
(2) Processing units (CPUs) of the test engine
(3) Test engine which is hosting all testing application.
(4) Environment simulation of the other systems required to move the DCS in an appropriate environment similar to real world installation within the aircraft.
(5) The test procedure implemented with respect to the test specification.

Script-based testing (SBT)

The same approach as used for the simulation verification has been used and no relevant aspects have been identified.

Model-based testing (MBT)

With respect to the general test approach task defined in Section 5.4.1 the following test activities are performed:

- For the integration of the system controller the approach as used for SBT is used for MBT as well.
- For the test specification an additional implementation for a test model based was required based on the system specification. The result of the model interpretation by using this test code generator is a formal test specification.
- Up to now the same process as used for the SBT approach can be used.

Criteria-based analysis

By using MBT the test coverage and preparation, unlike the simulation verification has been radically decreased, as shown in Figure 40, because of the missing reprehensive system controller model. In addition, the test specification and the test model were not based on the same model so that only a part of the entities can be tested within the limited time given in our assumption. For the test generation and execution no benefits are identified because of the use of the same approach as used for SBT. The test evaluation has been noticeably decreased because of the missing direct link between system requirements represented in the model and the test specification developed based on it.

ABT for simulation verification

Based on the approach used for the simulation verification, the following preparation tasks have been performed to define the specific implementation for the DCS verification:

- Based on the system requirements, a sysCTASM has been created which is a group of CTASMs.
- The test engine (ADS2 system) has been selected as a test execution host for this verification as well.
- The designed CCM instances have been configured.
- The same test objectives as executed for the RFs have been chosen.

The results of the use case evaluation are shown in Figure 40.

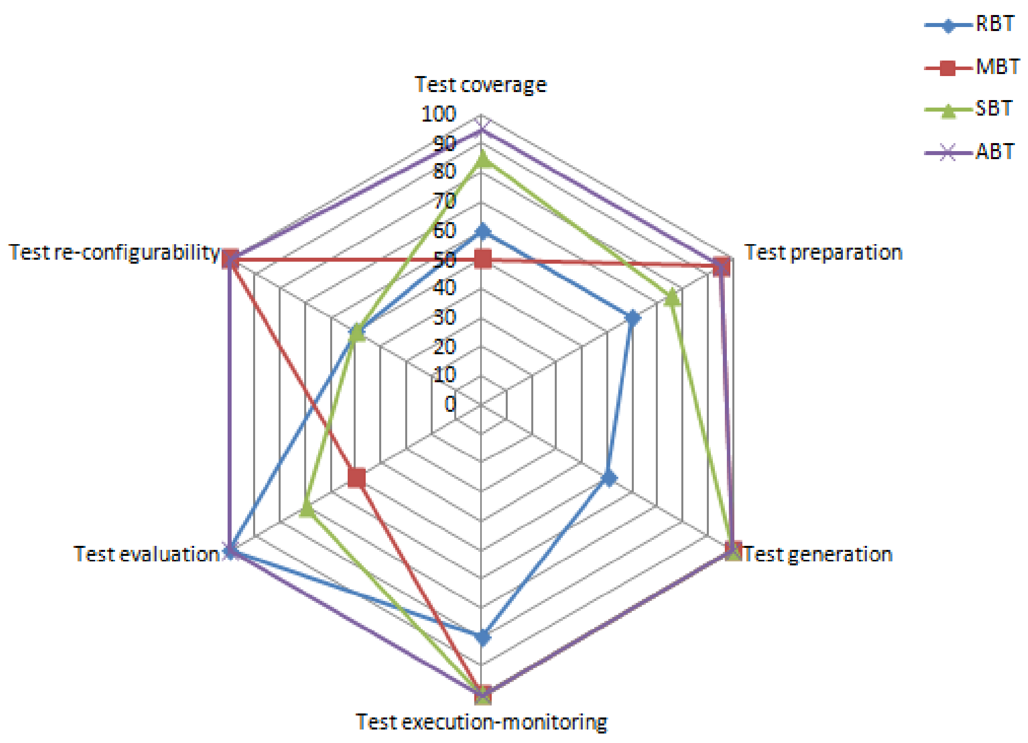


Figure 40 Evaluation of system verification

Criteria-based analysis

Test coverage

For the model-based testing a system model has been partially generated due to the limitation of a given time of 4 weeks so that only 50 % of this model has been generated. For the ABT the generation of the behaviours to be verified (only behaviours to be verified have been modelled; 95% representativity has been achieved, I/O behaviours were limited) with respect to the requests for test objectives (26 test objectives, 84 test cases) has been performed within 2 weeks. For the SBT the generation of these test cases which were manually implemented by the test team was limited by the complexity of several test cases which required a high implementation effort (60 % have been achieved). Like the simulation verification, the RBT was the weakest approach, due to the complexity of requirements formalisation to define the test requirements and their pass / fail criteria.

Test preparation

The requirement was to perform the test preparation in a specified time of 4 weeks. The same results have been achieved as shown in the simulation verification.

Test generation

The same results have been achieved as shown in the simulation verification.

Test execution monitoring

The same results have been achieved as shown in the simulation verification.

Test evaluation

Like the simulation verification, the MBT was impacted by complex correlated error messages across different controller entities because additional interaction models were required. Additionally, it was impacted by the limited behaviour modelled for this verification so that only 50% of the test cases could be evaluated. The ABT was the better solution here, due to the fact that the evaluation tasks were decoupled from the easy-to-generate sysCTASM and extended by learning (see state completion in Section 4.3). The SBT and the RBT had the same results as shown in the simulation verification.

Test re-configurability

The same results have been achieved as shown in the simulation verification.

Recommendation

For the system verification the ABT approach is preferred, especially if a basic model of the system under test is not available.

5.6. Case study: Data Security Testing

For this use case, a cabin intercommunication data system (CIDS) has been used. The CIDS hosts all cabin network services and can be considered as a multi-system function, due to the fact that the CIDS hosts 8 systems and about 130 services. The CIDS provides services and information to regulate the management of cabin intercommunication with respect to the flight phases and cabin operations. The verification of the CIDS data security requirements was part of the data security assessment of the cabin data security and has been considered as white box testing, due to the fact that all network protocols and their parameters and sequence activities were known. For these specific tests a funCTASM has been created for the representation of the CIDS network which is very similar to the real network including all protocols and virtualised network interfaces. The main scope of the CIDS verification is focused on the robustness behaviours and time constraints required to be released for the cabin intercommunication and its operations.

In this evaluation case study verification of data security has been considered for the evaluation of our test approaches with respect to the evaluation criteria defined in Section 5.3 with the following assumptions:

- 4 weeks preparation
- 80 test objectives
- 460 test cases
- CIDS hosts 8 systems and about 130 services.

Requirements-based testing (RBT)

For the classical RBT of data security test no test engine was required so that a new test build up has been defined, as shown in Figure 41.

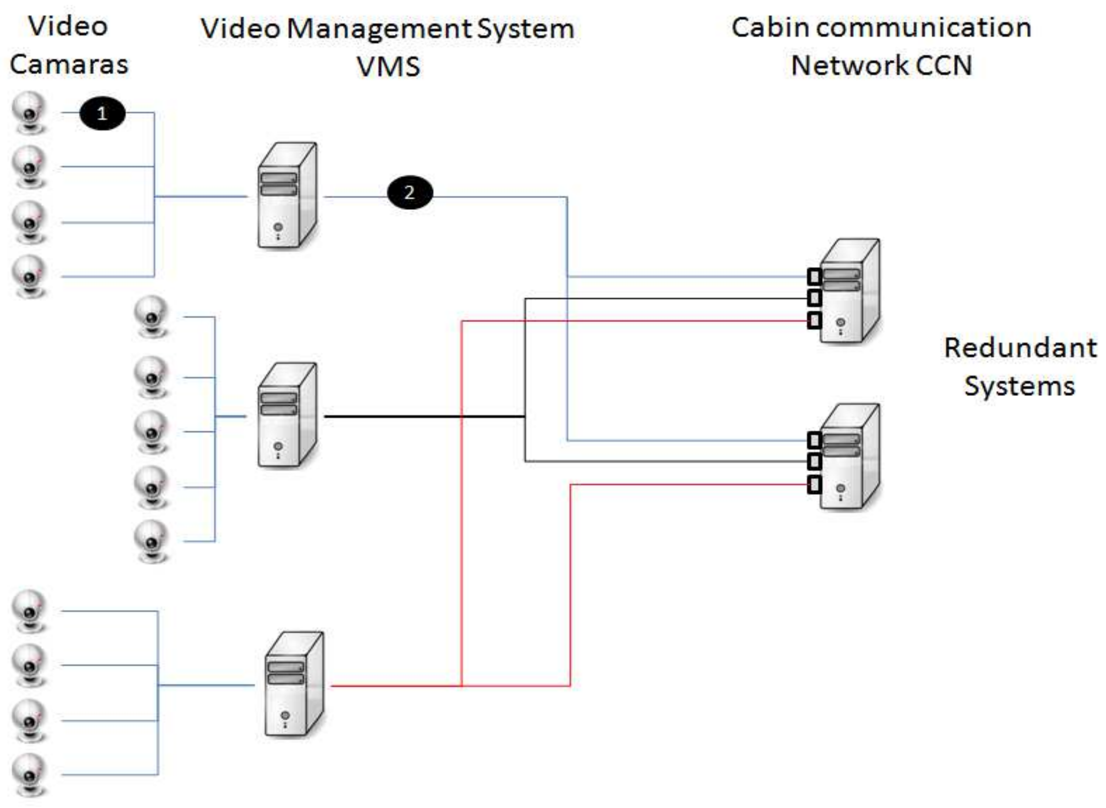


Figure 41 Data Security Testing

(1) The first network switch between the video cameras and the video management system which enabled the capability to monitor, manipulated the network streams sent by the video cameras.

(2) The second network switch between the video management system and cabin communication network which enabled the capability to monitor, manipulated the network streams sent by the video management system by playing the man in the middle.

<u>Script-based testing (SBT)</u>

Tools used for data security testing are often based on a scripting language such as Perl, Python, or Ruby. If a data security tester wants to extend, augment, or change the functionality of a tool to perform a test differently than the default configuration, the tester must know the basics of coding for the related scripting language.

Based on the generic test approach the following preparation tasks have been performed to define the specific implementation for the CIDS data security verification:

- Based on the data security requirements a funCTASM has been created. The network protocol specification was modelled within the funCTASM, and the intercommunication scenarios have been defined as agent configuration. The reason for this decision is the separation of the network protocol knowledge from the agent tasks so that different data security scenarios (flooding, fuzzing, and denial of services attacks ...) can be performed with the same low level network knowledge.
- A standard PC has been selected as a test execution host for this verification (for this use case there was no need for a specific test engine).

- The designed CCM instances have been configured as defined in Section 3.9, but no test engine adapter was required, since we used a standard PC.
- The same test objectives as executed for the RFs have been chosen.

Because of the wide use of scripting in data security testing and because of existing standard approaches the SBT is an established methodology for such testing activities. For this reason Figure 42 shows the noticeable increase of the testing aspects so that the SBT should be one of the favourite approaches.

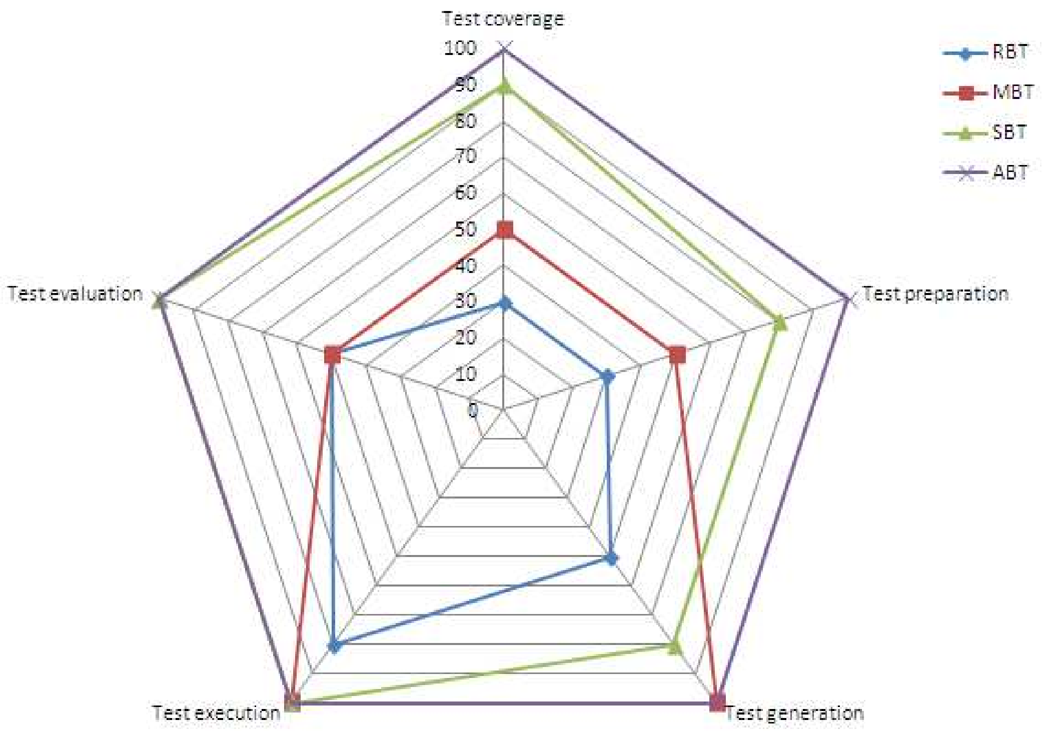


Figure 42 Evaluation of data security verification

<u>Model-based testing (MBT)</u>

Traditional approaches to security evaluation have been based on penetration testing of real systems, or analysis of formal models of such systems. The former suffer from the problem that the security metrics are based on only a few of the possible paths through the system. The latter suffer from the inability to analyse detailed system descriptions, due to the rapid explosion of state space sizes, which render the models intractable for tools such as model checkers. We propose in this work an approach to obtain statistically valid estimates of security metrics by performing testing automats (based on the SCAPY tool which is widely used for data security testing). Because of the limited time and the high complexity of such implementation, as shown in Figure 42, there is no benefit by using this approach. We assume that if the test model (SCAPY-automats) are fully defined or part of the system design process, better results can be achieved.

<u>Agent-based testing (ABT)</u>

For this approach a specific logical configuration of the MAIP has been developed, as shown in Figure 43:

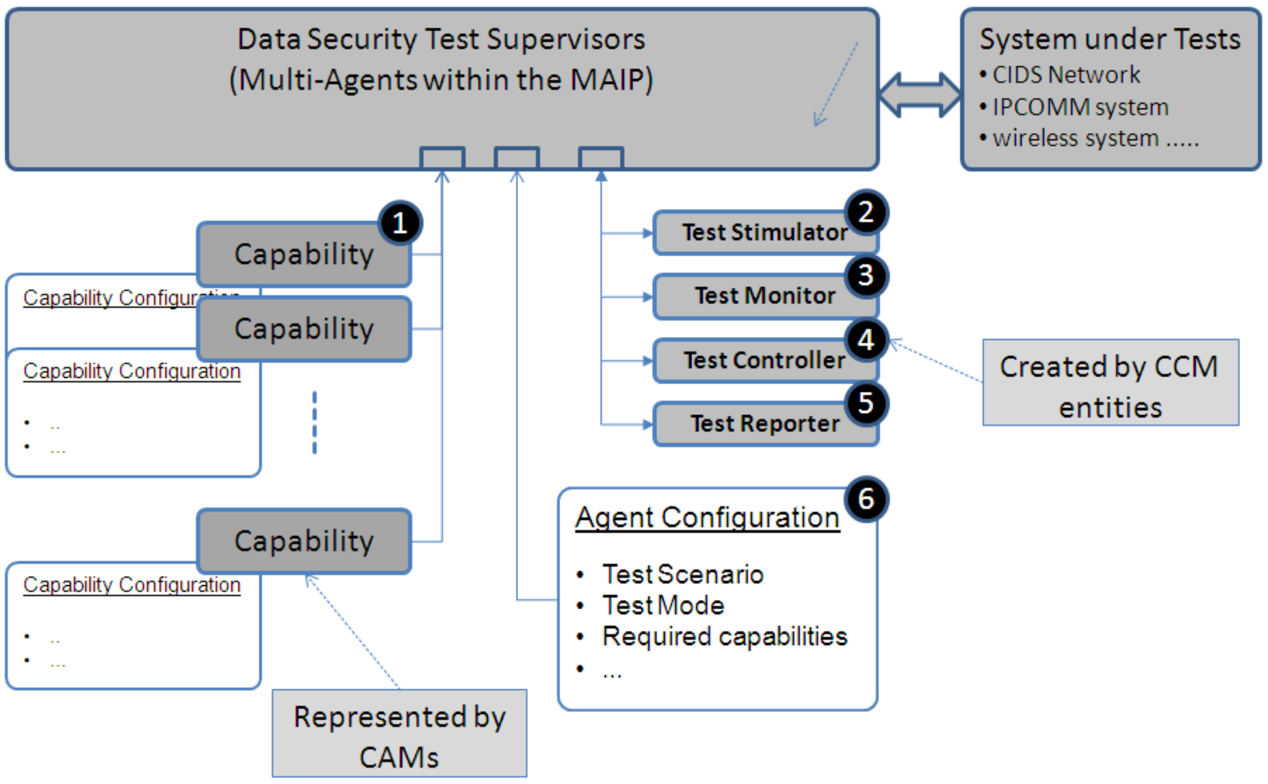


Figure 43 Entities of data security verification

(1) A formal description of the protocol message fields required to establish the network communication.
(2) TCP / UDP protocol-based simulation, including user interaction graphical user interface.
(3) Graphical user interface for the monitoring und manipulation of the protocol message data fields.
(4) Simulation panel for the test run control (initial, run, stop, configuration ...).
(5) Reporting engine to trace each testing step and display their results.
(6) Specific agent configuration editor to define the test mode (functional, flooding, fuzzing ...) and to select the required protocol capabilities (Ethernet, TCP, UDP, Aircraft specific protocols ...).

<u>Criteria-based analysis</u>

<u>Test coverage</u>

For model-based testing, a network model has been partially generated, due to the limitation of a given time of 4 weeks and the difficulties of modelling specific CIDS protocols (STFTP and Heartbeat protocol used to send a sign-of-life signal) so that only 50 % (standard TCP and UDP protocols) of this model have been generated. For the ABT, the generation of the data security functions to be verified (only data security functions to be verified have been modelled 100%) with respect to the requests for test objectives (80 test objectives, 460 test cases) have been performed within 3 weeks. For the SBT the generation of these test cases was performed without critical limitation because there was no need to take into consideration the environmental configuration (90 % have been achieved). Like the other verification, the RBT was the weakest approach due to the complexity of requirements formalisation to define the test requirements and their pass / fail criteria.

Test preparation

The requirement was to perform the test preparation in a specified time of 4 weeks. The ABT and the SBT have achieved this goal, but for SBT a few tasks (10%) required the specific knowhow of scripting (experts) so that 90 % were achieved. The MBT has not achieved this test preparation task because of the required network modelling knowhow (there is no standard SysML model). The RBT approach has not achieved the required objectives with respect to the given time so that only 30 % of test cases were implemented.

Test generation

Similar results have been achieved as shown in the simulation verification.

Test execution

Similar results have been achieved as shown in the simulation verification.

Test evaluation

Unlike the simulation verification, the MBT was impacted by missing the fully defined network models so that only 50% of the test cases could be evaluated. The ABT was the better solution here, due to the fact that the evaluation tasks were decoupled from the easy to generate funCTASM and extended scenario-oriented agent configuration as shown in the verification of system controller in the previous section. It was no problem for SBT to evaluate test results because of the clearly defined pass / fail criteria of the test objectives derived from the data security test request (each test result has been evaluated automatically, like the ABT approach). Because of the fine granularity of the data security requirements and the missing of fully defined test cases, the RBT approach has required additional evaluation methods so that this approach did not achieve this task either.

Test re-configurability

In the nature of data security testing the configuration is a part of the objectives to be tested so that the implemented test cases have to handle different configurations. For this reason the re-configurability has not been taken into consideration in this evaluation task.

Recommendation

For the data security verification the ABT and SBT approaches are preferred, especially if a network model of the system under test is not available.

5.7. Conclusion

Our approach has basically been confirmed by the use cases and is promising for applications in the field of system development. With respect to the evaluation of the use case of the simulation verification it has turned out that only model-based testing could keep up with agent-based testing. The reason is that the simulations have been tested with the same model (MATLAB-model) which was used to create them. All other approaches were far inferior and do not represent alternatives to agent-based testing. The agent-based testing is preferable due to its fast and simple creation of the CAM supported by the XML-based CAM-Editor including its knowledge representation (CTASM), since the classical creation of simulation models is, and will, remain a time-consuming matter due to a lack of adaptivity.

For the controller verification we have chosen a system in which automated testing is well established and was successful within the A380 system testing. Thereby a comparison with the new capabilities of the agent-based testing can be made and the approach can be proved. During the preparation of the test model it has quickly been found how simple and flexible generating the CAM was, compared to the model-based and script-based testing. Despite the known processes of the script-based testing, the approach of the agent-based testing was superior, in particular with respect to re-configurability and test evaluation. The reason was that for new configurations the entire test had to be implemented again, even if part of this implementation was running automatically. The automatic test evaluation was consistently difficult and imprecise, whereas the agent-based approach always led to good results, even if engineers master the handling of abstract models, the agent-based approach is to be recommended in any case. The difficulty of model-based testing was to develop a complete system model within the pre-set time on which re-configurability and test coverage also depend. Requirements-based testing has not been a good approach here either, since testing of single requirements has not provided a conclusion/statement with respect to system maturity.

For multi-system tests, testing of data security within the aircraft cabin has been chosen. There were more than 180 aircraft-specific network protocols which were used during testing. Test scripts and a certain type of low-level programming are required for these tests wherein the script-based testing became accepted during the last few years. Creation of corresponding CAMs requires a high degree of network knowledge and can be made by experts only with respect to the agent-based testing. After creation of the CAMs our approach was superior in all fields, wherein the script-based testing nearly received the same evaluation. All other approaches have not been successful, since their preparation is very time-consuming.

We have chosen three use cases which cover a large spectrum of testing, wherein other approaches have been used as well. While at the beginning of the nineties UML raised high hopes in the field of system development, SysML was promising with respect to system modelling at the beginning of this century, which may accompany the entire system development as far as the verification. Unfortunately, the implementation of such modelling methods suffers from the requirement that the entire system development must be defined again for this purpose. New specification methods, supplier interfaces, validation and verification activities as well as test system adaptation must be defined and established.

Our three use cases clearly showed that model-based testing could hardly be handled within the limited time and cost frame without the above-indicated requirement. Here, the strength of the agent-based testing was effective, since it can also be employed in an environment which is not completely defined, due to its adaptivity and learning aptitude, in particular the state completion explained in Section 4.3. The quality of the agent-based testing is dependent only on the quality of its implementation where only a partial implementation of it was possible within the ABT-Project. Within the frame of the ABT-project activities, the approach presented in this thesis has been implemented as a prototype, so that not all aspects of the agent-based approach could have been demonstrated. However, this prototype implementation was superior to all other approaches and has delivered good results in all evaluation categories so that industrialization of the prototype is recommendable and agreed by the ABT-Project. Though it is not necessary to change the entire development process with respect to the agent-based approach, however, a certain degree of rethinking and handling of abstraction methods is absolutely required as well here. The usage of CAM, CCM and agents show that this new methodology is very promising in a wide area of different test approaches regardless if real or simulated entities are to be tested.

CHAPTER SIX DISCUSSION

6. Discussion

In this thesis we have dealt with two complex subjects; the first one is validation and verification of aircraft systems and the second one is use of software agents for certain problems.

In the last few years the industry has tried again and again to get under control the process of system validation and verification, since, due to the complexity of the process, the classical methods of validation and verification have become inadequate. The complexity results from the high number of parameters and interfaces of the modern network systems which permanently increase in order to cope with the present client requirements.

Different attempts have been made to cope with the validation and verification process and its tasks. Full automation or modeling of test case evaluation and test execution are examples of such attempts only.

The main challenge was the determination of the test volume which had to be sufficient for testing, but also controllable with respect to complexity, time and costs. A problem occurring again and again is that the new test methods and approaches as a sub category of the whole system development must not necessarily lead to a change of the other categories. Another problem of the new methods and approaches is the limitation with regard to time and costs which must not be exceeded during the test process.

The other complex subject was the approach of software agents in the field of system validation and verification. The examination of software agents and their implementation made it possible to outline two capacities (adaptivity and learning capacity) of the software agents which bring forward the selection of software agents for system testing.

Since integration in different environments and, consequently, adaptation of environment are necessary for system testing, a solution has been developed in this thesis which uses the adaptivity of software agents in order to adapt to different environments.

Since testing of systems of different maturities is ensured, the test solution must cope with incomplete information as well, as long as the information left out can be learned (see state completion Section 4.3). In order to solve this problem a solution has been developed in this thesis which uses the learning capacity of a software agent to make the information complete.

To identify and locate errors in a complex environment is a challenge to be handled during the integration tests of this environment. This approach allows for new capacities which can cope with this challenge. This approach differs from other approaches in two above-mentioned basic capacities which help test engineers to perform their tasks, even though the system information available is incomplete. On the other hand, the simplicity (see test specification task in Section 5.2) of the model construction (CAM and included CTASM) helped to prevent generating an expert system to be operable by specifically trained engineers only.

This thesis argued that a promising way to create an effective and smart validation and verification platform is to involve both the user's ability to do easily direct modelling (i.e., to provide an approximately correct representation of the requirements to be fulfilled by the systems to be validated or verified) along with the agent's ability to accept and automatically create validation and verification test cases including the ability to be adaptive in the light of environment changes (see interaction capability of agent and test coverage in Section 5.4.1). Due to the complexity of multi-system nature, the complexity of the interaction and intercommunication in different contexts, a layered system abstraction approach, including the right representation and the accompaniment of context-based learning are more appealing than those solely based on either non-adaptive agent programming languages or large amounts of context-independent agent activities. This chapter presents the contributions of my thesis along with its limitations and some ideas with respect to future work.

6.1. Contributions

The underlying contribution of this thesis are its techniques refined for the use of agent-based approach for the validation and verification of complex systems. The specific contributions of this thesis to each task are listed below:

System abstraction model

For the use of software agents in system development and in particular for testing of complex systems the participating entities need to be modelled in order to ensure software-oriented inter-action. These entities consist of three groups: The entities to be tested (system components, system, multi-system functions) the testing entities (test engine I/O, simulation host, test scripts etc.) and additionally the test managing entities (messaging, notification, reporting inter-communication). In this thesis the abstraction models including its knowledge representation have been developed so that all the afore-mentioned entities can be integrated in a functioning environment (MAIP). Thereby it was taken into account that the entities to be tested are specified, developed and tested differently. For example, the system components are the sole entities which are manufactured and specified in detail so that plenty of knowledge can be gained from their specification. Thereby it must be considered that system components are specified and manufactured in different phases of different stages of expansion, which leads to the incremental availability of this knowledge. Systems are more roughly defined and form a logical clustering of system components wherein knowledge must be learnt with the support of the software agents. Multi-system functions are a further logical clustering of system component operations wherein knowledge is roughly available only so that learning is the most important source of information here as well. For the entities to be tested three abstraction models have been chosen, CAM for system components modelling, sysCAM for systems and funCAM for multi-system functions which have also been located in three different layers, wherein communication between layers is made possible by specific interfaces via agents.

For abstract modelling of testing and managing entities a generic approach has been chosen which can also be used for complex tests, by transferring several testing and managing entities into a large multi-MAIP.

By this modelling it is ensured that software entities and in particular software agents are enabled to interact with all entities; thus, the abstraction modelling represents the first contribution to this thesis. Among others, generation, loading, monitoring, controlling and commanding of such entities belong to the above-mentioned interactions.

The significant contribution of the system abstraction model is its ability to utilise the following three sources of information when learning to convert from general requirements to be fulfilled to a specific layer-based knowledge base to be able to provide an agent's instances with the required information to perform its validation and verification tasks.

- System component specification
- System definition (logical composition of system components)
- Function description (cooperation of multi-systems to perform a required inter-system function).

Due to the different types of specification of the respective entities to be tested, it was necessary to develop different methods for the knowledge representation. Usually , the system components as a specific case are specified in detail so that their states and state transitions are known. In addition, their operations are also specified by their valid states. The abstract state machine (ASM) of [Alur 1998] is suitable for representing the states and their transitions. Since within this thesis we needed the time-depending component and the linking of the operation with the valid states so that the software agents can cover the whole test scope, we have extended the defined ASM (see CTASM in Section 4.2.1). This extension has added two new aspects to the ASMs. The first aspect is the time dependency of the states and the state transitions, and the second aspect is that it must be ensured that during state or state transition operations can be executed on a time-controlled basis.

Since the CTASM is integrated in a CAM, all the interface information defined in the CAM is valid and available so that incorrect allocation can be excluded. By allocating a CAM and its integrated CTASM to software agent all the information required for its test activities is available for him.

The advantages of the layered abstraction are supplemented by a specific knowledge representation specifically performed for each layer, so that different sources and types of information can be represented in an appropriate way. The improvement ASM mentioned above by the extension with the coded operation (CTASM) provides agents with the capabilities to test system operations (system transfer functions) regarding their behaviours, state validities and time conditions in one task.

As discussed in Section 4.2.2, this thesis defined a new sysCTASM on the basis of the defined CTASM to represent the knowledge on a system. The sysCTASM has its own states and state transitions which can be defined from the composition of defined CTASM (see Section 4.2.2) which has been ensured by the flexibility and the combinability of the CTASM basic model. By means of the sysCTASM software agents can perform their test activities in the same way, independent of the layer in which they are.

Although multi-system functions require other compositions of the operations of the system component, for the definition of the funCTASM (see Section 4.2.3) the same procedure as used for the sysCTASM can be used. Implicitly software agents can perform their test activities also here and in this layer.

During the MAIP implementation the use of CTASM has proved to be very flexible so that additional potential layers can be modeled without great efforts and without new software agents, which represents our second contribution to this thesis.

<u>Context-based learning</u>

Due to the different modelling of the software component, systems and multi-system functions and the different contexts in which they act, the software agents have been modelled as context-oriented so that they perform different tasks depending on the context in which they are.

When the software agents have realised in which context they are, they can perform different activities, like testing, observing, reporting and also learning. Since the software agents can plan their activities, and the corresponding tasks are defined with respect thereto, they can optimise their strategy and its tasks in a certain context with respect to the respective time and resources they need. For testing this capacity is very useful, since the same test cases can be performed in different contexts and environments which leads to a better test depth. The adaptivity and learning capacity of the software agents can also use this capacity and observe different aspects in which system component, systems and multi-system functions act.

Learning by software agents has been implemented in this thesis on the basis of pattern discovery algorithms mainly, state completion and is adaptive with respect to the different layers and contexts in which it was activated (see Section 4.3). Pattern discovery within the system component layer deals with the internal operations of the system component and its interfaces, which functions mainly represent the interaction between the system components.

After new information has been discovered, a so-called knowledge establishing process is starting which deals with the validation of this information (see Section 3.8.3). Only knowledge which has proved to be valid may be used by software agents in the verification process. Knowledge which has not completely been validated and not been refuted either, may still be used in pattern discovery, test development and validation processes. Compiled knowledge which has not been validated or refuted may not be used by software agents.

The precondition for learning within the system component layer is the defined interfaces of the system component. Within the system layer these are the defined CAMs, and within the function layer these are the participating sysCAMs and their CAMs.

Due to the fact that the system under test entities are differently specified and act differently in different layers, it was helpful to actuate the agent learning process by refining it to act differently in different functional contexts. Three main advantages are to be mentioned:

- Reduction of learning tasks within one context.
- Linking the knowledge relevance with a certain context ensured that the knowledge can be differently weighted and used, depending on the context in which it is used. This linking made it easier to manage the huge amount of knowledge gathered by the agent learning process.
- Learned knowledge can be allocated to the right valid contexts even if it is not globally valid. Without this allocation it was not possible to establish knowledge when this knowledge was not globally validated.

6.2. Limitations

The main limitations of the agent-based approach:

Based on the review results of the ABT-project test teams, the following limitations have been identified:

- Detailed knowledge with respect to the interfaces of the system components and their parameter characteristics is a precondition for the success of the pattern discovery within this layer.
- For systems which can only be verified with verified tools (for the certification of critical aircraft systems) it is very difficult or almost impossible to verify adaptive tooling with learning

capabilities. The reason for this difficulty is caused by the requirements of a deterministic ensured by specific tool which is naturally not given in the agent-based tool implementation.
- For the definition of the abstraction model an abstract thinking capability is expected from users to be able to define the right model for a certain test objective.
- The advantage of this approach is noticeable and measurable when a certain level of system under test complexity is achieved. For simple systems there is no value to use this approach because of the relatively high preparation effort required to be able to use it.
- This approach is focused on the use of systems which are computer-controlled; for other systems without data exchange interfaces another agent-based approach is required.

6.3. Final Remarks and Future Directions

In order to achieve an effective solution of the verification problems of complex systems, it is desirable to start as early as possible (in the design phase already) so that many verification aspects with respect to all layers can be considered. Such a requirement implies that an approach like ours can generate flexible and complete CAMs on which the whole modeling of the MAIP is based. The quality of the CAMs on the system component level directly determines the quality of the agent-based verification and simultaneously reduces the requirements with respect to the learning capacity of the software agents.

Different from any other existing verification framework, our platform includes layered interfaces and a verification platform. The former link designs are represented at multiple levels of abstraction to the verification platform. The platform is the central element of the verification system where the test engine is integrated. CAM generation, response checking and the test case execution supported by the CCM and managed by the MAIP are all sub-components of this platform.

<u>Vision of the future:</u>

This section lists several topics that are not addressed in this thesis. Each topic would clearly benefit from further investigation and, hopefully, would make the MAIP methodology stronger. Candidate topics for future investigation are:

- How to utilise the methodology with special domains such as mechanical, acoustical, electromagnetical systems, etc.?
- How to perform verification for the resulting agent system software?
- How to deal with issues related to the discovering of multi-system functions by estimation or identification?
- How can software agents in system development be used for design optimisation, in particular with respect to minimisation of the resources and optimisation of the communication performance?
- Another aspect would be to extend the verification platform with a hardware-in-the-loop interface to complete the link between abstraction levels. With this hardware interface, the test engine will be reused for in-circuit validation and verification.

The usage of agent-based approach within the system domain is not established and very limited so that documentation and publication are limited as well. This thesis demonstrated the experimental successes of MAIP on which more researches to investigate agent-based approaches and learning systems in the system development domain can be based. In future researches the opportunities for using these approaches in the real aircraft instrumentation can be analysed with respect to their usage for the monitoring and failure detection during aircraft ground and flight test activities.

The contribution of this thesis regarding layered system abstraction and context-based learning are promising and can be the basis of an application for ground and flight testing, in particular for reduction on complexity and time costs. An overview of a future approach is shown in Figure 44.

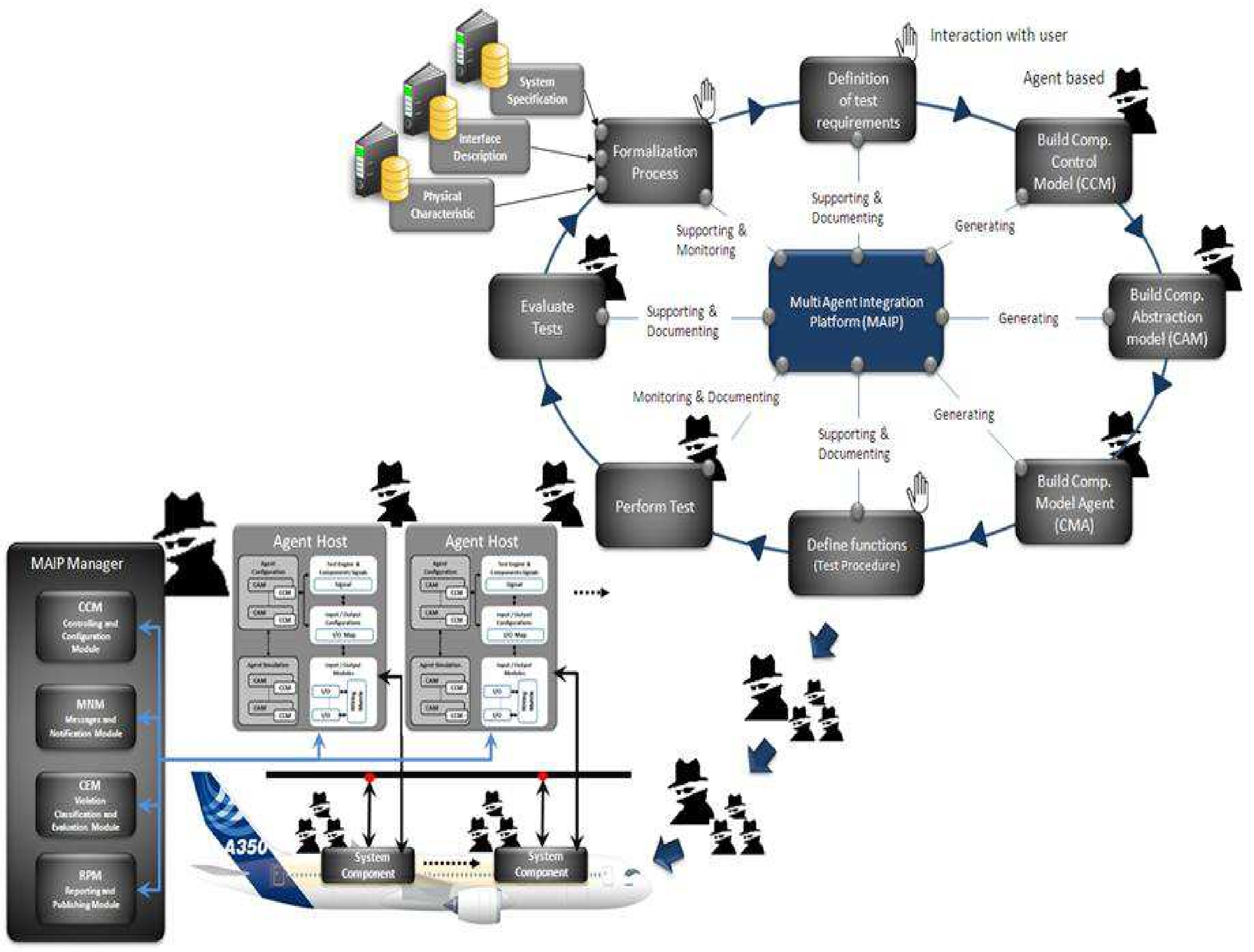

Interaction with user
Agent based
System Specification
Interface Description
Physical Characteristic
Formalization Process
Definition of test requirements
Build Comp. Control Model (CCM)
Supporting & Monitoring
Supporting & Documenting
Generating
Evaluate Tests
Multi Agent Integration Platform (MAIP)
Build Comp. Abstraction model (CAM)
Monitoring & Documenting
Perform Test
Define functions (Test Procedure)
Build Comp. Model Agent (CMA)
MAIP Manager
CCM Controlling and Configuration Module
MNM Messages and Notification Module
CEM Violation Classification and Evaluation Module
RPM Reporting and Publishing Module
Agent Host
System Component
A350

Figure 44 Ground and flight aircraft testing approach

<u>Proposal for concrete potential future works</u>

The application of the software agent approach to the area of complex system validation and verification was the goal of this dissertation. This thesis shows that the knowledge representation of the system to be tested is an essential issue in order to enable software agent computation. The evaluation of this thesis has pointed out that the CTASM shall fulfil many requirements to be mature enough for reasonable use within an agent-based approach. In addition, the learning capabilities of software agents have been demonstrated, based on the state completion and signal pattern discovery (see Section 4.3). One of the major problems of having a mature CTASM is that at times no mature CTASM can be modelled because of time constraints or limited availability of detailed information with respect to the system to be tested. The second problem is the knowledge establishing process that needs to be validated by the system designer (see Section 3.8.3). To solve these major problems and to improve the CTASM modelling process, the following aspects could be potential topics which could be investigated and analysed in future research:

- Building of principle models that represent the main functionality and behaviours of the system under test; these models can be improved automatically by specific agents. These specific agents can use discovery approaches with the goal to transfer the principle model into a mature CTASM. In this sense, the task a designer shall perform can only be a creation of the correct principle model of the system under test, and the completion of the CTASM shall be done by agents implicitly.

- During the thesis evaluation task regarding the knowledge establishing process it was difficult to differentiate between design error and implementation error. The reason for that was the immaturity of the system specification and of system implementation. In order to be able to reduce the dependency on the validation activity to be performed by the system designer (assuming that the system specification is mature), validation algorithms can be developed to ensure the following checks (against system specification only) when the principle model has been improved or extended in order to focus on implementation failures :

    - Model consistency.
    - Model completeness.
    - Model correctness.

Appendix A

System Component basic model

Systems can be represented as a composition of system components by using SysML block diagrams. The ***«block»*** is the basic unit of structure in SysML and can be used to represent systems, software, facilities or any system component. The system component is represented by block definition diagrams and internal block diagrams. A block definition diagram describes the system hierarchy and system component classifications.

The internal block diagram describes the internal structure of a system in terms of its parts (components), ports, and connectors. The package diagram is used to organize the model. Figure A.1 shows the block definition diagram **(bdd)** that describes the relationship (e.g., composition, association, and specialization) among blocks (components). A modified database basic representation of system components and their relations has been used in this thesis (see Figure A.2).

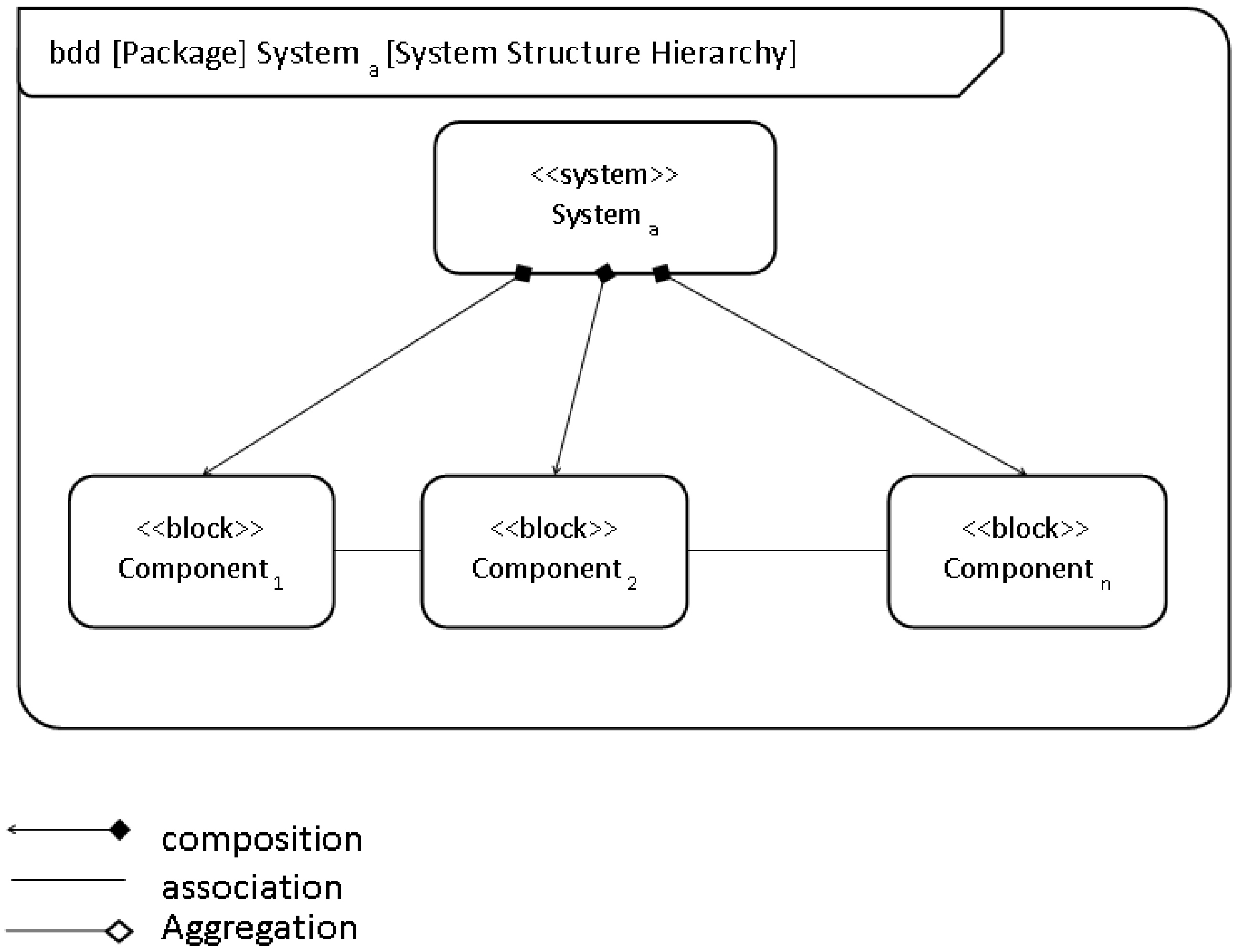


Figure A.1 System components represented by SysML block diagram

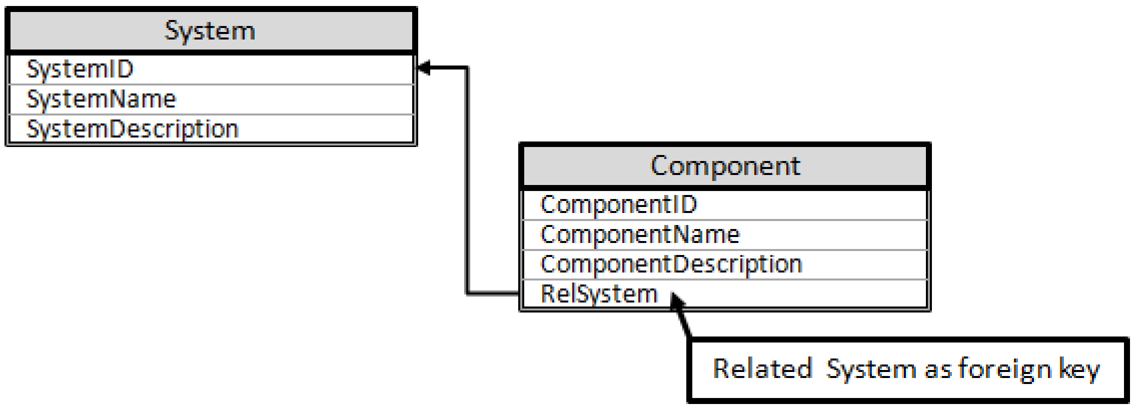


Figure A.2 Database-oriented system components representation

The description of the components can be represented by SysML blocks (basic structural elements) which provide a unifying concept to specify the structure of a system component. Multiple standard compartments can describe the block characteristics:

- Properties (parts, references, values, ports)
- Operations
- Constraints
- Allocations from/to other model components of elements (e.g. activities)
- Requirements which the block satisfies
- User defined compartments (an optional extension if necessary)

In Figure A.3 the block diagram depicted in Figure A.1 can be detailed by using the block description to represent the system components, their properties, operation constraints, allocations from/to other model components (or requirement models) and the requirements to be satisfied by them.

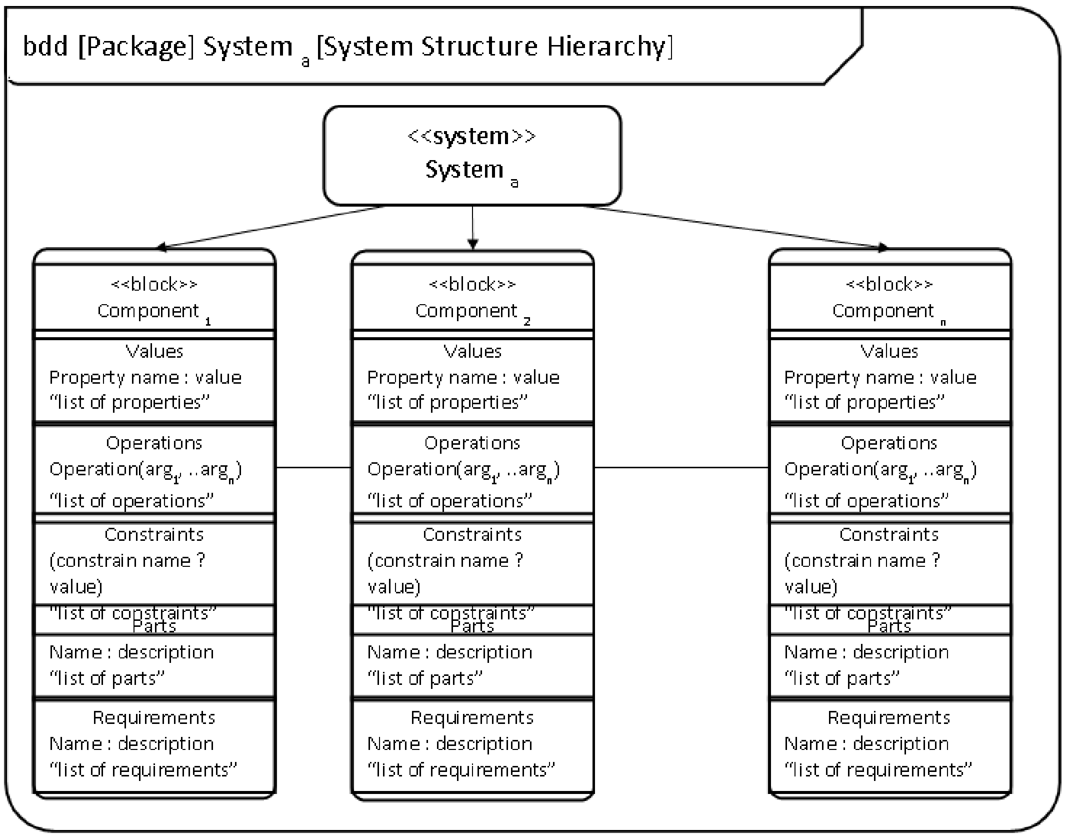


Figure A.3 System component block description

A modified database-oriented representation of system component block description will be used in this thesis (see Figure A.4). This database-oriented specification will be extended to specify the system components and their characteristics (Component Based System Specification CBSS). Based on this CBSS, the agents, agent adapters and the supporting simulation will be realized to build the prototype of the MAIP (Multi-agent integration platform). The basic CBSS will be extended and used to specify the detail system component (to be abstracted) implementation of the MAIP in the subsequent chapters.

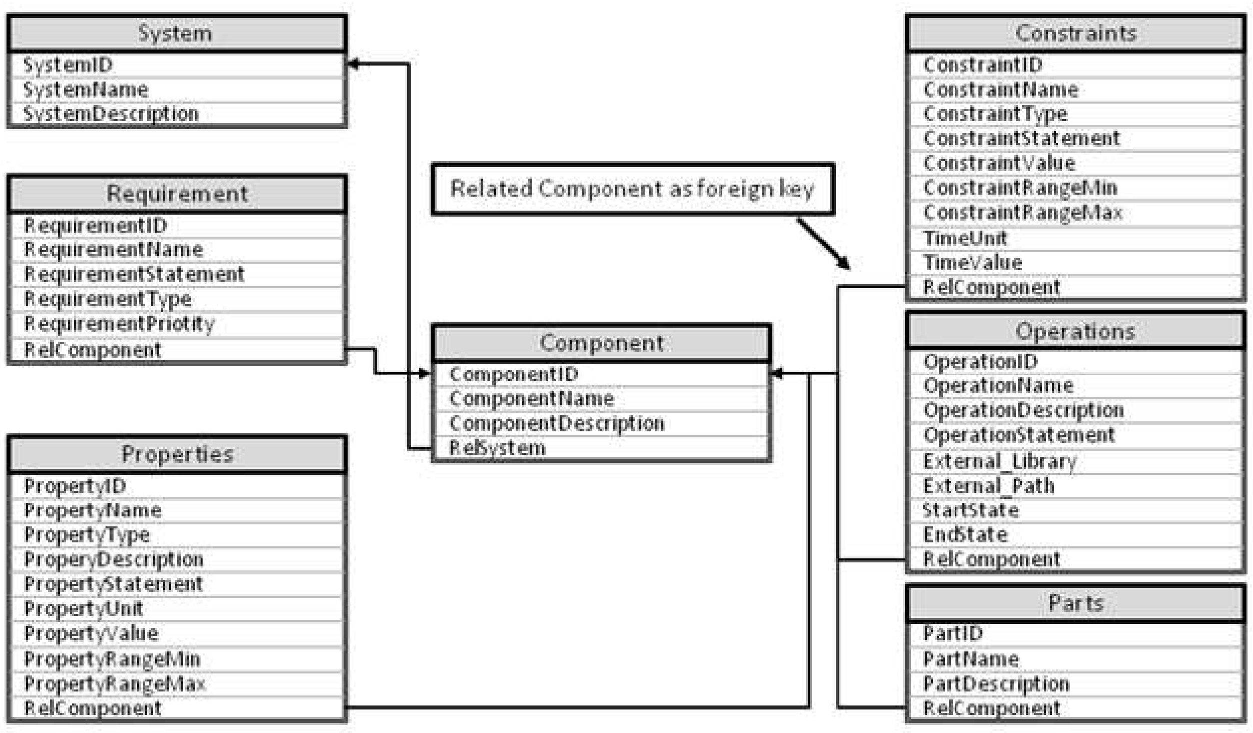


Figure A.4 Database-oriented representation of the system component description

To describe the component internally the internal block diagram **(ibd)** of SysML can be used. The ibd specifies the parts (function), interconnection (interfaces), connectors and flows (signal flow) enclosed by the component. In this thesis a component to be verified will be specified by the functions, their interfaces and the signal flow among them.

To describe the component internally by using the database oriented representation, two additional tables will be added to the database schema. The first table “parameters” represents the input, output and control signals including the link to the respective operation (component function). The second added table is the constraint specification including the link to the respective operation.

In this thesis a component to be verified will be specified by the functions, their interfaces and the signal flow among them. In Figure A.5 the basic component specification by using database specification is depicted.

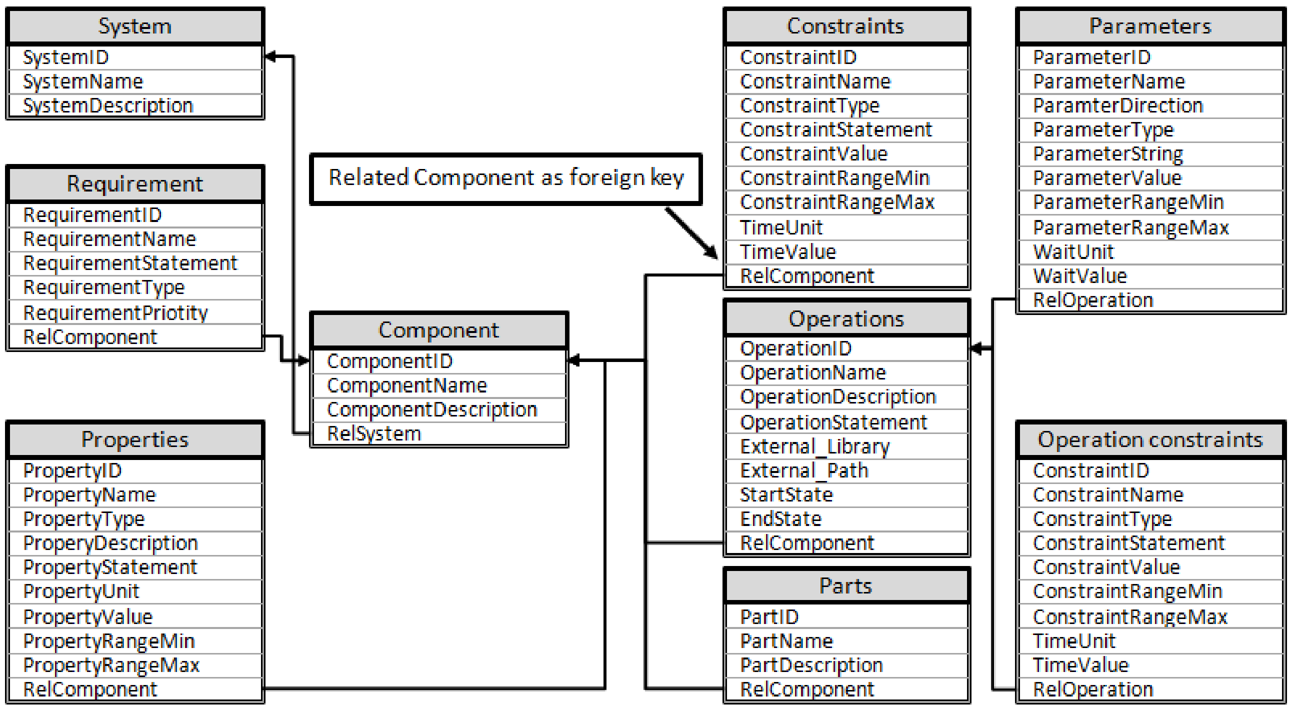


Figure A.5 System component internal database oriented specification

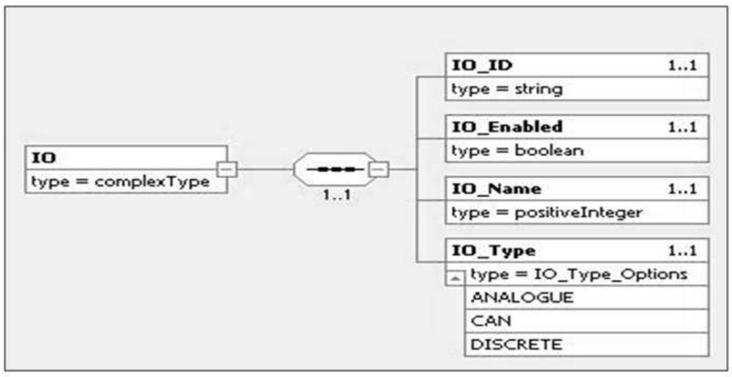
In this thesis basic database representation of the system component has been used and extended to demonstrate the concept of the component abstraction model and its interfaces as described in Chapter 3.7.2.

Appendix B

I/O Specification

I/O CONFIGURATION

The I/O configuration is a description of all integrated I/O modules in the test systems which will be managed by an I/O configuration file with the following describing information and format:



```
<IO>
   <IO_ID> xs: string </IO_ID>
   <IO_Enabled> xs: boolean </IO_Enabled>
   <IO_Name> xs: positiveInteger </IO_Name>
   <IO_Type> IO_Type_Options </IO_Type>
</IO>
```

Figure B.1 I/O Configuration file

HAL Commands:

GET_CHANNEL_COUNT

It will be used to get the count of channels hosted by the I/O module indentified by the command parameter.

Signature of the GET_CHANNEL_COUNT

<SYNOPSIS>

- **int HAL_GetChannelCount ("IO_Name")**
- **int HAL_GetChannelCount ("IO_ID")**

<DESCRIPTION>

Get_Channel_Count gets the count of the channels hosted by a certain I/O module.

<RETURNS>

On success, the count of channels (a positive integer number)
On error, 0 is returned

SET CONVERSION POLICY

It will be used to set the conversion policy of the I/O channel to specify how often data shall be converted. The Channel will be identified by the command parameters.

Signature of the SET_CONVERSION_POLICY

<SYNOPSIS>

- **int HAL_ SetConversionPolicy (“IO_ID”; “IO_Channel_Number”; “Policy_Option”)**
- **int HAL_ SetConversionPolicy (“IO_Name”; “IO_Channel_Number”; “Policy_Option”)**

Two policy options are allowed:

- **Always the value of the I/O channel will be applied and written to signal even if the value has not changed.**
- **Event: the value of the I/O channel will be applied and written to signal only if the value has changed.**

<DESCRIPTION>

SetConversionPolicy sets the channel policy option.

<RETURNS>

On success, 1 is returned
On error, 0 is returned

SET SCALE POLICY

It will be used to set the scale policy of the I/O channel to specify how data shall be scaled. The default scale factor will be 1.

Signature of the SET_SCALE_POLICY

<SYNOPSIS>

- **int HAL_ SetScalePolicy (“IO_ID”; “IO_Channel_Number”; “Scale_factor”)**
- **int HAL_ SetScalePolicy (“IO_Name”; “IO_Channel_Number”; “Scale_factor”)**

<DESCRIPTION>

SetScalePolicy sets the channel scale factor which is a positive integer number.

<RETURNS>

On success, 1 is returned
On error, 0 is returned

SET OFFSET POLICY

It will be used to set the offset policy of the I/O channel to specify how the offset of data shall be set. The default offset factor will be 0. It is a multiple of frame or milliseconds with the default value 0 (no delay).

Signature of the SET_OFFSET_POLICY

<SYNOPSIS>

- **int HAL_ SetOffsetPolicy (“IO_ID”; “IO_Channel_Number”; “Offset_factor”; “Offset_unit”)**
- **int HAL_ SetOffsetPolicy (“IO_Name”; “IO_Channel_Number”; “Offset_factor”)**

Two offset unit options are allowed:

- **MS: for milliseconds.**
- **Frame: for frames.**

<DESCRIPTION>

SetOffsetPolicy sets the channel offset factor which is a positive integer number where the offset unit is specified by Offset_unit.

<RETURNS>

On success, 1 is returned
On error, 0 is returned

SET_SAMPLE_RATE.

It will be used to set the sample rate of the I/O channel to specify how often the data is read. It is a multiple of frame rate with the default value 1 (one times every frame).

Signature of the SET_SAMPLE_RATE

<SYNOPSIS>

- **int HAL_ SetSampleRate (“IO_ID”; “IO_Channel_Number”; “Sample_factor”)**
- **int HAL_ SetSampleRate (“IO_Name”; “IO_Channel_Number”; “Sample _factor”)**

<DESCRIPTION>

SetSampleRate sets the channel sample factor to specify how often the data is read.

<RETURNS>

On success, 1 is returned
On error, 0 is returned

GET_MIN_RANGE

It will be used to get the minimum allowed value of I/O channels hosted by the I/O modules. The IO_ID or IO_Name (defined in the I/O configuration file) and the channel number can be used as parameter for this command.

Signature of the GET_MIN_RANGE

<SYNOPSIS>

- **float HAL_ Get_Min_Range (“IO_ID”; “IO_Channel_Number”)**
- **float HAL_ Get_Min_Range (“IO_Name”; “IO_Channel_Number”)**

<DESCRIPTION>

Get_Min_Range gets the channel minimum allowed value.

<RETURNS>

On success, the minimum value of I/O channel as a float is returned
On error, 0 is returned

GET_MAX_RANGE

It will be used to get the maximum allowed value of I/O channels hosted by the I/O modules. The IO_ID or IO_Name (defined in the I/O configuration file) and the channel number can be used as parameter for this command.

Signature of the GET_MAX_RANGE

<SYNOPSIS>

- **float HAL_ Get_Max_Range (“IO_ID”; “IO_Channel_Number”)**
- **float HAL_ Get_Max_Range (“IO_Name”; “IO_Channel_Number”)**

<DESCRIPTION>

Get_Max_Range gets the channel maximum allowed value.

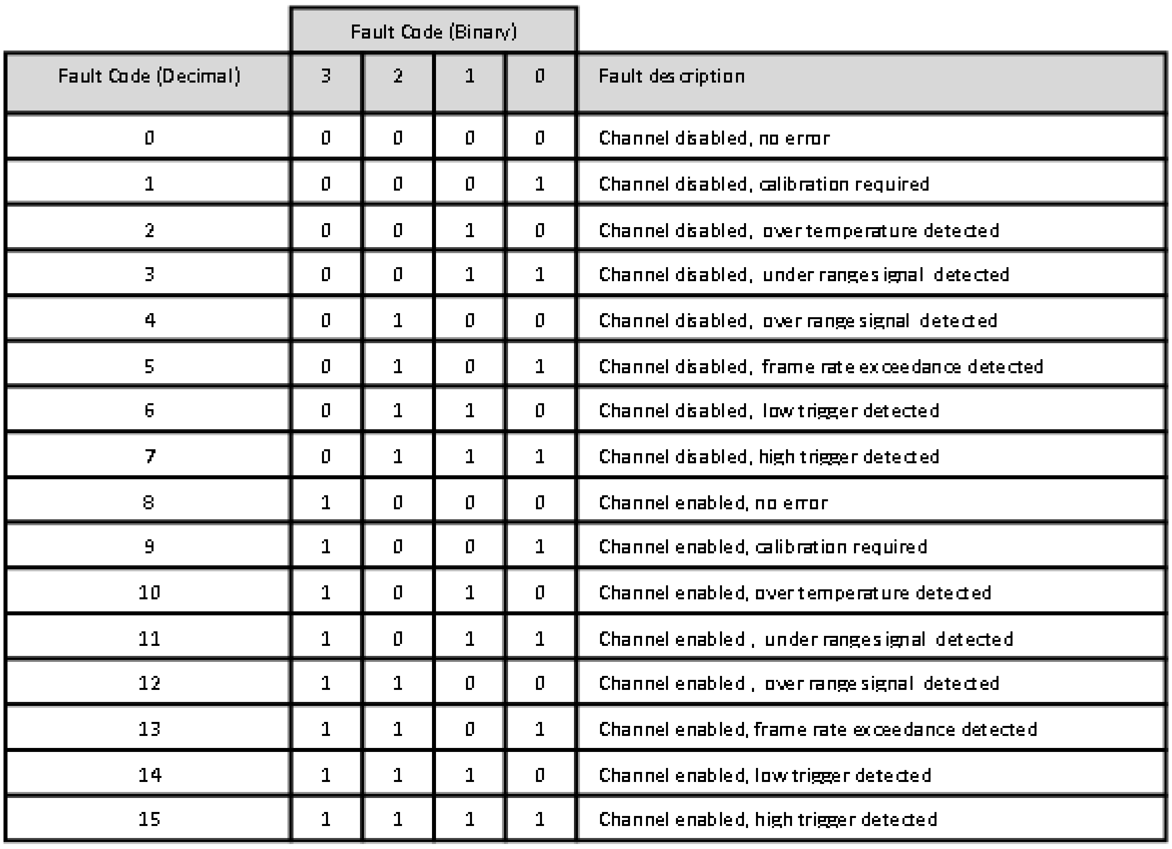
<RETURNS>

On success, the maximum value of I/O channel as a float is returned
On error, 0 is returned

I/O Monitoring Command

GET   CHANNEL STATUS

It has been used to get the fault word bits values of I/O channels hosted by the I/O modules. The IO_ID or IO_Name (defined in the I/O configuration file) and the channel number can be used as parameter for this command. Following table shows the minimum required fault code:



	Fault Code (Binary)				
Fault Code (Decimal)	3	2	1	0	Fault description
0	0	0	0	0	Channel disabled, no error
1	0	0	0	1	Channel disabled, calibration required
2	0	0	1	0	Channel disabled, over temperature detected
3	0	0	1	1	Channel disabled, under ranges signal detected
4	0	1	0	0	Channel disabled, over range signal detected
5	0	1	0	1	Channel disabled, frame rate exceedance detected
6	0	1	1	0	Channel disabled, low trigger detected
7	0	1	1	1	Channel disabled, high trigger detected
8	1	0	0	0	Channel enabled, no error
9	1	0	0	1	Channel enabled, calibration required
10	1	0	1	0	Channel enabled, over temperature detected
11	1	0	1	1	Channel enabled , under ranges signal detected
12	1	1	0	0	Channel enabled , over range signal detected
13	1	1	0	1	Channel enabled, frame rate exceedance detected
14	1	1	1	0	Channel enabled, low trigger detected
15	1	1	1	1	Channel enabled, high trigger detected

Figure B.2 I/O Fault code

Signature of the GET   CHANNEL STATUS

<SYNOPSIS>

- **int MON_ Get_Channel_Status (“IO_ID”; “IO_Channel_Number”)**
- **float HAL_ Get_Channel_Status (“IO_Name”; “IO_Channel_Number”)**

<DESCRIPTION>

Get_Channel_Status gets the channel status.

<RETURNS>

On success, the fault code as integer representing the I/O channel status (see fault code table above) is returned
On error, 0 is returned

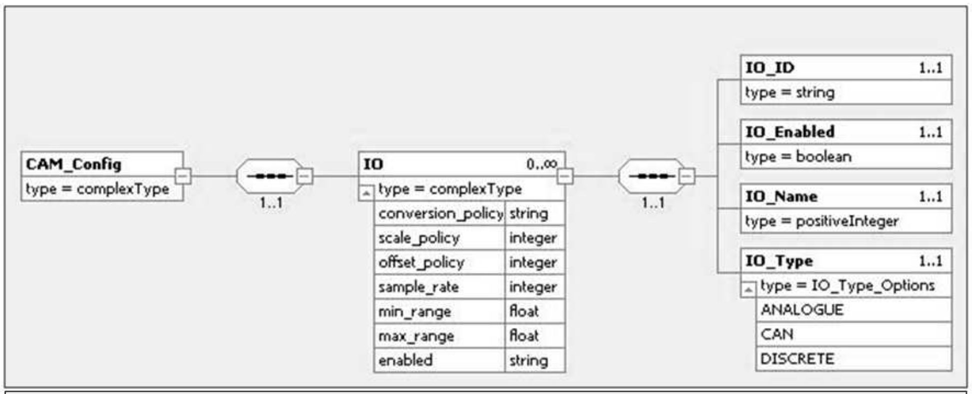
Appendix C

CAM Configuration file

The aim of the CAM configuration entity has been the generation of the CAM configuration file which specifies the CAM with respect to the following aspects:

- Definition of the system component interfaces to be instrumented in the MAIP environment.
- Specification of the system component interfaces characteristics required for the interface instrumentation.
- Configuration of the system component interfaces to be instrumented in the MAIP environment by using the HAL commands.
- Definition of the under-laying layer of the CCM entity.

The CAM configuration file with describing information and format is depicted in Figure C.1:



```
<CAM>
      <IO conversion_policy="xs:string„   scale_policy="xs:integer"
            offset_policy="xs:integer„ sample_rate="xs:integer "
            min_range="xs:float "  max_range="xs:float" enabled="xs:string" >
            <IO_ID> xs: string </IO_ID>
            <IO_Enabled> xs: boolean </IO_Enabled>
            <IO_Name> xs: positiveInteger </IO_Name>
            <IO_Type> IO_Type_Options </IO_Type>
      </IO>
</CAM>
```

Figure C.1 CAM configuration file

CAM Data base Model Schema

REQUIREMENTS

The general system component internal database oriented specification (see Appendix A) has been modified for the database representation of the CAM. In the first phase the component requirements have been classified and defined in the database table “requirements” with the following attributes:

- RequirementID: A global unique ID identifying the requirement in the MAIP context.

- RequirementName: A descriptive name of the requirement.
- RequirementStatement: This is a formal expression of what the component shall fulfil (functionally or non-functionally).
- RequirementType: Each requirement can be grouped to one of the following classes:

  - Operation: for **functional** requirements to be fulfilled by this component (operations will be specified in a separate table and linked to the related requirement).
  - Constraint: for **non-functional** requirements to be fulfilled by this component (constraints will be specified in a separate table and linked to the related requirement).
  - Property: for **static** requirements to be fulfilled by this component (properties will be specified in a separate table and linked to the related requirement).

- RelComponent: The link of this requirement to the component to which it is allocated.

In the second phase the requirements shall be prioritized (weighted). Therefore the requirements table will be extended with the following attributes:

- RequirementPriority: the weighting of the requirements priority. The following options are valid:

  - Very high
  - High
  - Middle
  - Low
  - Very low.

In the third phase the requirement dependency has been realized by adding a new dependency table in the database "req_dependency" to persist the relation to other requirements in the MAIP context. This table consists of two attributes:

- RequirementID: A globally unique ID identifying this requirement in the MAIP context.
- InterrelatedRequirment: A link to the other requirement which can also be allocated to another component.

The result of the first three phases of the component abstraction model is depicted in Figure.2:

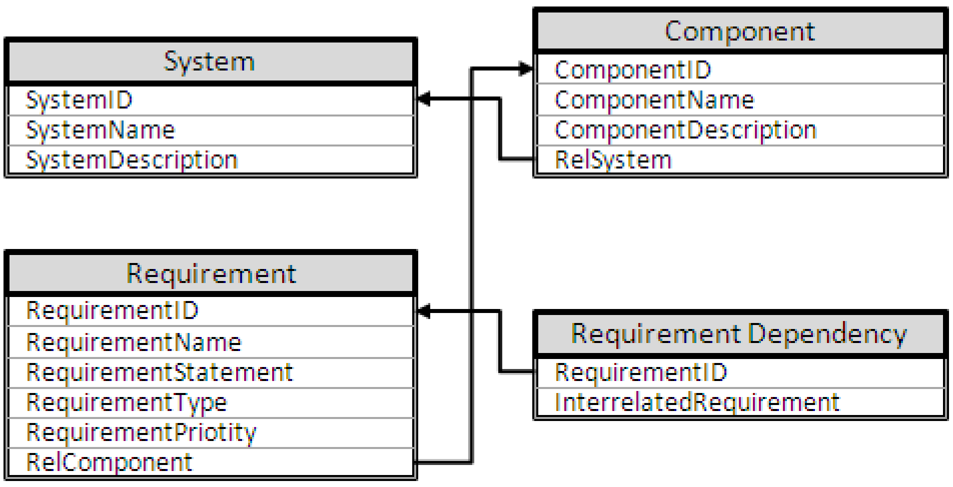


Figure C.2 MAIP-database schema of component abstraction model

For the modelling of the static requirements a new table "properties" has been created to persist the static properties of the component with the following attributes:

- PropertyID: A global unique ID identifying the property in the MAIP context.

- PropertyName: A descriptive name of the property.
- ProperyDescription: A description of the property.
- PropertyStatement: This is a formal expression of what the component shall fulfil (statically).
- PropertyValue: The default value of the property if defined.
- PropertyRangeMin: The minimum allowed value of the property if defined.
- PropertyRangeMax: The maximum allowed value of the property if defined.
- RelComponent: The link of this requirement to the component to which it is allocated.

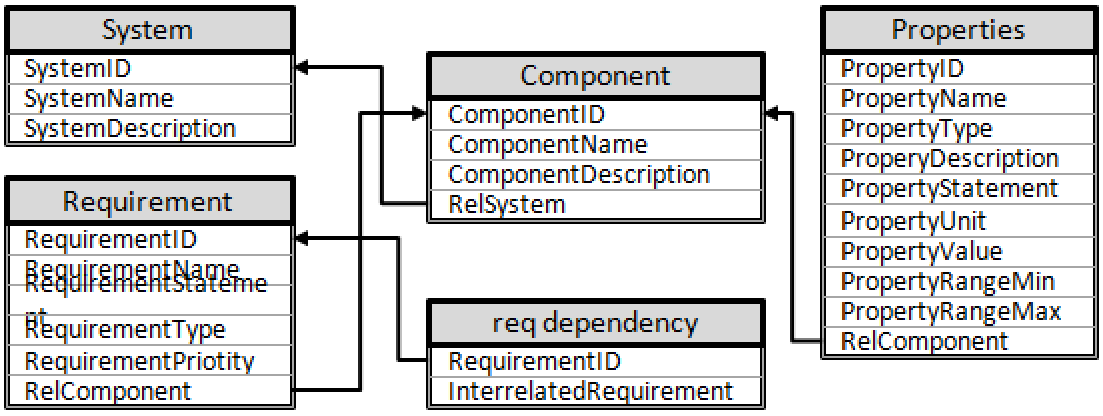


Figure C.3 MAIP-database schema of CAM properties

The result of the first requirements extension of the CAM is depicted in

Figure. The next step was the modelling of the functional requirement by the definition of the operations, their parameters and operation constraints. Therefore, three new tables have been created. The first table “operation” has represented the function to be performed by the component with the following attributes:

OperationID: A global unique ID identifying the operation in the MAIP context.
OperationName: A descriptive name of the operation.
OperationDescription: A description of the operation.
OperationStatement: This is a formal expression of what the component shall fulfil (functional).
RelComponent: The link of this operation to the component to which it is allocated.

To establish a link between the operation and the related requirement a database table “operation-requirement” has been defined to persist this in the MAIP context. This table consists of two attributes:

- RequirementID: A global unique ID identifying this requirement in the MAIP context.
- OperationID: A global unique ID identifying this operation in the MAIP context.

For the operation parameters (inputs, outputs, controls and waits) a new table has been created with the following attributes:

ParameterID: A global unique ID identifying the parameter in the MAIP context.
ParameterName: A descriptive name of the parameter.
ParamterDirection: The direction of the parameter wherein the following options are allowed:

- Wait: to determine that the parameter shall be considered as a wait statement within the operation.
- Control: to determine that this parameter is a control signal.
- Output: to determine that this parameter is an output signal.
- Input: to determine that this parameter is an input signal.

ParameterType: Three options are allowed (“String”;”Real”;”Integer”)
ParameterString: To be set for string parameters.
ParameterValue: If the parameter is not a string, the value shall be set.
ParameterRangeMin: The minimum allowed value of the parameter if defined.
ParameterRangeMax: The maximum allowed value of the parameter if defined.
WaitUnit: If the parameter direction is “wait”, the wait unit shall be defined. The following options are allowed:

- Frame
- Millisecond
- Second.

WaitValue: If the parameter direction is “wait”, the wait value shall be set.
RelOperation: The link of this parameter to the operation to which it is allocated.

For the operation constraints with respect to the performance requirements and the interface violation, a new table “operation_constraint” has been created with the following attributes:

ConstraintID: A global unique ID identifying this constraint in the MAIP context.
ConstraintName: A descriptive name of this constraint.
ConstraintType: Three options are allowed (“String”;”Real”;”Integer”;”Time”).
ConstraintStatement: This is a formal expression of what the operation shall fulfil (performance).
ConstraintValue: If the constraint is not a statement, the value shall be set.
ConstraintRangeMin: The minimum allowed value of this constraint if defined.
ConstraintRangeMax: The maximum allowed value of this constraint if defined.
TimeUnit: If the constraint type is “time”, the time unit shall be defined. The following options are allowed:

- Frame
- Millisecond
- Second.

TimeValue: If the constraint type is “time”, the time value shall be set.
RelOperation: The link of this constraint to the operation to which it is allocated.

The result of the second requirements extension of the CAM is depicted in Figure C.4.

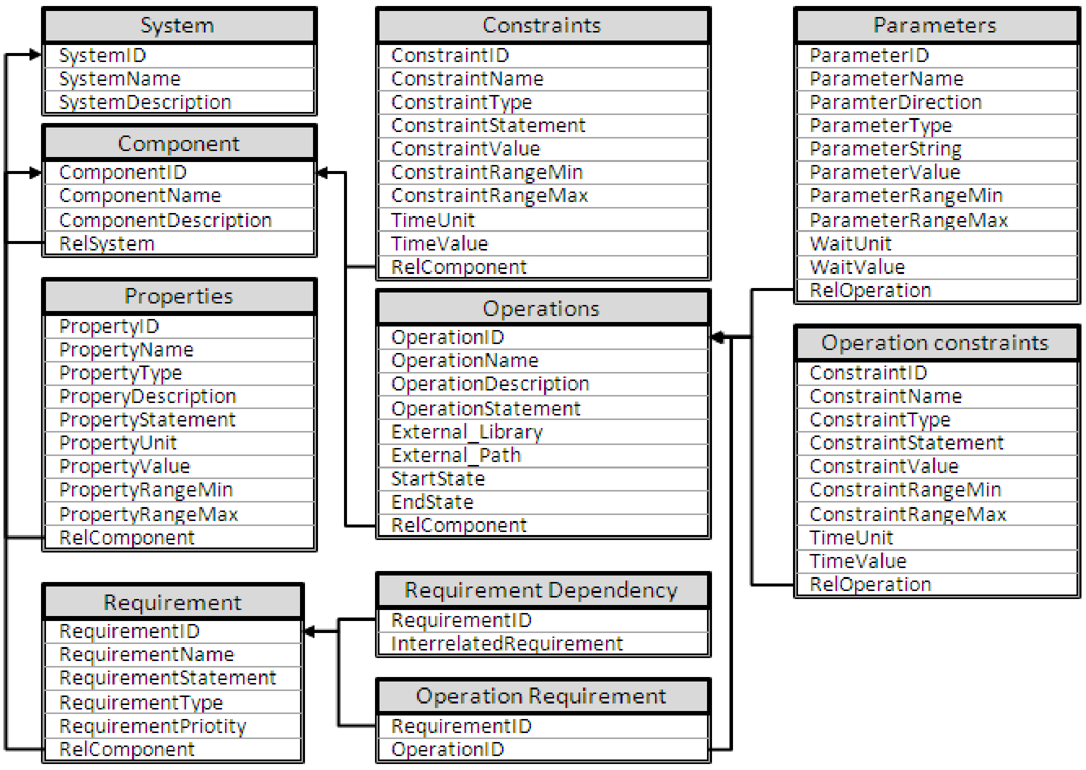


Figure C.4 MAIP-database schema of CAM operations

For non-functional requirements a new table “constraints” on the component level has been created to persist the non-function requirements which are directly linked to the component. A link table between constraints on the component level and requirements was also needed. In addition, a new table has been created in order to manage physically parts hosted by the component. The following tables and attributes have been created:

Table: “constraints”

ConstraintID: A global unique ID identifying this constraint in the MAIP context.
ConstraintName: A descriptive name of this constraint.
ConstraintType: Three options are allowed (‘String’, ‘Real’, ‘Integer’ and ‘Time’)
ConstraintStatement: This is a formal expression of what the operation shall fulfil (performance).
ConstraintValue: If the constraint is not a statement, the value shall be set.
ConstraintRangeMin: The minimum allowed value of this constraint if defined.
ConstraintRangeMax: The maximum allowed value of this constraint if defined.
TimeUnit: If the constraint type is “time”, the time unit shall be defined. The following options are allowed:

- Frame
- Millisecond
- Second.

TimeValue: If the constraint type is “time”, the time value shall be set.
RelComponent: The link of this constraint to the component to which it is allocated.

To establish a link between the constraint and the related requirement, a database table “constraint-requirement” will be defined to persist this in the MAIP context. This table consists of two attributes:

- RequirementID: A global unique ID identifying this requirement in the MAIP context.
- ConstraintID: A global unique ID identifying this constraint in the MAIP context.

Table: "parts"

PartID: A global unique ID identifying this part in the MAIP context.
PartName: A descriptive name of this part.
PartDescription: A description of this part.
RelComponent: The link of this part to the component to which it is allocated.

The result of the last requirements extension of the CAM is depicted in Figure C.5 which represents the component abstraction model of the system components. For a specific operation which couldn't be modelled in this way (less than 10% of the operations), the operation table has been extended with the attribute "**ExternalLibrary**". The external library has been used to additionally integrate external implemented functions which cannot be formally integrated in the database schema. This external library can also optionally use the parameters defined in the database.

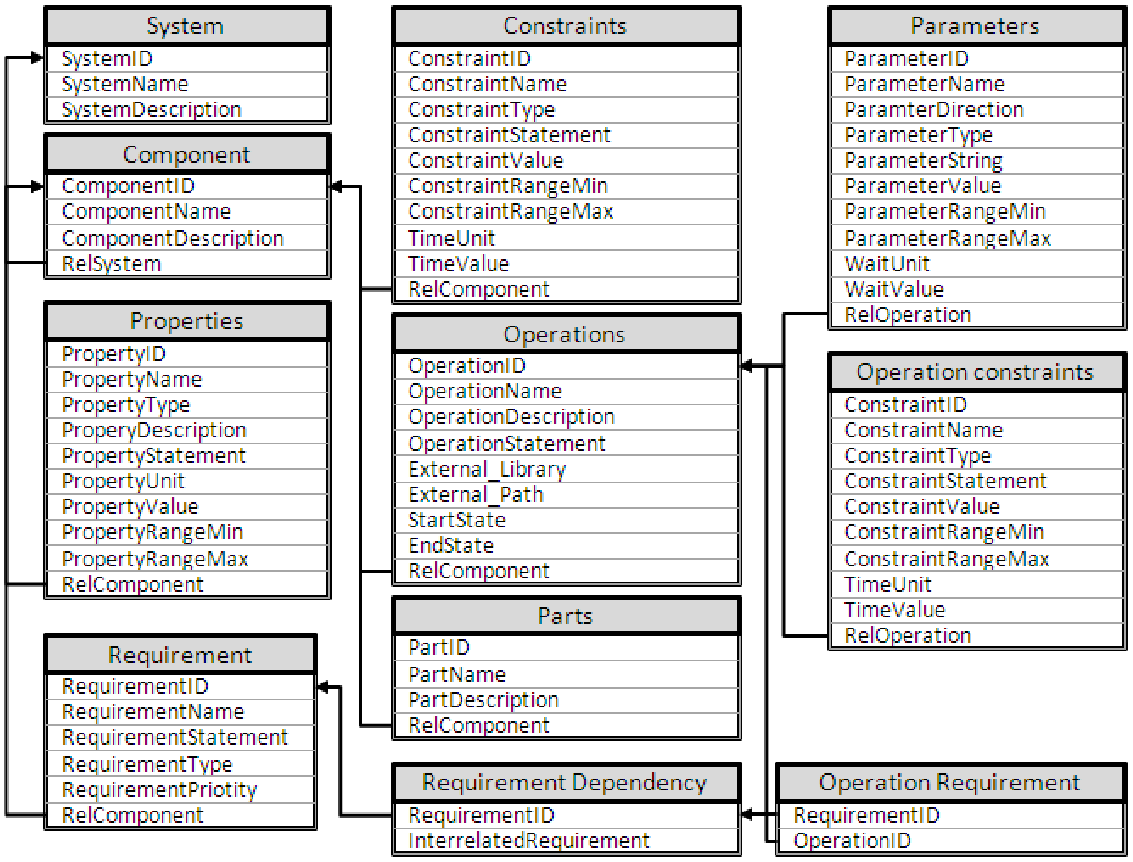


Figure C.5 MAIP-database schema of CAM constraints

Two additional entities were needed to model the state and physical interfaces of the component abstraction model. The state entities have been defined as a set of interface parameter, and state transition has been defined as a specific operation. We extend the operation entity "operations" by two attributes:

StartState: A link to the component start before the operation is starting to be performed.
EndState: A link to the component start after the operation has been performed.

Note: If StartState is equal to EndState, the operation has no state transition operation.

For the representation of state in the CAM model three tables with their respective attributes have been added to the database schema:

“system_state”:

SystemStateID: A global unique ID identifying this system state in the MAIP context.
SystemStateName: A descriptive name of this state.
SystemStateDescription: A description of this state.
RelSystem: The link of this state to this system to which it is allocated.

“component_state”:

ComponentStateID: A global unique ID identifying this component state in the MAIP context.
ComponentStateName: A descriptive name of this state.
ComponentStateDescription: A description of this state.
RelComponent: The link of this state to tje component to which it is allocated.

“state_validity_matrix”

ComponentStateID: Component state ID.
SystemStateID: System state ID representing the global state of the system in which the component state is valid.

For the physical interfaces the “properties” entities have been extended to represent the interface specifications which shall be fulfilled by the component. The new attributes were:

PropertyType: A specification of the interface type. Valid options are:

- discrete_input
- discrete_output
- analogue_input
- analogue _output
- can_input (Controller Area Network as an example for bus system)
- can_output (Controller Area Network as an example for bus system)
- general (for non-standard interfaces).

PropertyUnit: A specification of the interface unit. Valid options are:

- Ampere
- Voltage
- Ohm.

In Figure C.6 the CAM database schema is depicted which has been used as the CAM model for the modelling of the MAIP:

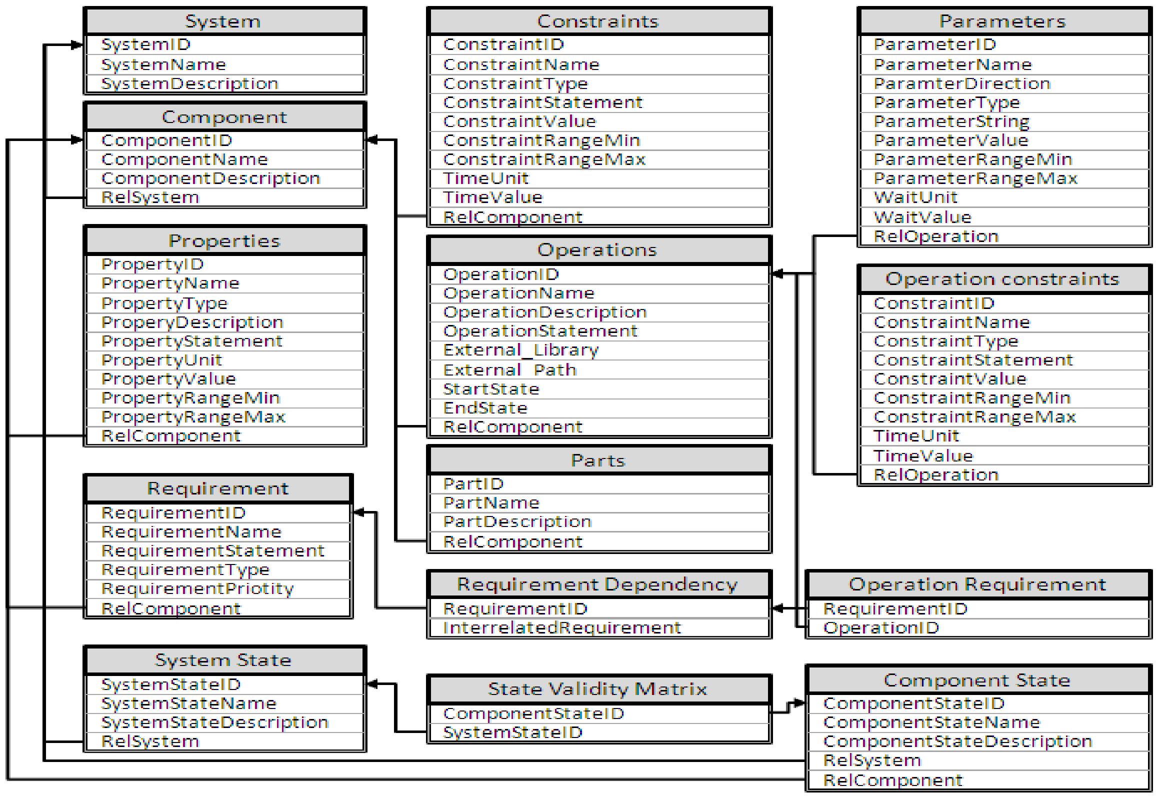


Figure C.6 MAIP-database schema of CAM

CAM- operation definition file

The CAM operation definition file, which represents the operation, is specified by describing information and format and depicted in Figure C.7. A simplified example for an operation definition is depicted in Figure C.8.

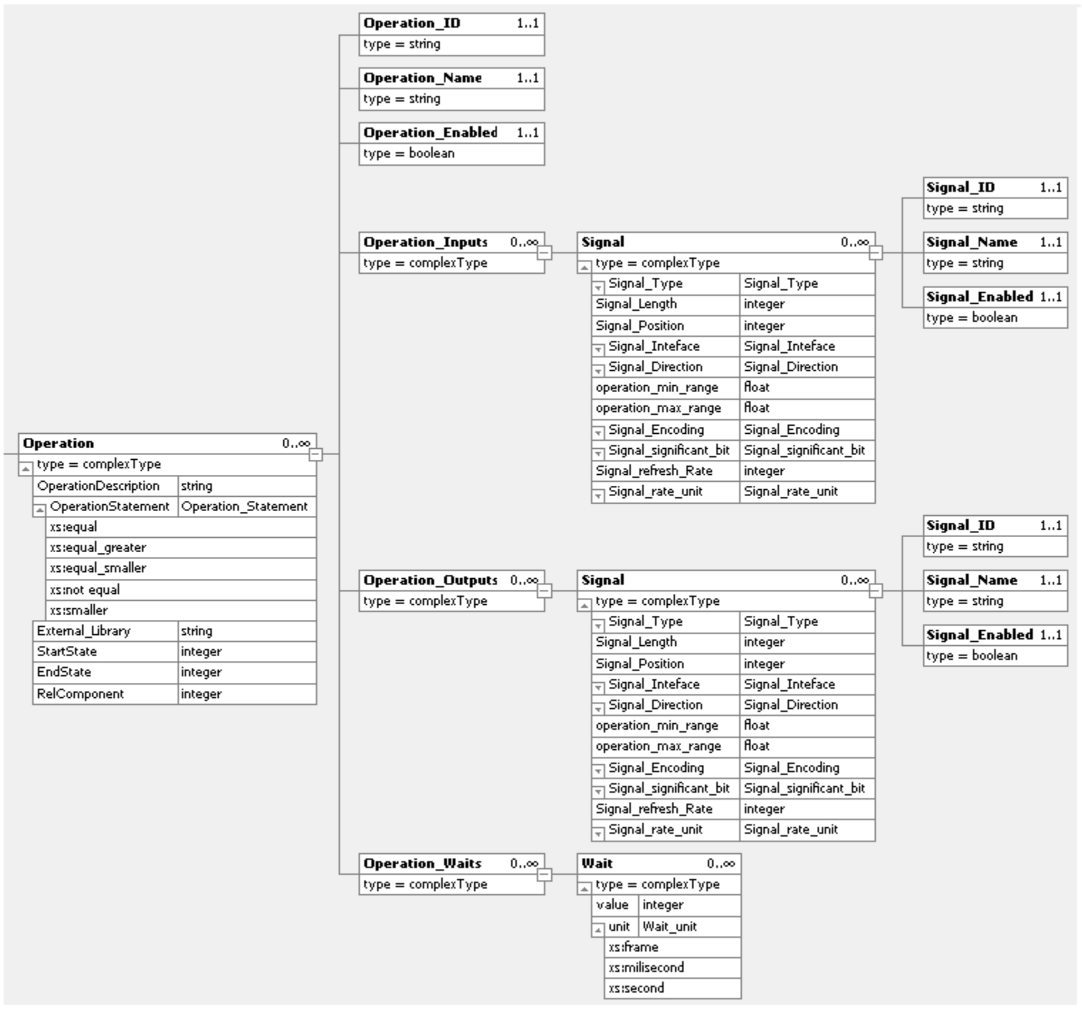
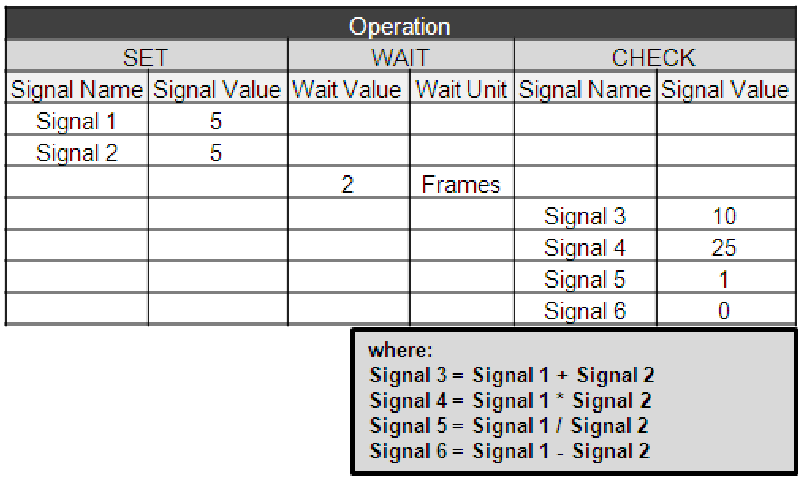


Figure C.7 CAM operation definition file



Operation					
SET		WAIT		CHECK	
Signal Name	Signal Value	Wait Value	Wait Unit	Signal Name	Signal Value
Signal 1	5				
Signal 2	5				
		2	Frames		
				Signal 3	10
				Signal 4	25
				Signal 5	1
				Signal 6	0

where:
Signal 3 = Signal 1 + Signal 2
Signal 4 = Signal 1 * Signal 2
Signal 5 = Signal 1 / Signal 2
Signal 6 = Signal 1 - Signal 2

Figure C.8 Simplified CAM operation definition

CAM-FunReMo-Commands

GET-COMPONENT-OPERATIONS

It will be used to get list of operations (Name and ID) which are hosted by a certain system component. The Component-ID or Component-Name can be used as parameter for this command.

Signature of the GET-COMPONENT-OPERATIONS

<SYNOPSIS>

- **string[][] CAM-FunReMo-GetComponentOperations ("Component-Name")**
- **string[][] CAM-FunReMo-GetComponentOperations ("Component-ID")**

<DESCRIPTION>

GetComponentOperations gets a list of the system component operations hosted by a certain system component.

<RETURNS>

On success, the list system component operations (2 dimensional string array)
On error, 0 is returned

GET-OPERATION-SPECIFICATION

It will be used to get a specification of a certain operation (identified by Name or ID). The Operation-ID or Operation-Name can be used as parameter for this command.

Signature of the GET-OPERATION-SPECIFICATION

<SYNOPSIS>

- **object CAM-FunReMo-GetOperationSpecification ("Operation-Name")**
- **object CAM-FunReMo-GetOperationSpecification ("Operation-ID")**

<DESCRIPTION>

GetOperationSpecification gets a specification of a certain operation hosted by a certain system component.

<RETURNS>

On success, the system component operation specification (as a XML-file).
On error, 0 is returned.

GET-OPERATION-SIGNALS

It will be used to get a list of operation signals (identified by operation Name or ID). The Operation-ID or Operation-Name can be used as parameter for this command.

Signature of the GET-OPERATION-SIGNALS

<SYNOPSIS>

- **string[][] CAM-FunReMo-GetOperationSignals ("Operation-Name")**
- **string[][] CAM-FunReMo-GetOperationSignals("Operation-ID")**

<DESCRIPTION>

GetOperationSignals gets a list of the operation signals hosted by a certain system component.

<RETURNS>

On success, the list operation signals (2 dimensional string array).
On error, 0 is returned.

CAM- interface definition file

The CAM interface definition file, which represents the interface is specified by describing information and format and depicted in Figure C.9.

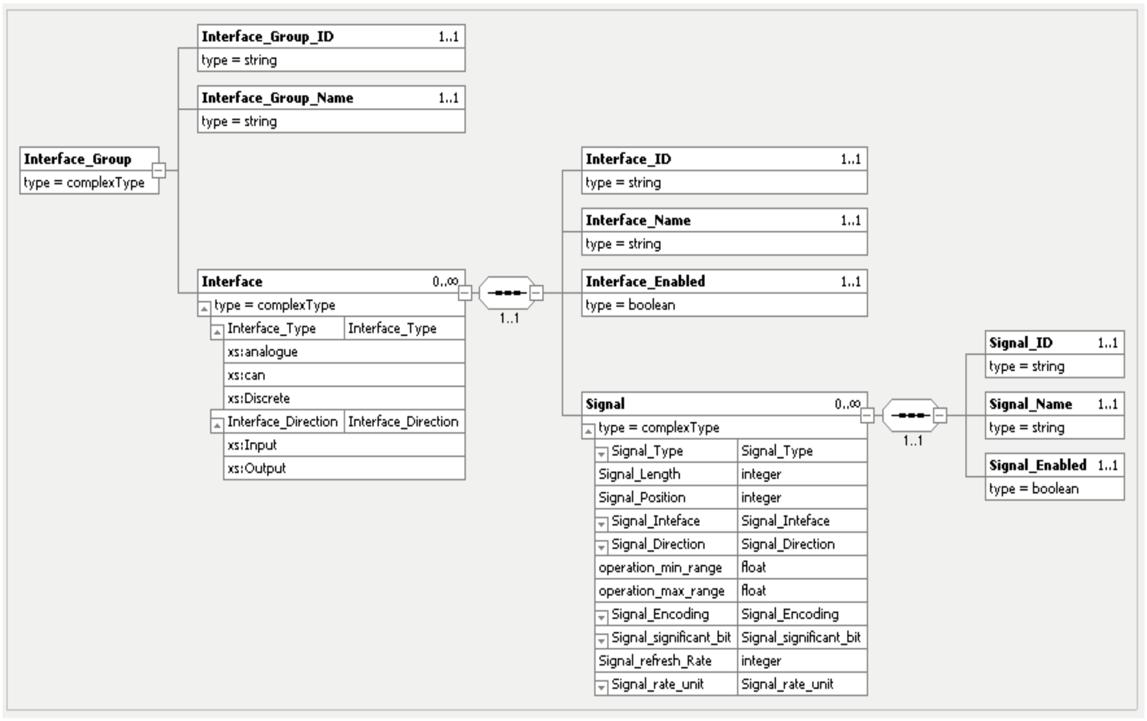


Figure C.9 CAM interface definition file

CAM-IntReMo-Commands

GET-COMPONENT-SIGNALS

It will be used to get list of signals (Name, ID, attributes) which are allocated to a certain system component. The Component-ID or Component-Name can be used as parameter for this command.

Signature of the GET-COMPONENT-SIGNALS

<SYNOPSIS>

- **string[][] CAM-IntReMo-GetComponentSignals (“Component-Name”)**
- **string[][] CAM-IntReMo- GetComponentSignals (“Component-ID”)**

<DESCRIPTION>

GetComponentSignals gets a list of the system component signals allocated to a certain system component.

<RETURNS>

On success, the list system component signals (2 dimensional string array)
On error, 0 is returned

GET-INTERFACE-SPECIFICATION

It will be used to get an interface specification of a certain system component (identified by Name or ID). The Component-ID or Component -Name can be used as parameter for this command.

Signature of the GET-INTERFACE-SPECIFICATION

<SYNOPSIS>

- **object CAM-IntReMo-GetInterfaceSpecification ("Operation-Name")**
- **object CAM-intReMo- GetInterfaceSpecification ("Operation-ID")**

<DESCRIPTION>

GetInterfaceSpecification gets a specification of an interface of a certain system component including signal group and signal attributes.

<RETURNS>

On success, the system component interface specification (as a XML-file).
On error, 0 is returned.

CAM- state definition file

The CAM state definition file, which represents the state, is specified by describing information and format and depicted in Figure C.10.

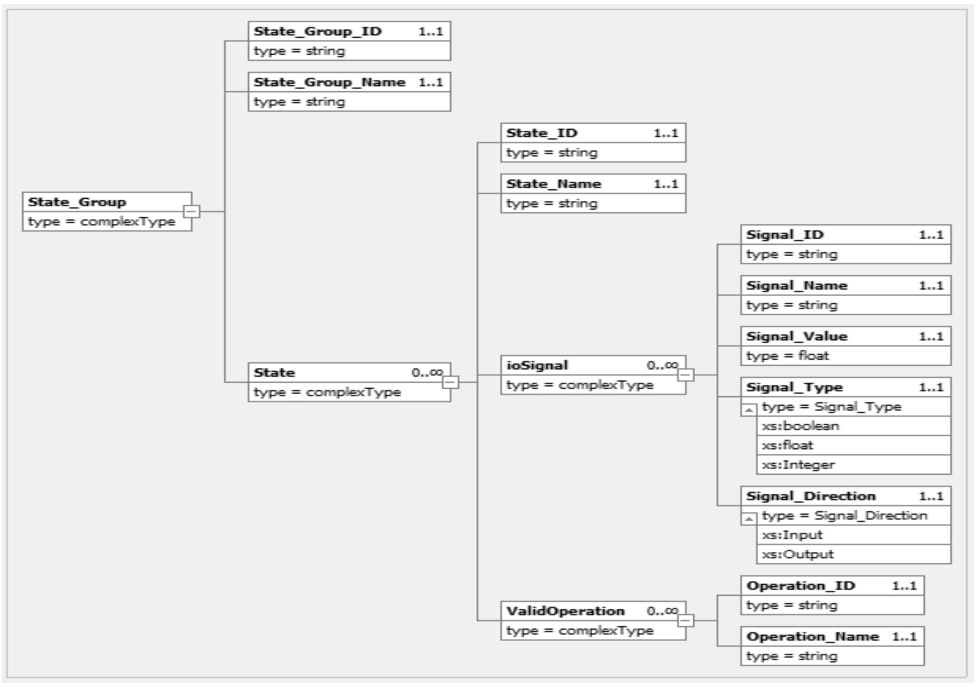


Figure C.10 CAM state definition file

CAM-StaReMo-Commands

GET-COMPONENT-STATE

It will be used to get list of states (Name, ID) which are allocated to a certain system component. The Component-ID or Component-Name can be used as parameter for this command.

Signature of the GET-COMPONENT-STATES

<SYNOPSIS>

- **string[][] CAM-StaReMo-GetComponentStates (“Component-Name”)**
- **string[][] CAM- StaReMo- GetComponentStates (“Component-ID”)**

<DESCRIPTION>

GetComponentStates gets a list of the system component states allocated to a certain system component.

<RETURNS>

On success, the list system component states (2 dimensional string array)
On error, 0 is returned

GET-STATE-SPECIFICATION

It will be used to get an state specification of a certain system and it components (identified by Name or ID). The System-ID or System-Name can be used as parameter for this command.

Signature of the GET-STATE-SPECIFICATION

<SYNOPSIS>

- **object CAM- StaReMo-GetStateSpecification (“System-Name”)**
- **object CAM- StaReMo- GetStateSpecification (“System-ID”)**

<DESCRIPTION>

GetStateSpecification gets a specification of the state of a certain system and its components including the valid operation.

<RETURNS>

On success, the system state specification (as a XML-file).
On error, 0 is returned.

CAM- performance definition file

The CAM performance definition file, which represents the performance, is specified by describing information and format and depicted in Figure C.11

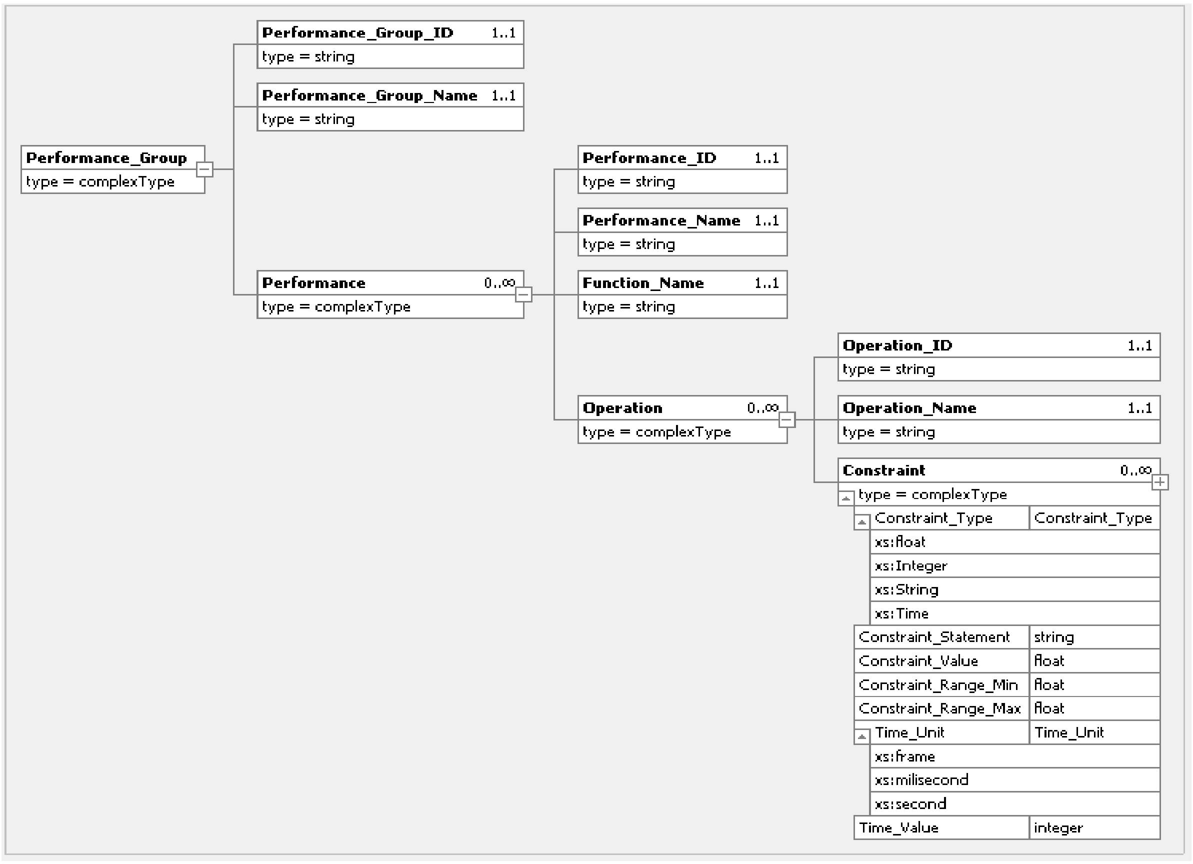


Figure C.11 CAM performance definition file

CAM-PerReMo-Commands

GET-OPERATION-CONSTRAINTS

It will be used to get list of operation constraints (all constraints attributes) which are allocated to a certain system component operation. The Operation-ID or Operation-Name can be used as parameter for this command.

Signature of the GET-OPERATION-CONSTRAINTS

<SYNOPSIS>

- **string[7][] CAM-PerReMo-GetOperationConstraints (“Operation-Name”)**
- **string[7][] CAM- PerReMo- GetOperationConstraints (“Operation -ID”)**

<DESCRIPTION>

GetOperationConstraints gets a list of the operation constraints allocated to a certain system component operation.

<RETURNS>

On success, the list system component operation constraints (7 dimensional string array)
On error, 0 is returned

GET-PERFORMANCE-SPECIFICATION

It will be used to get performance requirements specification of a certain function and its involved system components (identified by Function-Name or Function-ID). The Function-Name or Function-ID can be used as parameter for this command.

Signature of the GET-PERFORMANCE-SPECIFICATION

<SYNOPSIS>

- **object CAM- PerReMo-GetPerformanceSpecification ("Function-Name")**
- **object CAM- PerReMo- GetPerformanceSpecification ("Function-ID")**

<DESCRIPTION>

GetPerformanceSpecification gets a specification of the function performance of a certain function and its involved systems.

<RETURNS>

On success, the function performance specification (as a XML-file).
On error, 0 is returned.

Appendix D

CTASM Syntax

Operation syntax

To specify an elementary function of the system component which represents a relationship between values (usually input and outputs parameter) with respect to the time constraints, validity and computation following XML-Schema based annotation has been defined:

Operation

```
<xs:complexType name="operation">
   <xs:annotation>
     <xs:documentation>an operation implemented by a system component
                component</xs:documentation>
   </xs:annotation>
  <xs:attribute name="Code" type="xs:string" use="required" />
   <xs:sequence>
     <xs:element name="operationparameter" type="signal" minOccurs="0" />
     <xs:element name="performancerequirements"
               type="performancerequirements" minOccurs="0" />
     <xs:element name="operationrequirements"
               type="abta:operationrequirements" minOccurs="0" />
     <xs:element name="states" type=" valid_states " minOccurs="0" />
     <xs:element name="transitions" type=" valid_transitions " minOccurs="0" />
   </xs:sequence>
 </xs:complexType>
```

Code: An unique ID of the operation that is used for the coding of the states an transitions within the CTASM. The coding of the state and transition within the CTASM is an additional extension to TASM to specify the validity and visibility of the operations. Software agent can use this coding to perform operation tests during a certain state or transition. The coding of states and transitions allows software agent to easily link the operation to the right state or transition during the knowledge gaining process when wrongly no links are defined in the CTASM. In the same way a re-linking capability can be performed by software agent.

Operation Parameter

Specification of input and output parameters (system component signals).

```
<xs:complexType name="signal">
   <xs:annotation>
     <xs:documentation>a signal (input or output)</xs:documentation>
   </xs:annotation>
   <xs:attribute name="name" type="xs:string" use="required" />
   <xs:attribute name="type" type="datatype" use="required" />
   <xs:attribute name="direction" type="signal_direction" use="required" />
   <xs:attribute name="size" type="xs:int" use="optional" />
   <xs:attribute name="min" type="xs:string" use="optional" />
   <xs:attribute name="max" type="xs:string" use="optional" />
   <xs:attribute name="default" type="xs:string" use="optional" />
   <xs:attribute name="value" type="xs:string" use="optional" />
  </xs:complexType>
```

<u>*Name*</u>: unique name of the operation signal.
<u>*Type*</u>: following data type are allowed.

```
<xs:simpleType name="signal_datatype">
  <xs:annotation>
    <xs:documentation>the datatype of a signal</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="int8"/>
    <xs:enumeration value="uint8"/>
    <xs:enumeration value="int16"/>
    <xs:enumeration value="uint16"/>
    <xs:enumeration value="int32"/>
    <xs:enumeration value="uint32"/>
    <xs:enumeration value="real32"/>
    <xs:enumeration value="real64"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="byte"/>
  </xs:restriction>
</xs:simpleType>
```

<u>*Direction*</u>: Operation input, output or both.

```
<xs:simpleType name="signal_direction">
  <xs:annotation>
    <xs:documentation>the direction of a signal</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="input"/>
    <xs:enumeration value="output"/>
    <xs:enumeration value="inout"/>
  </xs:restriction>
</xs:simpleType>
```

<u>*Signal min and max values*</u>**:** the minimum or maximum allowed signal value.
<u>*Default*</u>: the default value of the signal if not assigned.
<u>*Value*</u>: the value of the signal at a certain point of time (time of observation)

<u>***Performance Requirements***</u>

Specification of the allowed maximum time required by the system component to perform the operation.

```
<xs:complexType name="performancerequirement">
  <xs:annotation>
    <xs:documentation>a performance requirement for an operation</xs:documentation>
  </xs:annotation>
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="required_time" type="xs:int" use="required" />
  <xs:attribute name="unit" type="timeunit" use="required" />
</xs:complexType>
```

<u>*ID*</u>: Unique ID of the performance requirement operation that is used for the identifying this requirements.
<u>*Required Time*</u>: the allowed maximum time to perform this operation.
<u>*Time Unit*</u>: units of time

```
    <xs:simpleType name="timeunit">
      <xs:annotation>
        <xs:documentation>units of time </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:enumeration value="frames"/>
        <xs:enumeration value="ms"/>
        <xs:enumeration value="s"/>
      </xs:restriction>
    </xs:simpleType>.
```

<u>Operation Requirements</u>

Specification of the function (relationship between output and input signals). For the operation requirements logical, mathematical expression as well as external library references (external coded function) can be used.

```
<xs:complexType name="operationrequirement">
    <xs:annotation>
      <xs:documentation>a requirement for an operation</xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="constraint" type="constraint"
                         minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
 </xs:complexType>
```

<u>Constraint</u>: the specification of the function with the following syntax

```
signal="signal name for example s3";
compare="compare enumeration for example gt for greater than ";
expression="for example s1 + s2”.
```

where validity is an additional expression specifying the signal pattern in which this requirement is valid (for example s3 = s4)

```
<xs:complexType name="constraint">
    <xs:annotation>
      <xs:documentation>a constraint to check for an operation
                 requirement</xs:documentation>
    </xs:annotation>
    <xs:attribute name="signal" type="xs:string" use="optional" />
    <xs:attribute name="compare" type="abta:compare" use="optional" />
    <xs:attribute name="expression" type="xs:string" use="required" />
    <xs:attribute name="validity" type="xs:string" use="optional" />
  </xs:complexType>

<xs:simpleType name="compare">
    <xs:annotation>
      <xs:documentation>comparers for a check definition</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:enumeration value="eq"/>
      <xs:enumeration value="lt"/>
      <xs:enumeration value="gt"/>
      <xs:enumeration value="lte"/>
      <xs:enumeration value="gte"/>
    </xs:restriction>
  </xs:simpleType>
```

ID: Unique ID of the operation requirement operation that is used for the identifying this requirements.

Valid states

A list of states references within the CTASM in which the operation can be performed.

```
<xs:complexType name="valid_states">
    <xs:annotation>
      <xs:documentation>a list of valid state references </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="states" type="IDREF" minOccurs="0" />
    </xs:sequence>
 </xs:complexType>
```

Valid transitions

A list of transition references within the CTASM in which the operation can be performed.

```
<xs:complexType name="valid_transitions">
    <xs:annotation>
      <xs:documentation>a list of valid transition references </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="transition" type="IDREF" minOccurs="0" />
    </xs:sequence>
 </xs:complexType>
```

If the two lists of valid states and valid transitions are empty the operation is globally valid within this CTASM as long as not revised by the knowledge gaining process performed by software agent during testing .

State syntax

```
<xs:complexType name="state">
    <xs:annotation>
      <xs:documentation>a signal pattern based state</xs:documentation>
    </xs:annotation>
    <xs:sequence>
<xs:element name="constraint" type="abta:constraint" minOccurs="0"
                                   maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
 </xs:complexType>
```

See Section 3.6.1 for the graphical representation.

Transition syntax

```
<xs:complexType name="transition">
    <xs:annotation>
      <xs:documentation>a single state transition</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="state" type="state" minOccurs="0" />
      <xs:element name="performancerequirement" type="idref" minOccurs="0" />
      <xs:element name="state_from" type="idref" minOccurs="1"
                  maxOccurs="unbounded" />
      <xs:element name="state_to" type="idref" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
    <xs:attribute name="name" type="xs:string" use="required" />
    <xs:attribute name="timeout" type="xs:int" use="optional" />
    <xs:attribute name="timeout_unit" type="timeunit" use="optional" />
 </xs:complexType>
```

See Section 3.6.1 for the graphical representation

LITERATURE

Literature:

[Alur 1998]
R. Alur: Timed automata. In Summer School on Verification of Digital and Hybrid Systems. NATO-ASI (1998).

[Althoff 2011]
Klaus-Dieter Althoff: Distributed learning systems, , intelligent information system IIS, university of Hildesheim (2011).

[Bender 2009]
Bender: Requirements Based Testing Process Overview, Cardinale Lane Queensbury (2009).

[Breskovic 2011]
I. Breskovic, M. Maurer V. Emeakaroha and Altmann: Towards autonomic market management in cloud computing infrastructures. In International Conference on Cloud Computing and Services Scienc (2011).

[Bellifemine 2011]
Bellifemine: An agent-based signal processing in-node environment for real-time human activity monitoring based on wireless body sensor networks (2011)

[Basili 1992]
Victor R. Basili: The Experimental Paradigm in Software Engineering, Experimental Software Engineering Issues: Critical Assessment and Future Directions (Pages 3–12, 1992).

[Bonabeau 1999]
E. Bonabeau: Editor’s introduction: Stigmergy. Artificial Life (Page 95–96, 1999).

[Bordini 2011]
Rafael H. Bordini. Agent and Multi-Agent Software Engineering: Modelling, Programming, and Verification (2011).

[Bonini 2013]
Luca Bonini: Neurophysiological bases underlying the organization of intentional actions and the understanding of others’ intention (2013),

[Brooks 1991]
R. A. Brooks :Intelligence without representation. Artificial Intelligence (Page 139-159, 1991).

[Clarke 2000]
E. M. Clarke, O. Grumberg, and D. A. Peled: Model Checking, volume Second Printing. The MIT Press (2000).

[Davis 1988]
R. Davis and R. G. Smith: Negotiation as a metaphor for distributed problem solving. Distributed Artificial Intelligence (Pages 333–356, January 1988).

[D’Inverno 1997]
D'Inverno and M. Wooldridge: A Formal Specification of dMARS”, Tech. Rep. 72, Australian Artificial Intelligence Institute, Melbourne, Australia (1997).

[Durfee 1989]
E. H. Durfee, V. R. Lesser, and D. D. Corkill: Trends in cooperative distributed problem solving. IEEE Transactions on Knowledge and Data Engineering (Pages 63–83, 1989).

[Durfee 2001]
E. H. Durfee: Distributed problem solving and planning. In J. G. Carbonell and J. Siekmann, editors, Multi-agent systems and applications (Pages 118–149, 2001).

[F. A. Board 2000]
F. A. Board. FIPA Communicative Act Library Specification: Technical report, Foundation for Intelligent Physical Agents (2000).

[Ferber 1999]
J. Ferber: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (February 1999).

[Foster 2004]
I. Foster and C. Kesselman: The grid, blueprint for a new computing infrastructure. Morgan Kaufmann (2004).

[Mihalis Giannakis 2011]
Giannakis, M. and Louis, M. "A multi-agent based framework for supply chain risk management" Journal of Purchasing and Supply Management 17 (2011): 23-31.

[Georgeff 2009]
Georgeff, M. "The gap between software engineering and multi-agent systems: bridging the divide." International Journal of Agent-Oriented Software Engineering 3 (2009): 391-196.

[Grieskamp 2002]
Grieskamp, W., Gurevich, Y., Schulte, W. and Veanes, M. "Generating finite state machines from abstract state machines." in ISSTA '02 Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis. Pages 112-122. New York: ACM.

[Grosch 2011]
V. Grosch: Requirements traceability within model-based testing: Applying path fragments and temporal logic. In Seppo Virtanen, editor, International Journal of Embedded and Real-Time Communication Systems (Pages 1 - 21, 2011).

[Heiko Rölke 2004]
Heiko Rölke: Modelling of agents and multi-agent systems, Basic and applications(2004).

[Huhns 2012]
Huhns: Agent-based organisational governance of services, Multiagent and Grid Systems (2012).

[Kephart 2000]
J. O. Kephart, J. E. Hanson, and A. R. Greenwald: Dynamic pricing by software agents. Comput. Netw (Pages 731–752, 2000).

[Kirn 2006]
S. Kirn: Flexibility of multiagent systems, Multiagent Engineering, International Handbooks on Information Systems (Pages 53–70, 2006).

[Kirn 2006-2]
S. Kirn and T. Bieser: Simulationswerkzeug zur Analyse von Wertschöpfungsnetzwerken. In J. Banzhaf and H. Kuhnle, editors, Entwicklungsperspektiven der Unternehmensführung und ihrer Berichterstattung (Pages 121–136. Gabler, Wiesbaden, 2006).

[Kirn 2006-Ref3]
S. Kirn, O. Herzog and P. Lockemann: Multiagent Engineering – Theory and Applications in Enterprises. Springer Berlin Heidelberg (March 2006).

[Krupansky 2008]

Krupansky, J. W: Foundations of Software Agent Technology, Agtivity: Advancing the Science of Software Agent Technology (2008).
[Leal 2001]
F. Leal and J. Rodriguez. Message: Methodology for engineering systems of software agents. Technical report, Telecom Italia Lab (2001).

[Lottor 2010]
W. Lotter: Einfach mehr. brand I (2010).

[Luck 2005]
M. Luck, P. McBurney, O. Shehory, and S. Willmott: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005).

[Lundqvist 2008]
Ouimet, M. and Lundqvist, K. "The Timed Abstract State Machine Language: abstract state machines for real-time system engineering." Journal of Universal Computer Science 14(12): 2007-2033 (2008)

[Marrow 2001]
P. Marrow: Agents in decentralised information ecosystems: the diet approach. In Proceedings of the Artificial Intelligence and Simulation Behaviour Convention (Pages 109–117, March 2001).

[Purvis 1996]
Purvis : Agent Modelling with Petri Nets. The Information Science Discussion Paper Series (March 1996).

[M. E. Nissen 2002]
M. E. Nissen: An extended model of knowledge-flow dynamics, Communications of the Association for Information Systems (Pages 251– 266, 2002).

[Miao Yu 2012]
Miao Yu: The Extension of KQML Primitive in Large Database Operation (December 2012)

[Müller 1997]
J. P. Müller: The Design of Intelligent Agents: A Layered Approach. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997).

[Muneendra 2012]
Muneendra: A Novel Service Oriented Model for Query Identification and Solution Development using Semantic Web and Multi Agent System (2012)

[Ouimet 2007]
M. Ouimet, K. Lundqvist: The Timed Abstract State Machine Language: An Executable Specification Language for Reactive Real-Time Systems (2007)

[Padgham 2002]
L. Padgham and M. Winikoff. Prometheus: a methodology for developing intelligent agents. In AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems (Pages 37–38, 2002).

[Resnick 1994]
M. Resnick: Turtles, termites, and traffic jams: explorations in massively parallel microworlds. MIT Press, Cambridge, MA, USA (1994).

[Ricci 2011]
Ricci, A., Piunti, M. and Viroli, M. "Environment programming in multi-agent systems: an artifact-based perspective." Autonomous Agents and Multi-Agent Systems 23 (2011): 158-192.

[Rosenzweig 2000]
Y. Gurevich, D. Rosenzweig: Partially Ordered Runs: A Case Study. In: Abstract State Machines (2000).

[Sabas 2002]
A. Sabas, S. Delisle and M. Badri : A Comparative Analysis of Multiagent System Development Methodologies, Towards a Unified Approach (2002).

[Sandholm 1999]
T.W. Sandholm: Distributed rational decision making. In Multiagent systems: a modern approach to distributed artificial intelligence (Pages 201–258, 1999).

[Searle 1969]
J. R. Searle: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, (1969).

[Smith 1988]
R. G. Smith: The contract net protocol: high-level communication and control in a distributed problem solver. In Distributed Artificial Intelligence (Pages 357–366, 1988).

[Timm 2003]
I. J. Timm: Dynamisches Konfliktmanagement als Verhaltenssteuerung Intelligenter Agenten. PhD thesis, Fachbereich Mathematik und Informatik, Universität Bremen: Bremen (2003).

[Utting 2006]
M. Utting, B. Legeard, and A. Pretschner: A taxonomy of model-based testing. Working Paper Series, Hamilton, New Zealand (2006).

[Weiss 2001]
G. Weiss: Mulitagent Systems - A Modern Appoach to Distributed Artificial Intelligence. The MIT Press Cambridge, Massachusetts, London England (2001).

[Wilensky 1999]
U. Wilensky: Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL (1999).

[Wooldridge 2000]
M. Wooldridge, N. R. Jennings, and Kinny: The Gaia methodology for agent-oriented analysis and design, Autonomous Agents and Multi-Agent Systems (Pages 285-312, 2000).

[Wooldridge 1995]
M. Wooldridge and N. R. Jennings: Intelligent agents: Theory and practice. The Knowledge Engineering Review (Pages 115–152, 1995).

[Wooldridge 1999]
M. Wooldridge and N. R. Jennings: Software engineering with agents: Pitfalls and pratfalls. IEEE Internet Computing (Pages 20–27, 1999).

[Wooldridge 2002]
M. Wooldridge: An Introduction to Multi-Agent Systems. Wiley, West Sussex, UK, March(2002).

[Xudong 2012]
Xudong Luo, Chunyan Miao: A knowledge engineering methodology for negotiation agent development, automated negotiation; agents; knowledge engineering; software engineering (MAR 2012)

[Zambonelli, Jennings, and Wooldridge 2003]

F. Zambonelli, N. R. Jennings M. Wooldridge: Developing multi-agent systems: The Gaia methodology, ACM Transactions on Software Engineering and Methodology (Pages 317-370, 2003.