KTH Computer Science
and Communication

A Framework for Component Based Modelling and Simulation using BOMs and Semantic Web Technology

FARSHAD MORADI

Doctoral Thesis in Teleinformatics
Stockholm, Sweden 2008

ISBN 978-91-7178-957-0
TRITA-ICT/ECS AVH 08:05
ISSN 1653-6363
ISRN KTH/ICT/ECS AVH-08/05—SE

Akademisk Avhandling som med tillstånd av Kungliga Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen fredagen den 23 maj 2008 kl 13.00 i Sal D, Forum, Isafjordsgatan 39, Kista.

Tryck: Universitetservice US AB

توانا بود هر که دانا بود
ز دانش دل پیر برنا بود

To be wise is to be strong
Search for knowledge keeps the heart young

- Ferdowsi,
Persian poet (935–1020)

Where flowers bloom, so does hope

- Lady Bird Johnson

Abstract

Modelling and Simulation (M&S) is a multi-disciplinary field that is widely used in various domains. It provides a means to study complex systems before actual physical prototyping and helps lowering, amongst others, manufacturing and training costs. However, as M&S gains more popularity, the demand on reducing time and resource costs associated with development and validation of simulation models has also increased. Composing simulation models of reusable and validated simulation components is one approach for addressing the above demand. This approach, which is still an open research issue in M&S, requires a composition process that is able to support a modeller with discovery and identification of components as well as giving feedback on feasibility of a composition.

Combining components in order to build new simulations raise the non-trivial issue of composability. Composability has been defined as the capability to select and assemble reusable simulation components in various combinations into simulation systems to meet user requirements. There are three main types of composability, syntactic, semantic and pragmatic. Syntactic composability is concerned with the compatibility of implementation details, such as parameter passing mechanisms, external data accesses, and timing mechanisms. It is the question of whether a set of components can be combined. Semantic composability, on the other hand, is concerned with the validity of the composition, and whether the composed simulation is meaningful. Pragmatic composability is yet another type which is concerned with the context of the simulation, and whether the composed simulation meets the intended purpose of the modeller. Of these three types syntactic composability is easiest to accomplish and some significant progresses on this issue have been reported in the literature. Semantic and pragmatic composability are much harder to achieve and has inspired many researchers to conduct both theoretical and experimental research.

The Base Object Model (BOM) is a new concept identified within M&S community as a potential facilitator for providing reusable model components for the rapid construction and modification of simulations. Although BOMs exhibit good capabilities for reuse and composability they lack the required semantic information for semantic matching and composition. There is little support for defining concepts and terms in order to avoid ambiguity, and there is no method for matching behaviour of conceptual models (i.e., state machines of the components), which is required for reasoning about the validity of BOM compositions.

In this work we have developed a framework for component-based model development that supports both syntactic and semantic composability of simulation models by extending the BOM concept using ontologies, Semantic Web and Web Services technologies, and developing a rule-based method for reasoning about BOM compositions. The issue of pragmatic composability has not been the focus of this work, and it has only been partly addressed. The framework utilises intelligent agents to perform discovery and composition of components, according to the modeller needs. It includes a collaborative environment, a semantic distributed repository and an execution environment to support model development and execution process.

The basic assumption of this work is that semantic composability should be achieved at conceptual level. Through precise definition and specification of components' semantic and syntax one can capture the basic requirements for matching and semantically meaningful composition of those components. This requires a common methodology for specification of simulation components. The specification methodology consists of meta-models describing simulation components at different levels. In order to enable automatic matching of meta-models they are formalized and structured using Semantic Web technology in OWL (Web Ontology Language). Hence, the models are based on ontologies to avoid misunderstanding and to provide unambiguous definitions as a basis for reasoning about syntactic and semantic validity of compositions.

Sammanfattning

Modellering och Simulering (M&S) är en multidisciplinär metod som används inom olika domäner. Det erbjuder medel för att studera komplexa system innan dessa har utvecklats och bidrar till att minska bl a utvecklings- och träningskostnader. Utveckling av modeller och simuleringar kan dock innebära initiala kostnader som bör reduceras för att öka metodens användbarhet. Att bygga simuleringar genom sammansättning av återanvändbara och validerade simuleringskomponenter är ett sätt att åstadkomma detta. Detta tillvägagångssätt, som är föremål för mycken aktuell forskning, kräver en sammansättningsprocess som kan stödja en utvecklare med såväl upptäckt och identifiering av lagrade komponenter, som validering av sammansatta simuleringar.

En av de viktigaste frågorna när det gäller komponering av simuleringar är *composability*, som har definierats som förmågan att välja och sätta ihop återanvändbara simuleringskomponenter i olika kombinationer och utveckla simuleringar som möter användarens krav och behov. Det finns huvudsakligen tre olika typer of composability, syntaktisk, semantisk och pragmatisk. Om implementeringsrelaterade detaljer för olika simuleringskomponenter är kompatibla råder syntaktisk composability mellan dessa. Exempel på sådana detaljer är mekanismer för parameterutbyte, dataöverföring och tidshantering. Semantisk composability handlar däremot om meningsfullheten av den sammansatta modellen. Genom semantisk composability kan giltigheten hos den sammansatta modellen säkerställas. Pragmatisk composability berör simuleringens kontext, d v s att den sammansatta simuleringen uppfyller modellerarens syfte. Av dessa ovanstående typer är syntaktisk composability den enklaste att åstadkomma, och där har stora framgångar uppnåtts med många ramverk som kan hantera denna typ av composability. Semantisk och pragmatisk composability är däremot svårare att uppnå och har inspirerat många forskare att genomföra både teoretisk och tillämpad forskning.

Base Object Model (BOM) är ett nytt koncept inom M&S med syftet att erbjuda återanvändbara modellkomponenter för utveckling och modifiering av simuleringar. BOM-konceptet baserar sig på antagandet att en simulerings beståndsdelar kan extraheras och återanvändas som byggblock. Även om BOM visar god förmåga och potential för återanvändning, saknar det den nödvändiga informationen för att garantera semantisk composability. Det finns för lite stöd för att definiera termer och koncept för att undvika tvetydighet, och det saknas metoder för att matcha olika BOM:ars tillståndsdiagram, vilka är nödvändiga för att kunna resonera om den sammansatta modellens meningsfullhet.

I det här arbetet har vi utvecklat ett ramverk för komponentbaserad modellutveckling som stödjer både syntaktisk och semantisk composability av BOM-komponenter genom att bygga ut BOM-konceptet m h a ontologier, Semantic Web- och Web Services-teknologier, samt utveckla en regelbaserad metod för att resonera om de sammansatta modellernas meningsfullhet. Frågan om pragmatisk composability har inte varit i fokus för det här arbetet och bara delvis behandlats. Ramverket använder sig av intelligenta agenter för att genomföra upptäckt och ihopsättning av komponenter, baserat på modellerarens behov och mål. Det inkluderar en kollaborativ miljö, ett semantiskt distribuerat modellbibliotek och en distribuerad exekveringsmiljö för att stödja processen med modellutveckling och -exekvering.

Det grundläggande antagandet i detta arbete är att semantisk composability kan uppnås på konceptuell nivå. Genom exakta beskrivningar och specificering av komponenternas semantik och syntax kan man fånga de mest grundläggande kraven för att semantiskt matcha och sätta ihop dessa komponenter. Detta kräver en gemensam arkitektur för att definiera simuleringskomponenter. Arkitekturen innehåller metamodeller som beskriver dessa komponenter på olika nivåer. För att möjliggöra automatisk matchning av dessa metamodeller är de strukturerade och formaliserade genom att använda Semantic Web-teknologi och OWL (Web Ontology Language). Därmed är metamodellerna baserade på ontologier för att undvika missförstånd och erbjuda otvetydiga beskrivningar, så att de kan konstituera basen för att resonera kring syntaktisk och semantisk composability av komponenter.

Acknowledgements

Yet another journey has come to an end with promises of new beginnings. In all humility I take a look back at how it all started and the experiences that I have gained. Whether it was curiosity, desire to learn, or just proving to myself that I could do it, I set out for a journey, which I was never really sure I would follow to the end. And I would have certainly not done that if it had not been for all the support, encouragement, and love that I have received from so many people, whom I would like to thank from the bottom of my heart.

First of all my mentor and supervisor Prof. Rassul Ayani for all his guidance, patience, help and support through the hard times without which none of this would have been possible. I am forever in your debt.

My dear colleagues within the NetSim project, especially Jenny Ulriksson, Martin Eklöf and Marianela Garcia Lozano, for all the hard work in the project, their support, great fun I had working with them and everything they taught me.

Prof. Axel Lehmann, for taking the time to be my opponent, reviewing the manuscript and for his constructive feedbacks and suggestions.

All my colleagues at FOI, particularly the old "gang" from *Systems Modelling*, Gunnar Holm, Lena Sporre, Birgitta Johansson and Lisbeth Pers, for making my working place a pleasant one, and for all the support I have received through the years. My former bosses Monica Dahlén and Ulla Bergsten, and my new bosses Martin Rantzer and Lars Lindberg for their support and providing me with the opportunity to pursue my studies.

My dear friend Dr. Gary Tan and my other co-authors Shahab Mokari, Hossein Akbari, Peder Nordvaller, Imran Mahmood, and Hu Yu, for great collaboration.

Dr. Wentong Cai, Dr. Gary Tan, and Dr. Yong Meng Teo Dr. Stephen J Turner for hosting my visits at NUS and NTU, and for fruitful collaborations through the years. And also Dr. Stan Jarzabek for his collaboration during the early phases of this project.

My wife, my parents, my siblings and their families, especially my sister Mitra, for all their love, support and encouragement. My sons, Adrian and Kian, who are my great teachers and constantly fill my heart with love, joy and pride.

Finally, I thank life for being good to me and for planting a flower in my heart to help me feel, believe and fight.

Contents

Part I

An Introduction to Component Based Modelling and Simulation

Chapter 1

Introduction

This chapter contains general information about the topic, presents the problem definition, explains what has been done, what the objectives are, and introduces the research issues, background technologies, proposed solution, research activities and the contributions of the thesis.

1.1 Background

Modelling and Simulation (M&S) is a multi-disciplinary field that is widely used in areas such as, engineering, training, education, manufacturing and health care. The basic concepts of M&S are *model* and *simulation*. A model is a representation of something that might or might not exist in the real world, and which might have been developed for the purpose of simulation. Such a model is referred here to as a simulation model. Simulation is the process of running a simulation model over a period of time, by e.g. using computers.

M&S provides possibilities to reduce, amongst others, manufacturing and training costs and a means to study complex systems before actual physical prototyping. Employing simulations early in the design phase of a product life cycle can detect and avoid costly errors in later stages of the development process. Also training of personnel using simulated systems can be done at a fraction of the cost of running exercises involving real systems. Thus, M&S is a valuable tool at the time of economic cut backs and streamlining within and stiff competition between organisations and companies.

However, development of simulation models can be a time and resource consuming process and involves some initial costs. Although, the benefits are many the initial costs may be discouraging to decision makers and project managers. Beside the initial cost there is also the issue of quality and usability of the simulation models. A simulation model development process involves different phases including, requirement specification, conceptual modelling, design, development, test, verification and validation. The process requires involvement of different actors such as modellers, subject matter experts, validation and verifications experts and end users. Handling the issue of quality and usability gets more difficult as simulation models get larger and more complex.

An approach to reduce the costs associated with the process and to improve the usability of simulation models is to compose them through reuse of predefined and already existing, validated simulation model components [141]. Using this method the simulation model is built in a component-based fashion. Component-based approach has been successfully deployed in all engineering disciplines, such as manufacturing, hardware and to some extend in software industry.

The component based methodology will help reducing the costs of development of simulation models and improves the quality and user-worth of those models. It has been tried in software development process through employment of object-oriented methodology and techniques, which has simplified the development of complex software systems. However, component based software development is still an open research issue in both M&S and software engineering [144]. Composing sub-models in order to build new models raise the non-trivial issue of composability. Composability is the capability to select and assemble reusable simulation components in various combinations into simulation systems to meet user requirements [103], [101], [35].

A (composable) simulation model component is a software element of a simulation model with well-defined functionalities and behaviours that conforms to a component model and can be independently deployed. It is subject to third-party composition with or without modification and conforms to a composition model. There are three main types of composability, syntactic, semantic and pragmatic. Syntactic composability is concerned with the compatibility of implementation details, such as parameter passing mechanisms, external data accesses, and timing mechanisms. It is the question of whether a set of components can be combined [185], [16], [149], [29]. Semantic composability is concerned with the validity of the composition [102], and pragmatic composability is concerned with the context of the simulation. There have been some significant achievements in syntactic composability both within software engineering and simulation communities, but semantic and pragmatic composability is a much harder problem [100], [30], [104] and has inspired many researchers to conduct both theoretical and experimental research.

1.2 Motivation

The component-based development approach is fairly new within the M&S community. The aim, as explained earlier, is to improve reuse and interoperability between simulations and simulation components developed by different organizations and stakeholders, and also simplify and leverage collaboration and sharing of those components, in order to lower development time and cost.

Currently, there is a lack of clear understanding of what component based M&S is, how model components could be structured, how they are to be specified and what technologies exist to facilitate syntactic and semantic composition [118]. Current approaches to component-based simulation model development (CBMD) are mainly based on reuse of simulation components i.e. program codes. However, it is not trivial to read, understand and modify computer programs, and even more difficult to comprehend the semantics of these codes.

Moreover, the existing simulation composition environments are mainly developed for specific simulators (simulation framworks) and within specific domains [29], [141], and components have to comply with specific architectures. Hence, their reusability is restricted to architectures they are developed for and can not be considered as generic components that could be used in different compositions. One the main reasons that is given for doing so is that the issue of reuse and interchangeability of components is very hard to tackle and not practically possible if components are expected to be used in different compositions. Especially since these compositions can differ from the purpose and the context for which the components have been originally developed [44].

The High Level Architecture (HLA) standard, which has been promoted by the DoD (Department of Defense) is an attempt to provide a general framework for improving reuse and interoperability of simulation models. The standard has been widely adopted within the Modelling and Simulation community. As this framework has been starting to set, new concepts are being formed to further improve reuse of simulations at a component level, in order to speed up and reduce costs of developing and implementing new simulations. The Base Object Model (BOM) is one such new concept. BOM [158] has been identified within HLA as a potential facilitator for HLA object model construction and providing reusable model components used for the rapid construction and modification of federates and federations. BOMs are structured into four major parts, Model Identification, Conceptual Model, Model Mapping, and HLA Object Model. However, BOM lacks the required semantic information to ensure semantic composability of BOM-based components. There is little support for defining concepts and terms in order to avoid ambiguity, and there is no method for matching BOM conceptual models (state machines). How the BOM concept can be used to improve reuse and interoperability of simulation models is currently a current issue within the M&S community [143], [159], [1].

In this work it will be shown how one can achieve semantic and syntactic composability of simulation models through utilization of meta-models, ontologies, BOMs, and technologies, described in this document, and develop a framework, which can support it.

1.3 Scope of the work

The aim of this work has been developing an environment for component-based model development that supports both syntactic and semantic composability. The issue of pragmatic composability has not been the focus of this work, and it has only been partly addressed. The basic assumption is that semantic composability should be achieved at conceptual level, i. e. model specification level. Through precise definition and specification of components' semantic and syntax one can capture the basic requirements for matching and semantically meaningful composition of those components. This requires a common architecture for specification of simulation models and components. The specification framework contains meta-models describing simulation components at different levels. In order to enable automatic matching of meta-models they should be formalized and structured in standard languages. They should also be based on ontologies to avoid misunderstanding and provide unambiguous definitions of those models. Consequently, the meta-models should also be constructed and structured such that it is possible to automatically generate executable components and simulations. The code generation process could provide the modeller the option of choosing different simulation architectures as the final product.

Besides the specification framework, the environment also includes collaborative support, a model repository to store and manage the components, and an execution environment to facilitate reuse.

The scope of the work can be summarized as:

1. Design of a general framework for CBMD, which supports both syntactic and semantic composability based on formal model component description, and includes
    - a semantic-based model repository
    - a collaborative environment for model development
    - an execution environment
    - and a process for component-based development using BOM-based simulation model components
2. Design and development of methods and techniques for supporting simulation model component discovery, match and composition
3. Utilization of BOM, SRML (Simulation Reference Markup Language), HLA, XML (eXtensible Meta Language), Semantic Web, Web Services and agent-based technologies for development of the proposed framework
4. Implementation of an agent based environment for realisation of the proposed framework.

1.4 Background technologies

In dealing with the above issues the work has been inspired by different research directions and technologies, each contributing in solving the main topic of developing a framework for CBMD. These research directions and technologies are briefly presented below, some of which have been directly used in this work.

In this thesis we have investigated the leverage-potential of BOM and a process has been developed on how to utilize BOMS as building blocks and simulation components for component based development of simulation models [13]. The concept of BOM has been extended with additional semantic information in order to support automatic discovery and composition of BOM based simulation model components.

Different simulation specification formalisms have also been studied, such as DEVS (Discrete Event System Specification) [10] and SRML (Simulation Reference Markup Language) [181] as languages for formal description of simulation models. DEVS has been chosen for its ability to describe and develop models using a mathematical formalism including well-defined separation of concepts supporting distinct M&S layers. SRML has been used for describing target simulation scenarios in a formal way.

The development and structuring of BOMs and ontologies has been done using the Semantic Web technology [157], Web Services, OWL (Web Ontology Language) [178], and OWL-S (OWL for Services) [131]. We have also studied some of the Web Service Composition techniques and made use of them for the purpose of composing simulation model components. The WS language that has been utilized in our work is OWL-S, which is an upper ontology to structure WS. In this research OWL/OWL-S has been used to structure the semantic annotation that has been added to the BOM descriptions as an attachment to enhance the semantic expressiveness of them [36].

The automatic code generation has been performed by utilizing the Frame Technology and XVCL (eXtensible Variant Configuration Language) [184]. XVCL's ability to capture common parts of simulation architectures provides us with the means to ensure syntactic composibility of simulation components.

The agent technology has been utilized in this thesis to develop an environment that glues all pieces in our framework together and automate the discovery and composition process. Here some of the features of agent based environments such as autonomy, social ability and adaptation has been in focus.

Besides the above mentioned, other technologies have been investigated and utilized in our work for development of the semantic-based model repository, collaborative working environment and the execution environment. These technologies include XML [174], GRIDS [48] and HLA [56].

1.5 The main contributions of the thesis

The focus of this research has been identifying the main components for CBMD, testing and developing technologies for implementing the CBMD framework, defining a process for discovery and composition, and developing an agent based environment for realisation of the process. The key questions to answer have been: (i) how to structure simulation components, containing *sufficient* semantic information to facilitate automatic/semi-automatic discovery and composition of components, (ii) how to identify suitable simulation components based on formal description of the target simulation model, and finally, (iii) how to perform model composition and reason about the composability of components. The framework should be able to check the compatibility of components based on semantic annotation defined in the component model and the related ontology information, and verify the resultant behaviours of the composed model. The latter requires a model composition process to describe the interactions and dependency of simulation components.

The automatic/semi-automatic discovery and composition aspects of the framework, aims at supporting simulation developers saving a lot of time and effort. This way developers do not have to manually find and study hundreds or thousands of components to find out which one is suitable for a simulation purpose. Moreover, using the semantic information inside components might ensure that the composition – when found – actually is optimal and will yield a better simulation result. Even a semi-automated process might provide some assistance to a simulation developer in terms of feedback on compatibility. Finally, an automated process might also support creating missing components needed in a simulation.

The main contributions of this research have been:

1. Design and development of the CBMD framework (see part II, papers 3 and 4) including
    a. a distributed, semantic-based simulation component repository (see part II, paper 8)
    b. a collaborative working environment (see part II, paper 1 and paper 5)
    c. an execution environment (see part II, paper 2 and paper 6)
    d. and a process for component-based development using BOM-based simulation model components and SRML (see part II, paper1 and paper 7)
2. Enhancement of BOM descriptions with a Semantic BOM Attachment, inspired from research within fields of Semantic Web and Web Services (see part II, paper 9)
3. Development of a rule-based method and three-layered model for discovery and composition of BOM-based components, including a method for matching BOM state machines (see part II, paper 9)
4. Design and development of an agent-based environment for discovery and composition of BOM-based components (see part II, paper 10)

1.6 Structure of the thesis

The thesis is structured in two parts.

1.6.1 Part I

This part consists of 7 chapters.

- In chapter two a survey of the Component-based Modelling and Simulation is presented, including an introduction to modelling and simulation, and distributed simulation.
- Chapter three is a brief survey of component-based methodology in software engineering, and different approaches for achieving this.
- Chapter four outlines a short introduction to the field of Semantic Web, Web Services and Web Service Composition including different matching techniques.
- Chapter five is an introduction to some related technologies that was studied and used in this work.
- Chapter six introduces NetSim and the CBMD framework that we have developed during this work as part of a project within the Swedish Defence Research Agency (FOI). This also includes the collaborative working environment, the distributed semantic-based resource repository, and the execution environment.
- Chapter seven is a description of our discovery and composition process, the Semantic BOM Attachment, the three-layered approach to composition of BOM-based components and the agent based environment developed to realize the above process including the agent architecture and implementation. This chapter also includes the evaluation of the proposed method and the conclusions.

1.6.2 Part II

This part consists of 10 published papers, presenting the main contributions of this thesis.

1. F. Moradi, R. Ayani, *Parallel and distributed simulation.* Applied system simulation – Methodologies and applications, book chapter, 2003, p. 457-486.

   Author's Contributions:
   The author was responisible for the sections regarding distributed simulations and different HLA services mentioned.

2. F.Moradi, R.Ayani, and G.Tan, *Some Ownership Management Issues in Distributed Simulations using HLA/RTI*, Parallel and Distributed Computing Practices, Vol. 4, No. 1, June 2001, Nova Science Publishers.

   Author's Contributions:
   The author is the main contributor of the paper; he designed and developed the air traffic control simulation test-bed, studied the Ownership Management services and compared different approaches, ran the experiments and analyzed the results.

3. G. Tan, F. Moradi, Yu Hu, *Automatic SOM Compatibility Check and FOM Development*, in Proceedings of the 7th IEEE International Symposium on Distributed Simulation and Real Time Applications, DS-RT 03, Delft, The Netherlands, October 23-25, 2003.

   Author's Contributions:
   This work was a collaboration between FOI and NUS. The author initialised the project, was the project leader at the FOI side, supervised the work and was responsible for integration of the developed software in the NetSim prototype.

4. F. Moradi, M. Eklöf, J. Ulriksson, *A network based environment for modelling and simulation*, in Proceedings of SimSafe Conference, June 15-17, 2004, Karlskoga, Sweden.

   Author's Contributions:
   The author was the project leader for the NetSim project. He initialised the project and the ideas behind it, and was responsible for design and implementation of the NetSim prototype.

5. J. Ulriksson, F. Moradi, M. Liljeström, N. Montgomerie-Neilson, *Building a CSCW Infrastructure based on an M&S Architecture and XML*, in Proceedings of the 2:nd conference on Cooperative Design, Visualization and Engineering (CDVE2005), Palma de Mallorca, September, 2005. Published in the Springer lecture notes, Springer ISSN: 0302-9743.

   Author's Contributions:
   The author was the project leader for the NetSim project. He was one of the designers of the system and contributed to development and analysis of the collaborative environment.

6. M. Eklöf, F. Moradi, R. Ayani, *A Framework for Fault-Tolerance in HLA-Based Distributed Simulations,* in Proceedings of WSC 05, Winter Simulation Conference, December 4 – 7, 2005, Orlando, USA.

   Author's Contributions:
   The author was the project leader for the NetSim project. He was one of the designers of the system and contributed to development and analysis of the execution environment.

7. F. Moradi, R. Ayani, P. Nordvaller, *Simulation Model Composition using BOMs*, in Proceedings of the 10-th IEEE International Symposium on Distributed Simulation and Real Time Applications, DS-RT '06, October 2006.

   Author's Contributions:
   The author initiated the project, designed the system and contributed to the development of the process.

8. M. Garcia Lozano, F. Moradi, R. Ayani, *SDR: A Semantic Based Distributed Repository for Simulation Models and Resources*, in Proceedings of the first Asia Modelling Symposium, AMS 2007, March 2007.

   Author's Contributions:
   The author was the project leader for the NetSim project. He was one of the designers of the system and contributed to analysis of the SDR.

9. F. Moradi, R. Ayani, G. Tan, H. Akbari, S. Mokarizadeh, *A Rule-based Approach to Syntactic and Semantic Composition of BOMs*, in Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real Time Applications, DS-RT '07, Crete, Greece, October 2007.

   Author's Contributions:
   The author is the main contributor of the paper; he initiated the project, designed the system and contributed to the development of the process.

10. F. Moradi, R. Ayani, I. Mahmood, *An Agent based Environment for Simulation Model Composition*, to appear in Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation PADS 2008, June 2008, Rome, Italy.

    Author's Contributions:
    The author is the main contributor of the paper; he initiated the project, designed the system and contributed to the development of the agent based environment.

Chapter 2

Component based modelling and simulation

This chapter gives a short survey of the topic of component-based modelling and simulation, and motivations behind component based approach to development of simulations/simulation models, as well as presenting main concepts and challenges. The survey begins with some background definitions regarding modelling and simulation, distributed simulations and related standards. Finally component-based modelling and simulation, the concept of composability and different techniques are presented.

2.1 Modelling and Simulation

Modelling and Simulation (M&S) is a method and tool that has been utilized throughout business, military, and academic community to find solutions to a great range of problems as well as an aid in decision making. M&S is a relatively inexpensive method for analysing and studying a system, whether it exists in the real world or yet to be built. It provides possibilities to reduce, amongst others, manufacturing and training costs and a means to study complex systems before actual physical prototyping. For instance, employing simulations early in the design phase of a product life cycle can detect and avoid costly errors in later stages of the development process. Furthermore, training of personnel using simulated systems can be done at a fraction of the cost of running exercises involving real systems. Some of the areas of application for M&S are [6]:

- development of military systems and concepts as well as for training, studies and analysis of military operations
- designing and analysing manufacturing system
- designing communication systems and protocols
- analysing financial or economic systems

The two basic concepts in M&S are of course, *model* and *simulation*. A model is a representation of something, such as a real world phenomenon. It is an abstraction, where those aspects of a system which are to be studied are highlighted and other aspects are simplified or omitted. A model can for instance be purely mathematical, represented as a Petri net, a queueing net or in a programming language. There are different types of models. Models can be static/dynamic, deterministic/stochastic, and continuous/discrete [6], [68].

Simulation is a process of running a model over a period of time, by e.g. using computers in order to evaluate the system under study. This way, simulations are used to describe the behaviour of a system in a formal way and to draw conclusions from it. A simulation constitutes a collection of entities that represent objects in the system, events that represent interactions between the objects and a set of states that represent collected information about the objects at each time interval. A simulation is of the same type as the corresponding model, i.e. static/dynamic, deterministic/stochastic, or continuous/discrete. In this work the focus is on discrete-event M&S [6].

A model that is developed for the purpose of simulation here is referred to as a *simulation model.*

2.2 Distributed simulation

As mentioned above, continuous time and discrete time simulation are two separate classes of methodologies, which have emerged over the years and are widely used for simulating complex systems. As the terms indicate, in a continuous simulation changes in the state of the system occur continuously in time, whereas in a discrete simulation changes in the system take place only at selected points in time. One kind of discrete simulation is the fixed time increment, or the time-stepped approach [6]; the other kind is the discrete event method [6], [147]. Thus, in a discrete event simulation (DES) events happen at discrete points in time and are instantaneous [10].

The traditional DES, as described above, is sequential. However, many practical simulations, e.g. in engineering applications, consume several hours (and even days) on a sequential machine. An alternative solution would be to use parallel simulation (PDES), where several processors cooperate to execute a simulation program and complete it in a fraction of the time that a single processor would need [75], [91]. Hence, parallel simulation techniques provide a way for reducing the execution time of simulation programs.

The rationale for distributed simulation (DS) is however different. Computer simulations started as simple computer programs that ran a set of predefined calculates and returned a result, but as simulations have become more and more complex, the need for distributing the execution of simulations has increased [42]. Hence, the focus of DS is to enable several simulations to interact and collaborate on a network of computers [25], with the objective of promoting reuse of simulation models, interoperability between simulations models and scalability of simulations. Today there are two IEEE standards for distributed simulations (Distributed Interactive Simulation - DIS and High Level Architecture - HLA) and it is normal that large simulations exercises are executed on distributed platforms. Doing so, several computers aid in processing events and doing calculations. In the following sections some of the techniques and standards in DS, such as Simulations Networking (SIMNET), DIS and HLA are presented and described.

2.2.1 Simulation Networking (SIMNET)

The first step toward distributed simulation was taken by ARPA (Advanced Research Projects Agency) in 1984 with the start of the SIMNET project [148]. In SIMNET, the simulation models, which were originally built as stand-alone simulations, were connected together through network protocols to participate in joint simulations. In 1986 two tank simulators were connected for the first time to interact with each other in a simulation. A scenario consisted of hundreds of simulators located on different parts of USA were developed by April 1988. The initiative proved to be successful. In the developed scenario it was even possible to let real live entities with communication and visualization equipment, participate in the simulations in the same manner as the simulated entities, [173].

The success of SIMNET proved that it was technically possible for models to interact, and simulations could be used in a much broader sense. The conclusion was that instead of building large stand-alone simulation models, for each training or analysis, one could develop small and specialized models that would interact with each other. Hence, it would be possible for models to be reused in different simulations, which were built in a modular fashion [25].

2.2.2 Distributed Interactive Simulation (DIS)

The success of SIMNET and introduction of specialized models that interact with each other raised the issue of interoperability and the need for a standardized communication interface, which has been the main concern of DS developers and users since 1990. In order to handle the interoperability issue DoD (Department of Defense) in USA proposed to adopt DIS as a standard framework for communication among simulations. The proposed DIS standard was approved by IEEE as IEEE 1278 in 1993.

The idea of DIS was to enable simulation models to broadcast and receive information over a local or a wide area network [167]. In the DIS architecture, each simulation model is an autonomous node and there is no central node. Nodes can enter or leave an ongoing simulation exercise. The communication of information is done, by broadcasting protocol data units (PDUs) over a computer network using the User Datagram Protocol/Internet Protocol (UDP/IP) [46]. The PDU formats are specified in the DIS protocol standards. A model broadcasts PDUs either in a case of an event occurrence or to inform the other models about its status. Other principles and features of DIS are the transmission of ground truth information and dead reckoning [97], [167], [168]. Dead reckoning is used in DIS to minimize the network communication cost.

DIS has been mainly used in "synthetic military exercises", but in the recent years, the work in DIS community has been expanding to encompass both military and commercial applications such as entertainment, air traffic control and emergency planning. Although DIS has been successful in many aspects, it is limited as a framework for distributed simulations. To start with PDUs are very rigid, e.g. in a case of new event types, new PDUs must be created and introduced. Moreover, since all PDUs are broadcasted (to all simulation nodes) the communication cost in DIS is very high and a lot of irrelevant information is communicated between nodes. Another problem with DIS is the lack of time management. All simulation models in a DIS exercise must have the same timescale. Hence, it is only suitable for real-time simulations. In order to solve the above problems a more general infrastructure had to be developed. The result was a shift from the existing DIS infrastructure to the new High Level Architecture (HLA). In 1996 DoD reassessed the interoperability issue and mandated that all M&S be HLA compliant by 2001. The strength of HLA was that it took advantage of current and emerging technologies.

2.2.3 Aggregate Level Simulation Protocol (ALSP)

ALSP is the DoD's standard for connecting constructive and time managed military simulations supporting analysis and training. The ALSP project was initiated in 1990 through ARPA to examine the feasibility of extending the distributed environment utilized by SIMNET to existing, so-called "aggregate" combat simulations [140].

By aggregate level simulations we mean simulations, which represent fundamental military entities such as battalions and fighter squadrons. Entity level simulations are used to train on a small scale (e.g., for individual soldiers), whereas aggregate level simulations provide a training environment at much larger scales for training command and battle staff [97]. ALSP uses an event-oriented approach and has mainly been developed for constructive simulation [150]. The protocol guarantees causality by applying conservative synchronization mechanisms based on the Chandy-Misra-Algorithm [91].

ALSP can be regarded as a first starting point for supporting interoperability among heterogeneous systems. In the terminology of ALSP, different participants (called confederates) form a common distributed simulation (called confederation). Data transmissions follow a broadcast principle (as under the DIS protocol). ALSP (as well as HLA) employs an object-oriented world-view. A confederation models objects with attributes. Ownership transfers of objects are possible between confederations. ALSP already contains a number of similarities to HLA and can be regarded as a subset of the HLA standards. Certain services are still missing (e.g., time management among different kinds of simulations, data distribution management). ALSP Infrastructure Software (AIS) [1] is analogous to RTI in HLA and can be regarded as a distributed operative system, which provides a set of basic services to confederates, mentioned above.

2.2.4 The High Level Architecture (HLA)

HLA also provides a framework for simulation models to participate and interact in joint simulation exercises [43]. It has adapted ideas from both DIS and ALSP and aimed at addressing the shortcomings of those approaches. While DIS is most suited for real-time simulations and ALSP is developed to handle constructive aggregate level simulations, simulations in HLA can be of different types, developed for different purposes, at different aggregation levels and with different timescales [57], [77]. These simulations also referred to as *federates* are connected and communicate with each other through a distributed operating system (HLA Runtime Infrastructure, RTI) and build a joint simulation, which is referred to as *federation*.

The main purpose of HLA is to facilitate interoperability among simulations and to promote reuse of simulations and their components. The HLA is composed of three major components: (1) HLA rules, which must be followed in order for the federation and federates to be considered HLA compliant; (2) HLA interface specification, figure 1; and (3) HLA object model template (OMT), which is used to describe objects and interactions in a federate/federation with their attributes and parameters [42]. An object model describing a federate is called Simulation Object Model (SOM), and the object model that defines a federation is called Federation Object Model (FOM).

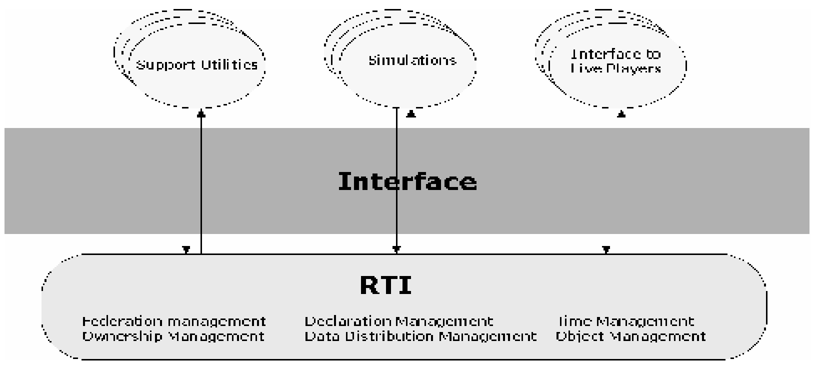


Figure 1 HLA Runtime Infrastructure and the associated services

The HLA Interface Specification is realised in a collection of software called Runtime Infrastructure (RTI). RTI provides common services required by multiple simulation systems. The services, which are described by the HLA interface specification [57], fall in six categories:

- *Federation management:* Provides functions for creating, modifying, controlling and destroying a federation execution. After creating a federation execution federates join and resign the federation as they wish as long as it serves the purpose of the simulation.
- *Object management:* Federates create, modify or delete objects and interactions through Object management services.
- *Declaration management:* Provides federates with the ability to express their intentions or interests in publishing or subscribing to object attributes and/or interactions.
- *Time management:* Provides a flexible and robust means to co-ordinate events between federates.
- *Ownership management:* Provides federates with the possibility to exchange ownership of object attributes among themselves.
- *Data distribution management:* Provides mechanisms for efficient routing of information among federates.

2.3 Component based M&S

As mentioned in section 2.1, simulation models have become more complex, and demand more resource and time to develop. Furthermore, there is also the issue of quality and usability of the developed simulation models. A simulation model development process involves different phases including, requirement specification, conceptual modelling, design, development, test, verification and validation, which requires involvement of different actors such as modellers, subject matter experts, validation and verifications experts and end users.

Issues such as, time and resource costs, quality and usability are some of the challenges that may prevent proper utilization of M&S. These challenges partly depend on the size and complexity of simulation models, but there are also other factors, as they have been pointed out by [5], such as:

- Increasing system complexity (e.g., networks of systems)
- Decreasing cycle times for system innovations
- Increasing lifetimes of systems
- Increasing variety of M&S-aspects and purposes, e.g., safety, reliability and so on
- User acceptance: Ease of use and credibility
- Integration of virtual and augmented reality

An approach to address the above challenges and reduce the costs associated with the development process and improve the usability of models, is to compose simulation models through reuse of predefined and already existing, validated simulation components [141], [137]. Using this method the simulation model is built in a component-based and modular fashion. The component based methodology has the potential to help reducing the costs of development of simulation models and to improve the quality and user-worth of those models. This approach has been amongst others pointed out by [5], where the author proposes introduction of a model engineering process "based on reusable and interoperable model components". DS techniques as explained in previous section are one method for addressing this issue.

The concept of component-based development has been successfully deployed in engineering desciplines, such as manufacturing, hardware and partly software industry. Software engineering has adapted this method through e.g. employment of object-oriented methodology and techniques [8], and technologies such as CORBA [78], EJB [156] and COM+ [20], which have simplified the development of complex software systems.

However, component based development approach is fairly new within the M&S community and developing models through composition is not trivial. Today, there is a lack of clear understanding of what component based M&S is, how model components could be structured, how they are to be specified and what technologies exist to facilitate syntactic and semantic composition. There are various techniques available for component based M&S, but there is no general approach and the terminology in the field is not mature. The existing simulation composition environments are mainly developed for specific simulators (simulation frameworks) and within specific domains [44], [29], [141]. One of the main reasons presented in the literature is that the issue of reuse and *interchangeability* of components is very hard to tackle and not practically possible if components are expected to be used in different compositions [44]. Especially since these compositions can differ from the purpose and the context for which the components have been originally developed [92]. Hence, the suggested solution according to those approaches is that components are interchangeable if they have been built with the same simulator. However, this is far from being the only issue that this field is faced with. Paul Davis in [141] sets up a list of factors affecting the difficulty of component-based M&S, which fall within the following four categories:

- Complexity of the system being modelled, including the size of the system being modelled, number of interfaces, parameters and messages to be exchanged among components, and the complexity of the components themselves.
- Difficulty of the objective for the context within which the model or simulation is being composed, which may depend on the uncertainty of the objectives, the degree of control needed, the degree of flexibility, and the degree of plug-and-play sought.
- Strength of the relevant science and technology, including standards. Here lack of knowledge about the system to be modelled, legacy modules, and the field of M&S can be contributing factors to the increased difficulty of CBMD.
- Quality of human considerations, such as the quality of management, the skill and knowledge of the work force, and also having an interest community.

Figure 2 illustrates the above categories in a diagram, where the x-coordinate represents the size and complexity of the system to be modelled and the y-coordinate presents the risk of failure.

In the next sections some of the techniques (architectures and frameworks), which have been developed or formulated to overcome the above difficulties are outlined.

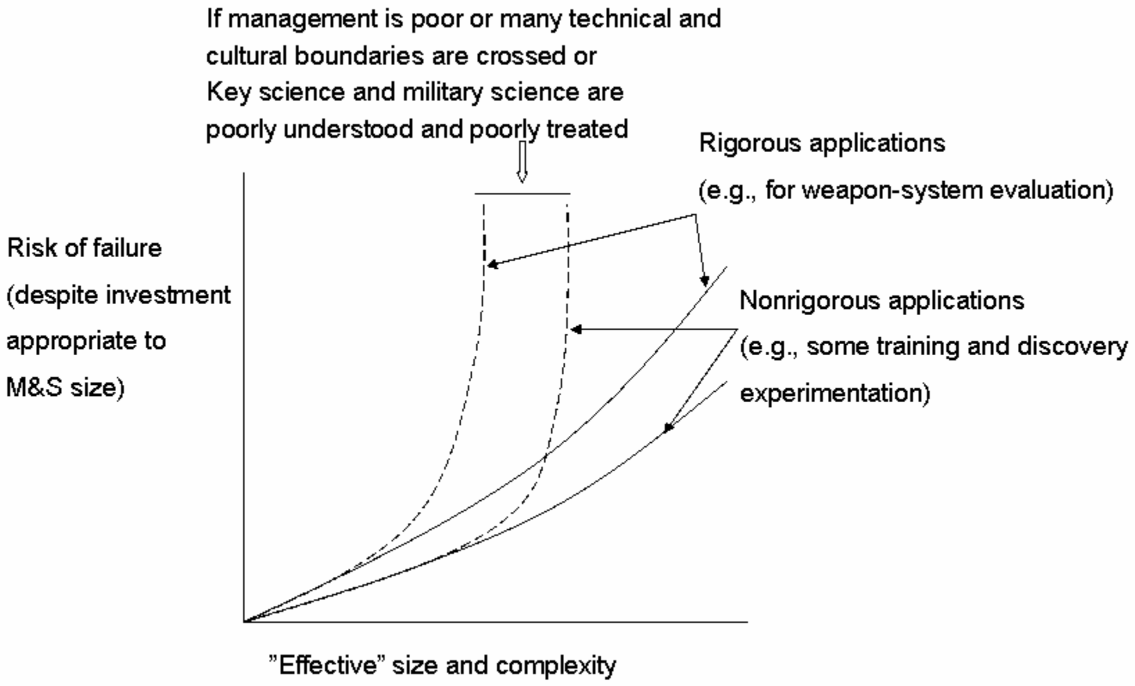


Figure 2 Diagram illustrating project risk vs. various factors (taken from [141])

2.3.1 Component-based modelling and simulation techniques

As mentioned previous section, there is no general approach to component-based M&S, rather various techniques exist for handling the issue. In this section two different classifications of these techniques are presented and the relation between them is discussed. In [5], Lehmann makes the following classification of different current techniques for component-based M&S:

Hierarchical modelling via decomposition and aggregation: Hierarchical modelling can be used and found in all aspects of systems modelling, and is the technique of building a larger system from smaller subsystems down to basic building blocks [10]. An important concept in this technique is decomposition, namely how a system may be broken down into a set of components. A second concept is that of composition, i.e., how components may be coupled together to form a larger system.

The main objective here is to be able to form an aggregated component by assembling a number of other components and hence, reduce complexity and enhance scalability.

Generic modelling object templates using, e.g., class/object libraries: This approach provides a basic structure, in the form of an object-oriented framework, to develop models in an easier fashion. DESMO-J [147], developed by the University of Hamburg, and JAMES II [85] , developed at the University of Rostock, are two examples of such object templates. It is an object-oriented framework for discrete-event simulation developed in Java. The general approach is similar to many other simulation systems, where models can be inherited from skeletal base classes, offering pre-defined methods which need to be overridden and implemented. There are a number of advantages to this approach. First of all there is no need to learn a specific simulation language, since many programmers are familiar with object-oriented development and are proficient in Java. Secondly, simulations written in a framework such as DESMO-J can be easily integrated in other software systems. However, there are also some disadvantages, such as lack of formal description of the behaviour, which is useful for model component composition.

Function/Program libraries: The aim of this method is to provide tools and utilities helping modellers, such as random number generators, input-data analysis, visualisation of statistical data, etc. These utilities are not an integral part of a model, but common to all of them. To express that a model component needs a certain service or library to fulfil its task, context dependencies are modelled with respect to tools/libraries.

Coupling of monolithic models: This method was partly described in the previous section describing the field of distributed simulation. The goal here has been achieving simulation reusability and interoperability through the coupling of models, which might be monolithic and not being constructed to work with other models in the first place [77], [142].

Another classification of current techniques is done in [29]. These techniques have been referred by the authors as engineering approaches to composability. By engineering the authors mean that these approaches mainly address the syntactic aspects of the compositions. These techniques are as follows:

Common Library Approach: This approach utilizes a library of reusable software modules. The library contains components in varying levels of composability, where no component is a stand-alone simulation. A single team is responsible for development of components, which are to be combined in different combinations. The approach relies on documentation to enable component reuse. It is based on open architectures, and provides tools, services, standards and interfaces for component-based simulation development. JMASS (Joint Modeling and Simulation System) is an example of the Common Library Approach [171].

Product Line Approach: This approach "provides a contained simulation development system utilizing layers of products for development of specific simulation systems and enabling comopsability" [29]. It includes services and tools for modification and reuse of components, and development, configuration, execution and analysis of simulation and models. Components in this approach are at various levels of composability, and as in the case of the Common Library Approach, none of the components are stand-alone simulations. However, the components may be written by different teams and still work together in various combinations. In this approach metadata and documentation is needed to ensure component reuse. OneSAF (One Semi-automated Forces) is an example of a system based on this approach [1]. OneSAF consists of various products, which all contain a set of standard components. These products include military scenario planner product, model and simulation composer product, simulation generator product, simulation core product, etc.

Interoperability Protocol Approach: As the name implies this approach utilizes an interoperability protocol such as DIS, ALSP or HLA for run-time exchange of data or services between different components. Components in this approach are simulations that basically can run independently, except for sending and receiving data. Using a standard interoperability protocol provides an open architecture. However, these protocols mainly provide data transfer, and semantic composability may be achieved manually. There are a large number of systems that use this approach such as, the Combined Arms Tactical Trainer (CATT) [53], the Joint Simulation System (JSIMS) [139], and the Close Combat Tactical Trainer (CCTT) [53].

Object Model Approach: The prerequisite of this approach is a standard for model specification, in which the models (components) are reusable only after modification or the development of suitable interface. Components are not stand-alone simulations, and documentation is needed in order to facilitate reuse. The main candidate for this approach is Base Object Model (BOM) [159], which provides an open architecture. The main purpose of BOM is to improve interoperability, reuse and composability by providing *patterns* and *components* of simulation interplay to be used as building blocks in the assembly of simulations and enterprises of simulations. The BOM is designed to work with interoperability protocols such as HLA. However, it is not a requirement. The component-based framework which utilizes this approach may include tools, services, interfaces and standards [34].

Formal Approach: The purpose of this approach is defining composability in a theoretical and mathematical manner using a simulation formalism such as, Discrete Event System Specification (DEVS) [10] and the Semantic Composability Theory (SCT) [101]. This approach uses formal methods for proving how components can be composed. DEVS is a formalism, introduced by Bernard Zeigler et al, for component based model development, which addresses engineering composability i.e. the syntactic aspects of compositions without proper support for ensuring validity of those compositions. SCT is a formal theory for composability, where a model is defined as a function, a simulation is viewed as execution of a function, and model composition is seen as a composition of functions. SCT addresses semantic composability of components. However, SCT (as will be explained in 2.4) has a limited scope, is still very theoretical and there is no formal language for describing models.

Please note that this is not a complete list and there are other approaches, which have not been covered here. What is notable however is that there are a large number of research and development efforts spent in this direction, and many frameworks have been constructed to reduce complexity of the M&S development process. Moreover, the two classifications mentioned above overlap and what they mainly have in common is that most of the approaches cover syntactic aspects of compositions. Semantic is basically achieved within specific application domains through precise specification of interfaces, which of course provides limited reusability. Syntactic and semantic aspects of compositions, brings us to the next topic, namely composability.

2.3.2 Composability

Composing sub-models to build new models raise the non-trivial issue of composability. There are different definitions for composability in the literature. One of the most accepted and quoted definitions, is by Petty et al, where composability has been described as "*the capability to select and assemble reusable simulation components in various combinations into simulation systems to meet user requirements*" [30], [101], [102], and [103]. What characterizes composability according to this definition is the ability of combining and recombining components in different compositions [101].

Another view on composability that relates highly to this work can be found in [93]:

"Given a set of components, structured descriptions or specifications of the components, sometimes called meta-data or meta-models, can be used to guide the process of selecting components for a specific purpose and determining if a set of components can be effectively composed."

There are three main types of composability, syntactic, semantic and pragmatic. Syntactic composability is concerned with the compatibility of implementation details, such as parameter passing mechanisms, external data accesses, and timing mechanisms. It is the question of whether a set of components can be combined [101], [16]. Semantic composability, on the other hand, is concerned with the validity of the composition [30], i.e. if the components that are composed can have meaningful communication and reach the simulation goal together. Pragmatic composability is yet another type which is concerned with the context of the simulation, and whether the composed simulation meets the intended purpose of the modeller [104].

Within some specific application domains, model components have already been implemented, and syntax specifications for composition of model components are available [185], [16], [55]. Hence, there have been some significant achievements in syntactic composability both within software engineering and simulation communities. However, semantic and pragmatic composability is a much harder problem [101], [35], [137] and has inspired many researchers to conduct both theoretical and experimental research. In this work the focus is mainly on syntactic and semantic composability and pragmatic composability has only been partly adressed.

A good example regarding syntactic composability is HLA, which provides a simulation environment and standards for specifying simulation (federate) via Simulation Object Models (SOMs) and interactions between simulations (federation) via Federation Object Models (FOMs). However, the problem with HLAs current standards for describing federates and federations using SOM and FOM, is that they contain only enough information for an underlying runtime implementation to assure that each federate/federation is behaving as it has promised via its SOM and FOM. Although a well-versed federate developer might be able to read a FOM or SOM and deduce how it works, neither of the formats were designed to contain semantic information about what they intend to simulate. Hence, HLA provides interface specifications and rules which only facilitate technical aspects of compositions, i.e. syntactic composability [57], and there is little support for semantic composability. As an attempt to handle the above issue, the (distributed) simulation community has recently formulated a standard, the Base Object Model (BOM), to ease reusability, composability, and interoperability [158]. This is a very promising standard, which we have investigated in this work. A detailed description of BOM will be presented in the section 2.5.

However, before we attempt to solve the issue of composability we need to better understand the concepts and the kind of approach required to do so. As mentioned before there is a lack of clear understanding of what component based M&S is, and how model components could be structured and specified. A very interesting initiative for facilitating better understanding of the concepts and laying the foundations for component based simulation model development is taken by [102], where the authors describe composability by giving it a clear definition and also clarifying its lexicon. Amongst others they depict nine different levels of composability, based on what is being composed, i.e. components, and what the result of the composition is. Composability as it has been described and referred to in the literature, falls within one of these following levels:

- *Application:* this level comprises applications that are composed together to build simulation events, exercises or experiments.
- *Federate:* a representation of this level is HLA, as explained earlier, where simulations are connected together to build distributed simulations that communicate in run-time.
- *Package:* simulations are being composed through utilization of pre-assembled packages of models, which form a subset of the domain of interest.
- *Parameter:* here is the focus on configuration of pre-existing simulation by using parameters.
- *Module:* modules are used here to compose executables
- *Model:* at model level models at smaller scales are put together to develop models at larger scales
- *Data:* databases are developed through composition of sets of data.
- *Entity:* where units such as military forces are build by composing entities/platforms.
- *Behaviour:* behaviours at lower level are composed to build higher level behaviours to be used by computer generated forces or in constructive simulations.

These levels are directly related to different initiatives and techniques (partly described in the previous section) that have been developed, all aiming at facilitating component based development and in some cases providing architectures and environments for that purpose. Besides their differences regarding what a component is and how compositions are accomplished, they support different degrees of composability. Realistically the vision of plug-and-play is more of a utopia and in reality we would always need some degree of component adjustment and adaptation before we are able to compose a set of components [141]. This adaptation is done through reconfiguration of simulations, adjusting the models and giving them proper interfaces, making changes in the existing simulation code, etc. In either case, these adaptations are hard to make without proper documentation [141], [137]. It is much easier for a human to read and understand a textual description of a component than a program code, and use it as a basis for selecting and adapting the component. Thus, documentation is a critical constituent for supporting successful composition, requiring that composability should be pursued at the conceptual and description level. Without proper documentation, incompatibility of components can not be discovered until after assembly and testing of compositions. Furthermore, to support modellers through automation of the selection and adjustment processes, and formal reasoning about the prospects of composability, these descriptions should be presented in a formal way. Experiences from the software engineering community, where this type of documentation and formal reasoning has been implemented, have the potential to provide a good support for component-based simulation/model development and hence provide potential path to progress. One of the technologies that can have great benefit for component based modelling and simulation is Semantic Web, which has been used in this work. In [144] a comparison between component-based software engineering and simulation composability is done. The authors also point out three different technologies from the software engineering domain with the potential to support simulation developers. These technologies are:

- Predictable Assembly from Certifiable Components (PACC) and Prediction-enabled Component Technology (PECT)
- Web Ontology Language (OWL) and Semantic Web
- Unified Modeling Language (UML) and Model Driven Architecture (MDA)

The above technologies are further explained in chapter 3 of this thesis.

2.3.3 Interoperability

The issue of interoperability has been of major concern within the modeling and simulation community [167]. The rationale behind interoperability is that connecting systems of various types developed for different purposes, during different technological eras and for different platforms, inflicts major difficulties. It is required that systems are capable of inter-communication, but also that the communication is semantically and syntactically agreed upon. If these basic requirements are not met, systems may interoperate for the wrong reasons.

Luckily, the rapid development of web/network related technologies brings new possibilities for overcoming the interoperability barriers and problems related to availability and management of resources. For example, through new ways of exchanging data, XML [174], distributing resources such as P2P [89] and Grid computing [48], and assuring semantic and syntactic correctness via Semantic Web initiative [157].

Interoperability in M&S is closely related to composability, and in order to avoid confusion it is of interest to separate these concepts. However, it should be noted that interoperability is not only restricted to the field of modelling and simulation. Consequently, there are various definitions of interoperability available. In the context of IEEE interoperability is defined as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [63]. The Department of Defense in US has several definitions of the term basically all related to weapons systems [88]. In [64] interoperability is described as “the ability of a system to use parts or equipment from another system”. And finally, in the context of modelling and simulation, interoperability has been defined as the ability of different simulations connected in a distributed system to collaboratively simulate a common scenario [100]. Even if the definitions are slightly different and represent different disciplines, they are some communalities between them, such as the concepts of system, communication, reuse, and mutual understanding, which are of course also essential concepts in composability.

Furthermore, similar to composability there are two types of interoperability, technical and substantive. Technical interoperability requires that simulations (components) are compatible with an interoperability protocol. The interoperability protocol is responsible for exchange of data between simulations. Substantive interoperability on the other hand is satisfied if the exchanged information is semantically meaningful [100].

The difference between composability and interoperability that has been pointed out in the literature is that components that are interoperable in one configuration, but cannot be combined and recombined in other ways (*without significant effort*) are not composable. In other words, interoperability is a prerequisite of composability, i.e. components that are composable are also interoperable. However, the reverse relation does not hold, i.e. components that are interoperable are not necessarily composable.

2.3.4 Simulation model components

So far we described what component-based M&S is in the context of this work and how we define composability. But no definition has been given concerning simulation model components, which is the purpose of this section.

We begin with differentiating between *simulation components* and *simulation model components*. A simulation component in our terminology is an executable program, while a simulation model component is a description of a model in a formal way. For instance an HLA federate is a simulation component, and a BOM is a simulation model component which does not include any program code. Now having that in mind when we go through the literature we do not find any clear and unambiguous definition of what a simulation model component is. For instance a component can either be a stand-alone simulation or not. It is also unclear how much a component reveals about its internal functions.

Since a simulation is a special type of software, a good place to start to get some idea and describe a simulation model component is to look at what a software component is. There are different definitions of software components in the literature. The most informal definition is given by [14], where the author means that a component is *"whatever you want it to be"*. Unfortunately, this definition does not help our case that much, but it gives some indication regarding the state of the field.

A more formal and precise definition is presented in Unified Modelling Language (UML) specification [133]:

"A component can always be considered an autonomous unit within a system or subsystem. It has one or more provided and/or required interfaces (potentially exposed via ports), and its internals are hidden and inaccessible other than as provided by its interfaces. Although it may be dependent on other elements in terms of interfaces that are required, a component is encapsulated and its dependencies are designed such that it can be treated as independently as possible. As a result, components and subsystems can be flexibly reused and replaced by connecting ("wiring") them together via their provided and required interfaces."

This definition is valid for any type of system, e.g. a simulation system. The two important aspects of the components (pointed out in this definition) are the interface that they provide in order to facilitate composition, and the fact that their internals are hidden and only accessible through their interfaces. Furthermore, the components can be deployed independently, be used and reused through their contractual specified interfaces [19].

A third, not so detailed, definition that has been of interest to us is given by [175]:

"A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard."

Here it is pointed out that a component should conform to a component model which is quite clear, and that components can be used (reused) in different combinations, as in the previous definition. However, according to our understanding of composability, the above definition is quite rigorous when it states "…*and can be independently deployed and composed without modification*…". In our opinion some degree of component modification is acceptable.

Another issue to consider is if a component should be perceived as a black-box, which it has traditionally been seen in software engineering, and is evident from the above definitions. We believe that in order to be able to reason about the composability of components, besides the knowledge about the external interfaces of the components, we need to have access to information regarding the internal behaviour of the components. It is not about implementation details rather information about the internal states of the components, and how and when they change. This is more like looking at the component as a white-box [141], rather than a black-box. This view has also been adapted by the software engineering community as discussed in chapter 3.

Based on the definitions above, our discussions, and the concept of composability, presented in the previous section, we have adapted the following definition of simulation model component in the context of this work.

"A composable simulation model component is an autonomous element of a simulation model, which conforms to a component model, and has well-defined functionalities and behaviours, presented via its interface describing its communication with other components and a formalised description of its internal behaviour. A simulation model component is not a stand-alone component, but can be independently deployed, and it is subject to third-party composition with or without modification."

2.4 Theory of Composability

In this section an introduction of the theory of composability is given. Theory of composability or "semantic composability theory" (SCT), which it has also been called, is an initiative developed at the Virginia Modeling, Analysis & Simulation Center (VMASC). The aim of the SCT is to check and prove the semantic composability of components using formal descriptions and reasoning [101], [103]. For this purpose formal definitions of basic concepts such as, model, simulation, composite model and validity have been suggested. These definitions act as a basis for building the theory. Furthermore, the theory has also two main assumptions. First of all, SCT assumes simple composition (horizontal composition), i.e. composition of the form *f*(*g*(x)), where the output of one model is the input of the next model. Second assumption is that data is passed between the models without same-step loops, i.e. output of one model is not input to itself (directly or indirectly) within the same computation step. This section gives a short description of SCT. For a complete description of SCT see [101].

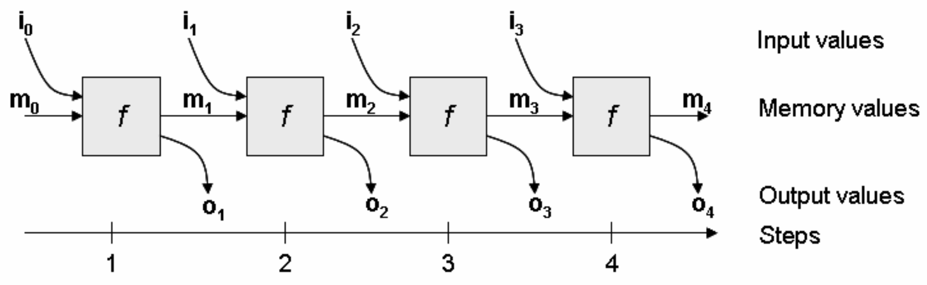


Figure 3 Presentation of simulation in the context of SCT (taken from [101])

In the context of SCT a model is a computable partial function, which transforms a set of inputs and current states (domain) to a set of outputs and consequent states[1] (co-domain). A computable function is a "*function that can be computed in a finite number of steps by a computer program or any equivalent model of computation*" [101]. And what is special about a partial function is that it associates each element of the domain at most to one element of the co-domain. This means that not all the elements of the domain have to be associated with an element of the co-domain.

Simulation has been defined in SCT as a sequence of executions of a model, where the output from each execution step is the input to the next step of the execution. This definition is equivalent with the description of synchronous systems [101]. Figure 3 depicts how simulation is done as an execution of a model through different steps, as presented in SCT. In this figure the execution of model *f* at each step is presented, where the input to the model at each step is partly from the previous execution step, $m_{x,}$ and partly from outside, i_x.

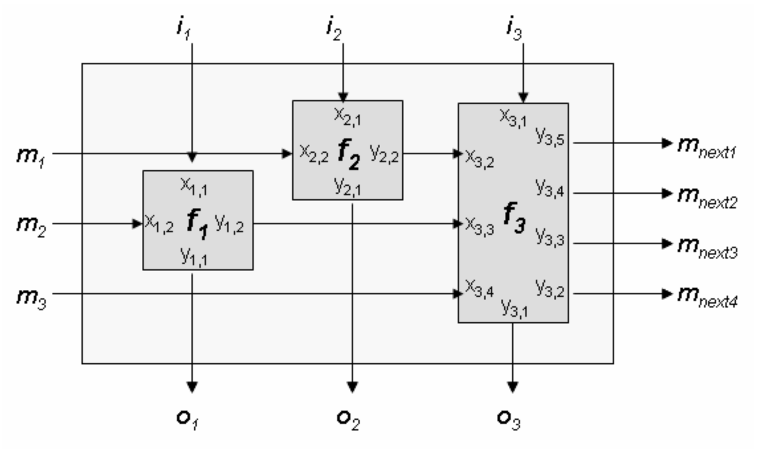

i_1
i_2
i_3
m_1
m_2
m_3
$x_{1,1}$ $x_{1,2}$ f_1 $y_{1,2}$ $y_{1,1}$
$x_{2,1}$ $x_{2,2}$ f_2 $y_{2,2}$ $y_{2,1}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ f_3 $y_{3,5}$ $y_{3,4}$ $y_{3,3}$ $y_{3,2}$ $y_{3,1}$
m_{next1}
m_{next2}
m_{next3}
m_{next4}
o_1
o_2
o_3

Figure 4 Composite model (taken from [101])

In SCT models can be atomic or composite. A composite model is developed through composing other models, while an atomic model is developed directly and not through composition. A composite model in SCT has a formal description and is defined as set of models, each having a set of inputs and outputs. The composite model as a whole has also a set of inputs, outputs, current states and next states. Figure 4 illustrates a composite model. Validity in SCT is defined as the relation between a model and the real system that the model represents. The validity of the model is based upon how closely its behaviour matches the behaviour of the real system. A more formal definition requires specifying a perfect model. A perfect model is a notional model of a system, where given a set of current states, it produces a set of next states that exactly match the next states of the modelled system. Hence, a more formal definition of validity suggests that a model is valid if the difference between the model and the perfect model is below an application-specific threshold. There is an even more formal definition of validity, which follows the same line as the above definition, but will not be covered here [30].

SCT also defines validity under composition and presents formal methods for identifying classes of models and validity relations for which validity is preserved. However, the theory mainly considers validity of compositions of similarly developed models and validity relations [30].

Another important definition in SCT is the concept of interface, which is a computable function that maps (copies) input variables and output variables of component models in a composition without changing any of their values. The interface may however, change the order of the input variables or ignore a subset of them.

[1] Note that there are formal descriptions of the terms. However, for the sake of simplicity no such descriptions are presented here.

Besides the definitions SCT includes theorems, and proves that if one has a finite set of component models and specification on how to connect them, it is enough to perform simple composition together with interfaces, to build any set of compositions from the component models. However, SCT, which is in its infancy, is a theoretical approach and in order to be able to use this theory one has to describe simulation components as functions, specify how they can be connected and define interfaces for compositions. Hence, SCT is the first phase in a four phase research program, where the other phases focus on specification of formal meta-models, development of a composability architecture and framework, and development of a simulation development framework [103].

Furthermore, validity of compositions is a general problem. In the context of SCT, there are couple of issues which one can point out. First is the question of how to access and develop a perfect model of a system, especially considering systems that do not exist. And second, even if we had a perfect model defining a threshold value is not a trivial task, particularly if the system under study is dynamic and stochastic.

2.5 BOMS

In this section a general description of the BOM concept, which is the chosen model for presenting simulation model components in this work is presented. As mentioned earlier the HLA standard does not provide sufficient semantic information for reasoning about semantic composability of federates. SOM and FOM contain only enough information to inform the underlying RTI implementation about how a federate/federation is going to behave, e.g. what objects are defined and updated and what interactions are being sent. Although a skilled federate developer might be able to read a FOM or SOM and deduce how it works, neither of the formats were created to contain information about what they intend to simulate, and how it is done. This shortcoming is one of the motivations behind introduction of BOM. The BOM concept has been introduced by the HLA community to provide key mechanism for reuse, interoperability and composability [159]. Through its formal description BOM also facilitate better understanding and management of components and simulations [12]. BOM is based on the assumption that piece-parts of simulations and federations can be extracted and reused as modelling building-blocks or components. The interplay within a simulation or federation can be captured and characterized in the form of reusable patterns. These patterns of simulation interplay are sequences of events between simulation components. The BOM standard is intended to influence the following seven capabilities within the M&S community [12], [159]: *Interoperability*, *Reusability*, *Composability*, *Adaptability*, *Aggregation*, *Multi-resolution Modelling*, and *Rapid Application Development (RAD)*. BOMs can also support migrating from existing system centric solutions to Service-Oriented Architecture (SOA) capable modelling and simulation services [159].

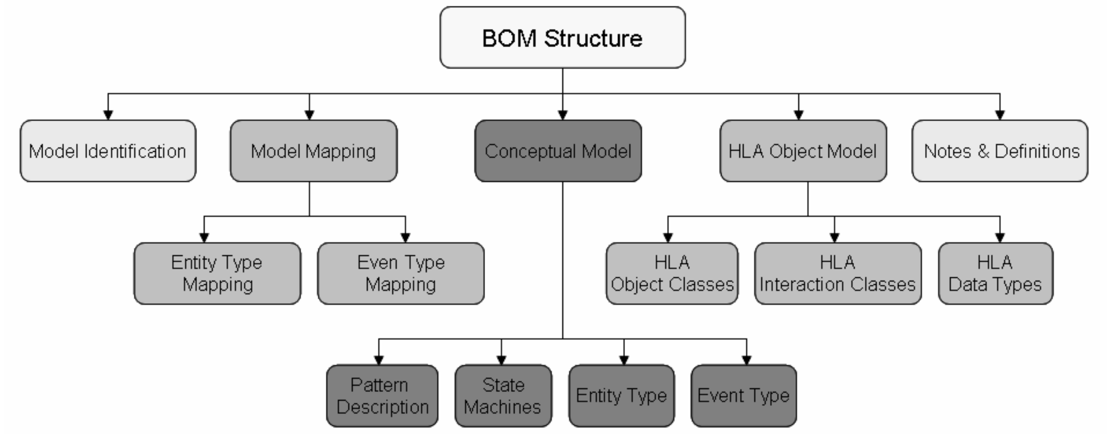


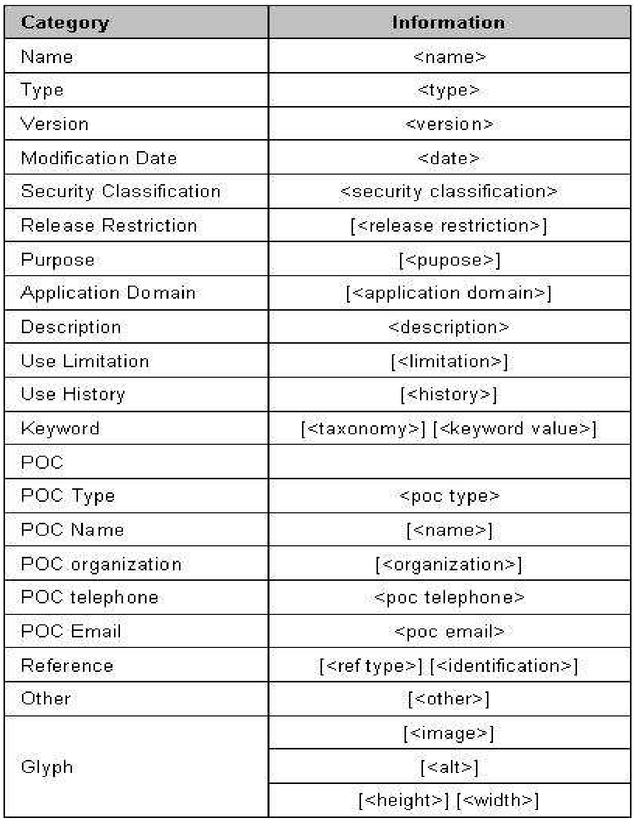
Figure 5 BOM structure

A BOM is an XML document that encapsulates the information needed to describe a simulation component. BOMs are meant to be used as components in a simulation, so that a federation developer can select a number of BOMs from for example a repository and compose them together into a simulation without needing to know the exact implementation of each component. This is similar to the use of components in other fields, as was describe in section 2.3.1. BOMs are the Modelling and Simulation community's proposed component standard. Even though BOMs were created based on ideas from HLA, meaning they include HLA OMT information, BOMs may be used without this information to describe any type of simulation component, a feature which makes them even more versatile. Since BOMs are still in their infancy, there are a limited number of tools available for BOM creation and modelling. The most comprehensive tool available is BOMworks [13] from SimVentions.

2.5.1 BOM Structure

BOMs are structured into four major parts [158] as can be seen in figure 5, Model Identification, Conceptual Model, Model Mapping and HLA Object Model. The first part, the Model Identification, is where metadata about the component is stored. This part includes Point of Contact (POC) information, as well as general information about the component itself – what it simulates, how it can and has been used as well as descriptions aimed towards helping developers find and reuse it. Figure 6 shows a complete list of all items in this part.

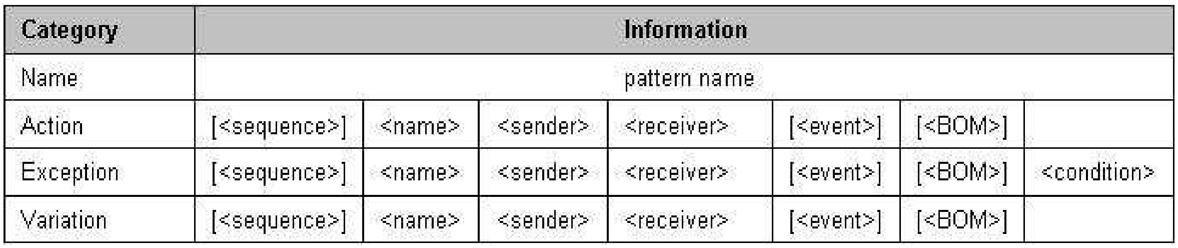
The second part is the Conceptual Model, which contains information that describes the patterns of interplay of the component. This part includes what types of actions and events that take place in the component, and is described by a pattern description, a state-machine, a listing of conceptual entities and events (conceptual entities and events correspond to how real-world objects and phenomena are modelled in the simulation), together describing the flow and dependencies of events and their exceptions.



Category	Information
Name	<name>
Type	<type>
Version	<version>
Modification Date	<date>
Security Classification	<security classification>
Release Restriction	[<release restriction>]
Purpose	[<pupose>]
Application Domain	[<application domain>]
Description	<description>
Use Limitation	[<limitation>]
Use History	[<history>]
Keyword	[<taxonomy>] [<keyword value>]
POC	
POC Type	<poc type>
POC Name	[<name>]
POC organization	[<organization>]
POC telephone	<poc telephone>
POC Email	<poc email>
Reference	[<ref type>] [<identification>]
Other	[<other>]
Glyph	[<image>]
	[<alt>]
	[<height>] [<width>]

Figure 6 Items of the Model Identification part of BOM

The Pattern Description provides a list of all actions taken by the component, and describes what conceptual entities and events take part in the action as well as describing different variations and exceptions to the action that might occur. Each action in the Pattern Description has a specified Name along with a Sender and a Receiver. Figure 7 illustrates the structure of the Pattern Description.



Category	Information						
Name	pattern name						
Action	[<sequence>]	<name>	<sender>	<receiver>	[<event>]	[<BOM>]	
Exception	[<sequence>]	<name>	<sender>	<receiver>	[<event>]	[<BOM>]	<condition>
Variation	[<sequence>]	<name>	<sender>	<receiver>	[<event>]	[<BOM>]	

Figure 7 Pattern Descsription

The State Machine, which is deterministic, provides means to formalize how the state of an entity is changed via actions. The State Machine lists all possible states of an entity, and the conditions, which must be satisfied in order to exit one state – via a specified action – and enter a next state (so called state transition diagram). This provides the BOM composer with a clear overview to of how a federate that is implemented from a BOM will behave and if it satisfies the needs of the end simulation. Figure 8 illustrates the state machine of a waiter entity, which has the states such as, ready/idle, prepare bill for the customer, present menu, and bring food (details of the waiter entity can be found in part II, paper 9).

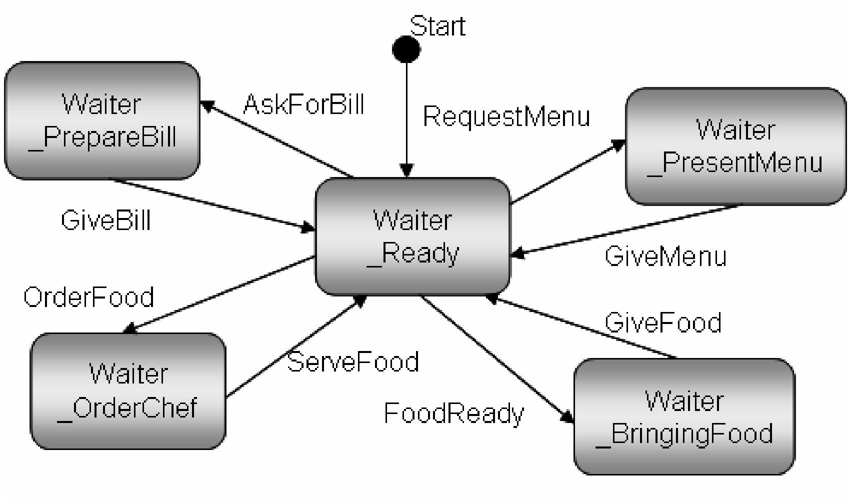
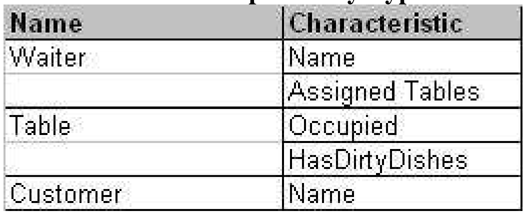


Figure 8 State machine of the Waiter entity

The Entity Type is a listing of all the Entities present in a BOM. An entity is, as previously explained, a conceptual mapping to a real-life object. In the above example entities might be Waiter, Customer and Table, each holding certain information. For example a waiter might have a Name and a list of Tables she/he is assigned to, and similarly a "Table" entity might include information indicating whether it is occupied/free and whether there are dirty dishes on it, as shown in table 1.

Table 1 Example Entity Types



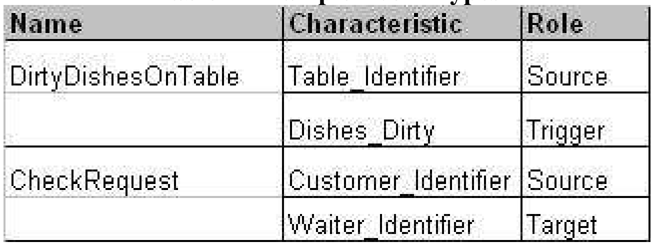
Name	Characteristic
Waiter	Name
	Assigned Tables
Table	Occupied
	HasDirtyDishes
Customer	Name

The Event Type contains a table of the Events present in a BOM. The BOM Specification contains two different types of Events, BOM Trigger and BOM Event. A BOM Trigger is an undirected event that is triggered by some entity without a specific receiver. The entity that is sending out a Trigger does not care about who receives a notification of the event. This is similar to Interactions and Object Attribute updates in HLA, where a federate does not directly notify another federate, but merely informs the RTI of the interaction or object attribute update and allows it to handle forwarding to any relevant federate. The other type of event, BOM Event is an event sent from one conceptual entity directed to another specified entity. This type of event is in fact directed to another entity similarly to how a phone call is made from one person to another.

Examples of Events can be a table announcing a Trigger that it contains dirty dishes. This trigger can be noticed by a waiter that would then come and clean up the table. Another example event would be an Event where a Customer explicitly asks a Waiter for the check. This would represent a BOM Event that is directed between two entities. The table below illustrates these examples in BOM representation.

The third part is the Model Mapping where conceptual entities and events are mapped to their HLA Object Model representations. This part essentially bridges the Conceptual Model with the HLA Object Model that is described in the fourth part of the BOM, the HLA Object Model. In the Entity Type Mapping, each entity is mapped to a HLA Object/Interaction class, and each Characteristic of that Entity is mapped to an Attribute or Parameter. In case there is an ambiguity of the Attribute or Parameter, a condition can be supplied to explain how the mapping should take place. This condition is not formalized in any way, and is a simple freetext explanation. Event Type Mappings are nearly identical to Entity Type mappings, but pertain to BOM Triggers and Events instead of Entities.

Table 2 Example Event Types



Name	Characteristic	Role
DirtyDishesOnTable	Table_Identifier	Source
	Dishes_Dirty	Trigger
CheckRequest	Customer_Identifier	Source
	Waiter_Identifier	Target

The fourth part, the HLA Object Model, contains the information that is found in a normal FOM/SOM – objects, attributes, interactions and parameters - and should conform to the HLA OMT. There are two additional parts in the BOMs that are not mentioned in-depth in this paper, namely Notes and Definitions. These two parts contain semantic information about events and entities as well as actions that are specified in the Conceptual Model, and are used to provide a human readable understanding of the patterns described in the BOM, figure 6.

2.5.2 BOM Assembly

The BOM concept provides a mechanism for combining BOMs and creating High-Level BOMs, called BOM Assemblies. A BOM Assembly is built in a hierarchical manner and includes information about other BOMs, which in turn provide more detailed information about components. Typically a developer of a simulation would search a BOM repository for suitable BOM candidates for use in a simulation and combine those into a BOM Assembly (i.e. a simulation model), which is then used to create the actual simulation. A BOM assembly is also defined as a BOM document, using the constructs described above, as explained in the BOM Guidance document:

"... a BOM Assembly must contain a Model Identification and a Pattern Description within the Conceptual Model view. A BOM Assembly also has associated with it the metadata from each integrated BOM, plus an understanding of the interplay among conceptual entities being represented, which is provided through the association of BOMs to the various Pattern Description actions that the BOM Assembly identifies." [158]

Since it references other BOMs, BOM Assemblies often only contain high-level information and not specific OMT-information.

2.6 Summary

In this chapter we discussed the issue of component based M&S, the rationale behind it, and some of its main concepts, such as composability and interoperability. We also outlined the challenges and different techniques addressing those challenges. One of the main issues regarding component based M&S as is shown in this chapter, is ensuring semantic composability and validity of compositions prior to the actual implementation.

Moreover we presented our definition of simulation model component and the concept of BOM. BOM is the component model chosen in this work as a basis for developing a process for component based simulation development. However, the current BOM standard does not represent a complete simulation model component according to our definition. Hence, in order to make use of BOM in our process, we have extended the BOM definition with additional information to facilitate discovery and composition of components through formal reasoning. In chapter 7 we cover the work that has been done to improve the BOM description for our purpose.

Chapter 3

Component based development in software engineering

Component based software development has been in focus for a long time, but has not become popular until the 1990's. It mainly started with different programming languages presenting methods for development of generic components/language constructs that could be (in some cases with slight modification) reused in different combinations. These language constructs, which are used for structuring programs, have various names such as, class, module, package, cluster, etc. For instance Ada had the concept of generic packages, which could be viewed as black boxes with well-defined interfaces for input and output of data [146].

Object Oriented Programming, OOP, paradigm with programming languages such as C++, Java, Objective C, C#, etc has been an essential effort in introducing component-based development in software engineering. The motivation behind OOP is to provide modularity, flexibility, and reusability [99]. In OOP everything in the real world is modeled as an object, with properties and behaviors. For instance a square is an object with edge as a property and CalculateArea as a method for calculating the area of the square. Such square object can be used and reused in all programs where an object with the above property and method is required.

Originally software components were small with simple semantics and interactions, thus the issue of composability was rather straight forward and mainly limited to syntactic compatibility of components. Component based simulation development on the other hand started by composing large and mainly monolithic simulations using protocols and architectures such as SimNet, DIS and HLA. However, these two directions are being converged, and as the complexity of software components and their interactions is increased, software engineers are faced with similar composability challenges as simulation developers [144]. Hence, several technologies and standards are being developed in the software engineering domain to address the issues of semantic validity of composition, which have the potential to support component based simulation model development [144].

Today component based software development is quite mature and has been accepted across the software engineering community. Technologies such as CORBA [133], EJB [156] and COM+ [20] are well-established and have simplified the development of complex software systems. These technologies however are component architectures mainly supporting syntactic composability. They provide means for components to expose their external interfaces and interact with each other, without support for ensuring the reliability and consistency of the exchanged information and the validity of the component compositions. Furthermore, even though these technologies are basically (in an abstract sense) quite similar in the way they function, there is little support for inter-architecture communication and building systems across different architectures. This is however something that has interested the community and efforts has been done to support interoperability across architectures [144].

To address to issue of semantic composability the software engineering community has promoted some component based development architectures/technologies that aim at separating the specification details (i.e. conceptual model) from the implementation, or application logic from platform technology [4]. They provide formalism for describing the components thus facilitating reasoning about the validity of compositions before actual implementation. This approach to provide semantic composability falls within the scope and ambitions of this thesis. Hence, three such technologies that promote the component-based development methodology, namely Predictable Assembly from Certifiable Components (PACC), Model Driven Architecture (MDA) and Architecture Description Language (ADL) are briefly presented in this chapter. It should be noted that this is not a comprehensive list, rather some approaches that have been studied here.

Another technology with great potential that has been in focus in this work is the Semantic Web technology, which due to the special roll that it has played in this thesis is being presented in a separate chapter (chapter 4).

3.1 Predictable Assembly from Certifiable Components (PACC)

Predictable Assembly from Certifiable Components (PACC) is an initiative from the Software Engineering Institute at Carnegie Mellon University (CMU) for automatic prediction and certification of runtime behaviour of component assemblies [71], [90]. PACC's objective is achieved through Prediction Enabled Component Technology (PECT) by development and enhancement of component technologies and related tools and methods. As opposed to technologies such as COM+ and EJB, PACC focuses on proving the validity of component assemblies before actual composition. PACC is an abstract and theoretical approach consisting of two main concepts, namely a construction framework, and a reasoning framework. The construction framework is responsible for specifying component models including their interactions with other components and reactions to input. The reasoning framework allows reasoning about component properties and their composability under composition.

The Construction Framework is an abstract component technology (ACT) plus tools that supports its utilisation, such as editors, repositories, and constraint checkers. ACT is component-technology-independent and defines a vocabulary and notation for specifying components, compositions, and their runtime environments. It also specifies the constraints, imposed by reasoning frameworks, which must be satisfied for predictions to be valid [90].

Components in PACC are the building blocks of the predictable assembly and quite similar to component descriptions in other formalisms, such as DEVS and SCT. The component is seen as a box with an interface including input and output channels facilitating communication with other components, and component behaviour specified as reactions. The behaviour decides how the component reacts to inputs and what outputs are generated. These reaction patterns are described using Communicating Sequential Processes (CSP). The input and output channels in PACC are called *pins*. *Assemblies* are set of components linked together and *interacting* through their pins. Assemblies can be part of larger assemblies, thus facilitating hierarchical compositions. The construction framework also includes a *component runtime environment*, which can be regarded as a high level component-aware, possibly application-specific virtual machine. The runtime environment provides runtime services that may be used by components in an assembly, an implementation for one or more interaction mechanisms, and a closure for, and containment of, all assumptions made by a reasoning framework.

Furthermore, component and assemblies in PACC are assigned some properties to support the reasoning process. In simulation terms, a property might be an inter-arrival time, or a lookahead value [147]. Construction and Composition Language (CCL) is a construction language proposed by [90] and [72] for specifying components, assemblies and their properties.

The next concept of PACC is the *Reasoning Framework,* which aims at supporting the construction framework in performing analysis on the properties and behaviour of assemblies. A reasoning framework includes the mechanisms needed to use sound analytical theories to analyse the behaviour of a system with respect to some quality attribute [95]. In the context of PACC the reasoning framework comprises a property theory, an automated reasoning procedure, and a validation procedure. This includes a mathematical formalism for describing properties of a system, logic for reasoning about the properties, and methods and tools for automatic reasoning about the validity of compositions and computing predicted properties. The Reasoning Framework exposes any assumptions about the system it models, as a basis for the component technology to ensure that components and assemblies satisfy these assumptions. These is done mainly through static checking, resulting in a predictable assembly behaviour.

PACC provides means for defining semantics of components and formal reasoning about compositions. This is what we believe to be the best approach for achieving semantic composability of simulations, making PACC to be a candidate technology for supporting component based simulation development.

3.2 Architecture Description Language (ADL)

ADLs are a family of languages developed to support a more abstract and formal modelling approach to architecture enabling reasoning about architectural properties without considering implementation details. ADLs have the potential to support component based systems due to their ability to specify context dependencies [118]. Hence, a short description of ADL and one example are presented here explaining how it could facilitate component based simulation development.

Traditional object modelling, where a component is represented by a class and class interfaces, has a number of drawbacks, such as lack of context dependency notations and means for specifying meta-properties by itself like system performance and/or semantics. ADLs address these shortcomings by having the ability to formally define components and specify how components interact and what meta-properties interfaces and their connections exhibit. The main common properties of ADLs are the *components*, *connectors* representing interactions between components, *systems* representing component/connector configurations, *properties* allowing semantics to be added, *constraints* representing claims about the architecture which remain true, and *styles* representing "families of related systems" [24].

There are large number of ADLs available developed for different purposes. One such language is CommUniy, which is a parallel design language based on UNITY, incorporating parts from Interacting Processes [74]. CommUnity is a formal language used to describe components, their state, actions and interactions, locality and how to *extend* them. Extending is the mechanism for building assemblies of components.

The basic building blocks of the formalism are not programs but abstractions of programs – called designs - that can be refined into programs in later stages of the development process. Designs have a very basic structure, allowing them to specify a set of *channels* or *ports*, and a set of operations, called actions. Ports consists of input, output and private channels, necessary for communication with the environment (i.e. other designs) as well as to store private data. Actions are either shared or private. Only shared actions can be *seen* by the environment and therefore used for interaction.

There are two concepts for extending designs/components defined in the CommUnity, namely *superposition* and *refinement*, where superposition is the one that is interesting for the purpose of component based simulation development.

In superposition a design is extended by overlaying it with additional attributes and behaviour, through what is called *composition morphisms*. Hence, the composition of designs is performed by overlaying their behaviour and properties, facilitating hierarchical component development. Furthermore, composition morphism must follow precise rules, not to violate the categorical integrity of a design.

ADLs through their formal modelling approach providing formalism for describing components and enabling reasoning about architectural properties without considering implementation details are interesting alternatives to investigate for the purpose of semantic composability of simulation models. In [118] the potential of these languages for component based M&S has been studied, which provides a good basis for further study.

3.3 Model Driven Architecture (MDA)

MDA is a software design approach and framework launched in 2001, but the concepts that underpin this approach were first discussed by [155] during the late 80's. MDA was the second framework sponsored by the Object Management Group (OMG), which as opposed to its predecessors CORBA and Object Management Architecture (OMA), is not a framework for implementing distributed systems, rather an approach to using models in software development separating designs from architectures [127]. Furthermore, the term architecture in MDA does not refer to the architecture of the system being modelled, but the architecture of various standards making the technology basis of MDA. These standards include Unified Modelling Language (UML) [133] below, Meta-Object Facility (MOF) [134], XML Metadata Interchange (XMI) [135], etc.

In MDA's context a model is a specification and description of a target system and its environment, where the system could be a program, a single computer system, or a federation of systems. MDA provides a set of guidelines for developing models, which are either computation independent, platform independent or platform specific. A Computation Independent Model (CIM) does not contain any details about the structure of systems. It focuses on the environment of the system and the requirements for the system. It is sometimes referred to as *domain model*, containing a vocabulary used in a specific domain. CIMs are used by practitioners that do not need to have any knowledge about the models and tools for realisation of functionalities of which requirements are expressed in the CIM.

A Platform Independent Model (PIM) is used to define system functionalities in a domain specific language or a general purpose modelling language. It focuses on the operation of a system without presenting any platform specific details. A PIM is platform independent in a sense that it is suitable for use with various platforms of similar types.

PIMs can be *transformed* to Platform Specific Models (PSM) given a Platform Definition Model (PDM) corresponding to CORBA, .Net, etc. PSM is an executable model that computers can run using domain specific or general purpose languages such as Java, Python, C#, etc. It combines the platform independent specification of a system i.e. PIMs with details about how the system utilises a particular platform. One rationale for separating the concepts and developing PIMs is that they represent the conceptual description of a system that potentially lasts much longer than the realisation technologies and software architectures.

The *transformation* from PIMs to PSMs is a *forward engineering* process that produces code from abstract specifications. This process can be more or less automatic. There contexts in which a PIM contains all the information required for automatic code generation. Component based development is one such context as been pointed out in the MDA guidelines [127], where an application developer can build a complete PIM with regards to classification, structure, invariants, and pre- and post-conditions. The required behaviour can also be specified directly in the model using an action language [4], making the PIM computationally complete. Consequently, tools can interpret the model directly or transform the model directly to program code.

MDA employs a set of tools for development, transformation, interpretation, verification, etc. of models and meta-models. These tools mainly take models as inputs and generate models as outputs, such as taking PIMs as input and generating PSMs as output. However, in spite of its potential MDA has not gained the deserved industry acceptance, mainly because of incomplete standards, vendor reluctant to make their MDA toolsets interoperable, and requiring high level of expertise for using tools and methods. Furthermore, MDA's predecessor, CORBA did not manage to be established as a widely used standard, which does not give OMG a good track record. This is of course something that could change in the near future as the above shortcomings are being addressed.

As for the potential of MDA for M&S some efforts have been done to use MDA concepts for supporting simulation reuse and interoperability ([7], [23], and [176]) with various degree of success. However, these efforts do not solve the issue of semantic composability of simulation models [144]. In [141] the authors suggest a combination of UML and DEVS as a viable approach for documentation as basis for supporting composability. This is also what we believe to be the right approach, and is similar to how we have addressed the issue of composability in this work.

3.4 Summary

The convergence of component based software engineering domain and component based simulation development domain provides opportunities for both communities to take advantage of the progresses made in the other community. In this chapter three approaches taken by the software engineering to address the issue of component based system development are presented and their relevance for component based simulation development is discussed. All these approaches aim at separating modelling aspects from implementation details, by providing formalisms for describing components and mechanisms for composition. Of the technologies presented above PACC, with its approach for reasoning about validity of compositions before implementation, is the one that come closest to our way of addressing component based simulation development. However, the technology that has mainly inspired our work is Semantic Web Services, presented in the next section.

Chapter 4

Semantic Web Service Composition

Previous chapter outlined some of the software engineering approaches that have the potential to support component based M&S. One additional technology which was not covered in that section is the Semantic Web technology, including the Semantic Web Services technology. This technology has been dedicated a whole chapter, due to the special part that this technology has played in this work. Here an overview of the technology, particularly Semantic Web Services Composition and its potential for component based M&S, and the research that have inspired our work, is presented.

4.1 Semantic web

The Semantic Web is an extension of the current Web technology which brings structure to the content of Web pages. This is achieved through a set of techniques by making formal description of the contents, which is understandable and processable by machines. The reason behind the initiative is to improve the ways on how to find, use and reason about the information found on the Internet [164]. Originally the available information on the Web was only presented in a human-readable form, making it almost impossible for computers to understand and interpret the information. Using the Semantic Web technologies the information is given well-defined meaning, better enabling computers and people to cooperate. It should be noted however that the Semantic Web is not about teaching machines to process natural language. Rather it structures everything in languages, understandable by machines. Moreover, the aim of the Semantic Web is not to build Artificial Intelligence (AI), but collecting data in a useful way, like a large database.

Two aspects have been pointed as characterizing the Semantic Web. Firstly, the Semantic Web is about common formats for integration and combination of data drawn from diverse sources, as opposed to the original Web which was mainly concentrated on the interchange of documents. Second, it is about language for recording how the data relates to real world objects [126].

Tim Berners-Lee the director of the W3C [178] originally expressed the vision of the semantic web as follows:

"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The intelligent agents people have touted for ages will finally materialize." [164]

Consider the following example, today if a user wants to list the prices of airplane tickets to a specific destination during a certain time period and combine that with information on available hotels, she/he has to use search engines that are individually tailored to every website being searched. Using the Semantic Web a computer can be instructed to make the list, since the Semantic Web provides common standards such as RDF (Resource Description Framework), OWL (Web Ontology Language) for websites to publish the relevant information in a more readily machine-processable and integratable form.

RDF and OWL are two examples of languages and constructs have been specified, to aid in the construction of the Semantic Web. Other techniques used in the Semantic Web context include ideas from the Artificial Intelligence-field and can be used not only for the Web but for other application areas as well. A brief explanation of the main concepts in the Semantic Web initiative is included below. More information about these concepts can be obtained from the Semantic Web homepage [178].

4.1.1 Ontologies

Internet is a complex and dynamic environment and if computers are to exchange information and reason about he contents of the Web, then it is essential that the semantic information of the contents are defined explicitly, such that there is an agreement between involved parties. Ontology is one of the basic concepts of the Semantic Web, which aims at providing common and unambiguous description of terms used in the Web in order to facilitate search and reasoning about the available information. According to Tom Gruber:

"An ontology is a specification of a conceptualization" [166]

The word ontology has a long history within philosophy where it describes a systematic account for Existence.

"[Ontology is] the study of being in so far as this is shared in common by all entities, both material and immaterial. It deals with the most general properties of beings in all their different varieties" [73]

Hence, ontology defines the terms used to construct a description of reality in its most general sense and how the terms are related. Recently the ontology has been coined as a term within Computer Science and in particularly AI [162]. In a computer science context, ontology is an establishment of joint terminology/frame of reference between entities that interact, and is used as such to agree upon what is communicated between the entities. Hence, the use of Ontologies can create a shared understanding of entities and events. For example, ontology can define that there is a concept of an object called Vehicle, and that a Vehicle can carry passengers. It can then reason around this object in some formal way, to convey that for example a Car is a Vehicle. These terms and properties can be declared in a formal system and using logical inference it can be concluded that a Car can carry passengers. This is from the fact that a Car is a Vehicle and Vehicles can carry passengers. The point of declaring this type of ontology is so that if two entities (for example two computers that try to mimic intelligence) have agreed upon using this ontology, and one of the entities can mention the word Car, then the other entity knows that this Car they are talking about is a Vehicle and nothing else.

Ontologies exist at several abstraction levels, which are presented in the literature with some slight variations. However, the three main levels are Reference (Upper) ontology, Core ontology, and Domain Ontology. Upper-level ontology captures mostly concepts that are basic for the human understanding of the world, such as time and space. It represents a framework using which the building blocks of reality are described, separated from any specific domain. Hence, concepts defined in the upper ontology should be of use in several domains (ideally an arbitrary domain). The upper ontology provides a knowledge base which can be used for building more specialized ontologies, such as mid and domain ontologies. Hence, knowledge and semantics already specified in the upper ontology is reused supporting interoperability between different specialized ontologies. The Core Ontology aims at capturing concepts and their relations in a field of practice. The Domain Ontology defines the specific reality and specialises the Core Ontology. DeMO is an example of a Core Ontology for modelling and simulation [137].

In the Semantic Web initiative, Ontologies are described in the RDF and OWL languages, explained briefly below.

4.1.2 RDF – Resource Description Framework

The Resource Description Framework (RDF) is a general-purpose language for representing information on the Web. It provides a data model and a syntax to facilitate utilization and exchange of the data model. The data model together with RDF Schema - which is a vocabulary description language used to describe properties and classes of RDF resource and its hierarchies - provides a means for describing entities, classes and their properties [51]. RDF is a World Wide Web Consortium (W3C) Recommendation, written in XML and designed to be read and understood by computers. It describes resources in the form subject-predicate-object expressions, which is called a triplet. The subject is the resource itself, and the predicate presents a relationship between the resource and the object. For instance, the expression "FOI is under the Department of Defence in Sweden" is represented in RDF as triple, with "FOI" as the subject, "the Department of Defence in Sweden" as object and "is under" as predicate.

The following example shows how with the help of RDF (written in XML) one can describe that "MyCar" is of type "Car" and that it has a "silver" colour.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                                   xmlns:ns="http://farshad.moradi.se/#">
   <ns:Car rdf:about="http://farshad.moradi.se/#MyCar">
      <ns:hasColour rdf:resource="http://farshad.moradi.se/#silver" />
   </ns:Car>
</rdf:RDF>
```

A full description of the RDF and RDF Schema can be found in [177].

4.1.3 OWL – Web Ontology Language

The Web Ontology Language (OWL) is a markup-language based on XML and standardized also by the W3C that formalizes a syntax in which an ontology can be defined. OWL was developed from RDF, and has been designed such that every RDF document is also a valid OWL document.

Besides this, OWL includes a large number of additional syntax constructs that allow for a greater freedom of use than RDF allows. For instance, OWL facilitates building large ontologies, which has been a problem before due to lack of clear definitions [178]. This is done by having an explicit logical basis, based on Description Logics (DL). DLs are a family of logics that are decidable fragments of the first-order logic [31]. The semantics of OWL are given through translation to a particular DL. Hence, OWL is not only a syntax for describing and exchanging ontologies, it also has a formally defined semantics for providing the meaning. Furthermore, as for DLs there are complete and terminating reasoners for OWL. A reasoner (reasoning engine, rules engine, or semantic inference engine) is a system which derives consequences of the knowledge presented in an ontology.

The following example shows a class "Car" with the property "colour", defined in OWL.

```
<rdfs:Class rdf:ID="Car">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#colour"/>
            <owl:allValuesFrom
            rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</rdfs:Class>
```

An OWL document, also called a knowledge base, can be queried and modified. Querying can be done e. g. using SPARQL, which is an RDF query language standardized by W3C. SPARQL is a recursive acronym and stands for "SPARQL Protocol and RDF Query Language" [28]. Modification of the document can be done by adding and retracting concepts, roles, and assertions. A restricted mechanism to add assertions are rules. Rules are an extension of the logical core formalism, which can still be interpreted logically [70].

OWL comes in three 'flavours' with different levels of expressiveness: OWL Full, OWL DL (Description Logic) and OWL Lite. OWL Lite was intended to only support classification hierarchy and simple constraints, whilst OWL DL provides the maximum expressiveness possible while retaining computational completeness, decidability, and the availability of practical reasoning algorithms. OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF-S. OWL-Lite is not widely used mainly because it is limited and development of OWL-Lite tools has thus proven almost as difficult as development of tools for OWL-DL. And as for OWL Full, it is unlikely that any reasoner will be able to support complete reasoning.

For a more information on OWL, please see the W3C websites [178].

4.1.4 Inference

Another important concept and driving principle of the Semantic Web is inference. Basically inference is about being able to derive new data from already known data. As mentioned earlier, the semantic web is intended to provide "machine-understandable" web resources. Exploitation of these resources requires the ability to process and inference ontologies written for instance as OWL documents or OWL Knowledge Bases.

Inference or reasoning can be used for checking the consistency of an OWL ontology and a set of data descriptions, finding implicit subclass relationships induced by the declaration in the ontology or finding synonyms for resources (either classes or instance names), etc [162]. Querying is also a form of inference, i.e. being able to infer some search results from a mass of data. Hence inference facilitates creating Semantic Web applications quite easily.

To demonstrate the potential of inference consider the case when you have an executable program, e.g. a simulation, with specific runtime requirements, software and hardware. You also have access to a network of heterogeneous executions resources, i.e. computers, which can be used for running your simulation. One way to find out which computer is suitable for running your simulation is to go through all computers and find the one that matches the runtime requirements of your simulation. This is of course surmountable, but this situation gets more difficult as the number of simulations and computers grow. One way to solve this problem is to design a software agent that can do the job for you. But this requires that the characteristics of the computers and the runtime requirements of the simulations are presented in a way that is understandable by your software agent e.g. using a formal language such as OWL or RDF. Having this information at hand the software agent must match the simulation against the computers. This is where inference can help you.

Inference can also help in merging databases, e.g. by recording in OWL somewhere that "User" in one database is equivalent to "Consumer" in another database, and then putting all this information together and getting a processor to handle it. And indeed, today this is already possible with available Semantic Web tools.

4.2 Web Services

The introduction of Web Services (WS) extends the current Web from a distributed source of information to a distributed source of services. This is a new paradigm, which supports development of complex Web based applications. WSs are platform and language independent composable software components, which are designed to provide interoperability between diverse applications. Hence, enabling users to access business functionalities and support heterogeneous enterprise application integration.

According to IBM [62], Web Services are self-contained, modular applications, accessible via the Web through open standard languages, which provide a set of functionalities to businesses or individuals.

This definition presents the basics, but is somewhat unclear. A better definition is provided by the World Wide Web consortium (W3C):

"A Web Service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols." [179]

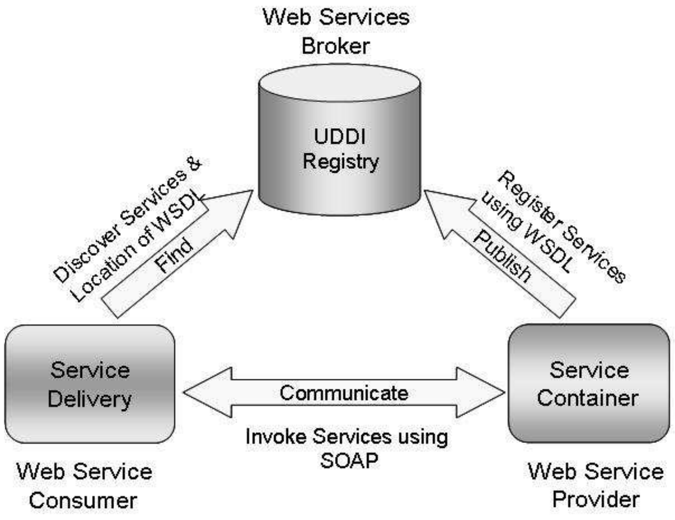


Figure 9 Web Service roles and operations

A WS is described using a service description, which is a standard, formal XML notation. The service description provides all the necessary information to interact with the WS, including message formats, location and transport protocols. The descriptions are expressed in Web Service Description Language (WSDL), separating service implementation and specification. In the context of WS there are three distinct roles: a WS provider, a WS consumer, and WS broker. A WS and its description is created by a service provider and then published with a service registry (the broker) based on a standard called the Universal Description, Discovery, and Integration (UDDI) specification. A service requester (the consumer) may find a published service via the UDDI interface. Since the service implementation and specification are decoupled the requester does not need to deal with underlying implementation issues. Hence, all the requester has to know is how to address a service, what parameters to send and what response to expect. The UDDI registry provides the service requester with a WSDL service description and a URL (Uniform Resource Locator) pointing to the service itself.

The service requester may then use this information to directly bind to the service, i.e. communicating with the provider, and invoke it, as illustrated in figure 9.

According to [105] the WS architecture is a three-layer stack comprising, basic services, composite services and managed services, illustrated in figure 10. The first layer, basic services includes the basic operations described above, i.e. publication, discovery, selection and binding, realized with WSDL, SOAP and UDDI. The second layer, composite services, encompasses necessary roles and functionality for composition of services. Resulting composite services may be used as applications by service consumers or as components in future compositions. The issues handled in this layer are service cooperation, defining coordination, monitoring, quality of service, etc. Service composition is one of the important aspects of WS which will be discussed in the next section.

The last layer, managed services, deals with the critical issue of managing services, administration and integration across a diverse, distributed environment. This layer provides service level agreements mechanisms, operation assurance, and certification and reputation systems.

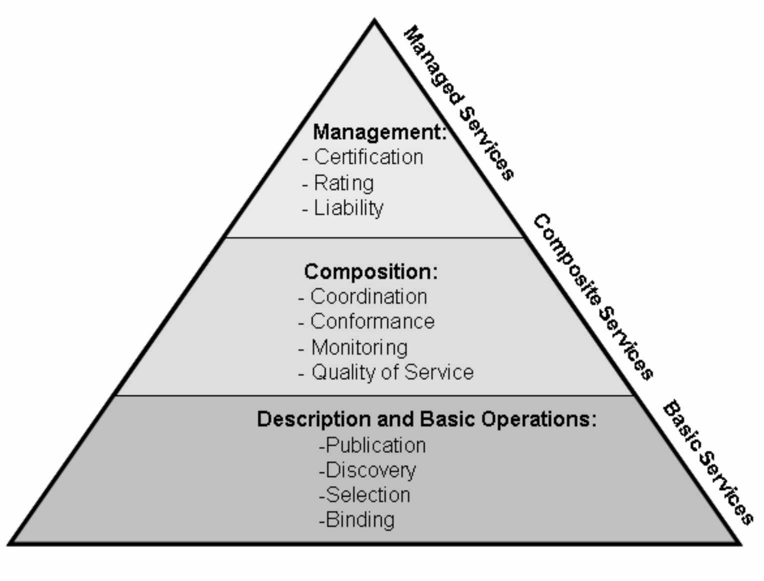


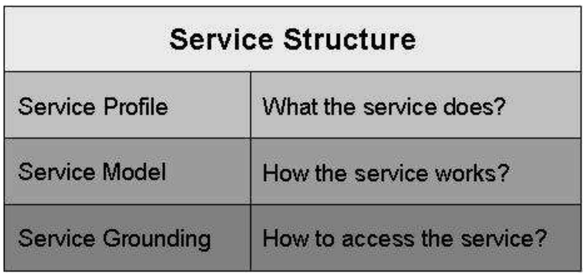
Figure 10 Web Services Architecture Stack

The next section gives a presentation of the Web Ontology Language for Services (OWL-S), which is one of the languages used for describing WSs.

4.2.1 OWL-S

OWL-S is based on Web Ontology Language and aims at establishing a framework for describing Web Services in the context of the Semantic Web. OWL-S is an extension of the DARPA Agent Meta Language for Services (DAML-S). It has been developed to provide support for discovery, composition, invocation and interoperation of Services. OWL-S consists of three parts: the service *profile*: "for advertising and discovering a service", the service *process model*: "for detailed description of a service operation", and finally *grounding*: "for describing how to interoperate with a service", figure 11.

There are two constraints regarding OWL-S. First, a *service* can be described by at most one *service model,* and second a *grounding* must be associated with exactly one service. There is no restriction for service *profile* and service *grounding*; in fact it is very useful sometimes to have multiple service *profiles* and/or service *groundings* [131].



Service Structure	
Service Profile	What the service does?
Service Model	How the service works?
Service Grounding	How to access the service?

Figure 11 OWL-S structure

4.3 Web Service Composition

WS can either be invoked thus used as they are, or be composed in different combinations to provide new capabilities and services to different users. The process of developing composed services includes discovery of appropriate WSs and composition of those to meet desired goals. The challenge is that even though WSs are individual components usually implemented at different locations, and executed in different contexts, when composed they need to communicate to yield the desired behaviour, which is not trivial. This problem gets even harder when we think about automatic or semi-automatic approaches and techniques for WS discovery and composition [60].

Today there are various techniques and approaches presented by different researchers for addressing the above issue, such as, template-based [32], interface-based [60], logic-based [154], ontology-driven [182], quality-driven [69], automata-based [163] and Petri net-based techniques [153], [183]. These approaches are either static or dynamic [151]. Dynamic approaches are model-driven as opposed to static approaches which are hard coded.

The above techniques can also be grouped as industrial-based or based on Semantic Web solutions. In the industrial solutions the interaction between different WSs is done using languages BPEL4WS (Business Process Execution Language for Web Services) [61] or WSCL (Web Services Conversation Language) [180]. The process for composition of WSs based on these languages involves specifying the role of each participating WS and the logical flow of messages between them.

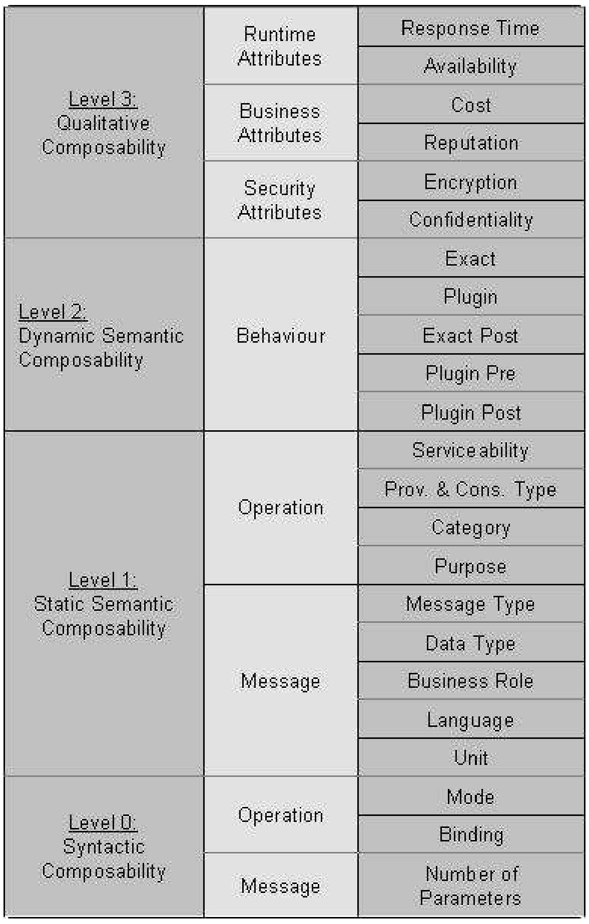
In the Semantic Web solution each WS is annotated based on its functionalities, location, domain, etc. using concepts, defined in an ontology, making them machine-readable. Hence enabling software agents to *understand* the purpose and capabilities of a WS, and apply logical inferencing techniques to compose WSs. The languages that are used for describing the WS are DAML-S (DARPA Agent Markup Language for Services) [21] and OWL-S (Web Ontology Language for Services) [131]. These WSs have also been referred to as Semantic Web Services.

The main goal of these techniques is to provide automatic ways to discover and reason about combination of services. They often apply software synthesis and composition methods to WS composition, which first of all requires a description of the services in formal logic or a formal language, such as OWL-S, a synthesis mechanism which automatically selects, adapts and composes services using inference, and a manager that invokes the services and transfers data between them. This is quite similar to the processes of selection and composition of simulation model components, and even though there are some basic differences between a simulation model component and WS (e.g. a simulation model component is stateful, while a WS is usually stateless), there are many lessons to be learned here. Hence, we have explored Semantic Web and Web Service (WS) technologies to improve development of methods for discovery and composition of simulation model components [36].

Two approaches to Semantic Web Service Composition that have inspired part of our work are briefly described below.

The first approach is a work done by Medjahed et al [9] introducing a multilevel composability model in which the composability of Semantic Web Services is checked in four levels: Syntax, Static Semantic, Dynamic Semantic and Qualitative level.

The authors introduce a set of rules for checking the composability at each level and specify what characteristics of a WS to be checked at that level. The composability check starts from the syntactic features, i.e. the lowest level, of services like mode and binding and finishes at quality of service aspects of services such as cost and availability, illustrated in figure 12. The semantic of services is divided into two categories: *Static Semantic*, i.e. semantics that is fixed and related to ontology, and *Dynamic Semantic*, i.e. semantics that depends on the execution conditions and varies during the runtime. Furthermore, two types of bindings, namely *Horizontal* and *Vertical* bindings are studied in this work and the influence of each binding type on the composability of services is described. The work also includes a composability degree based on the introduced composability levels.



Level 3: Qualitative Composability	Runtime Attributes	Response Time
		Availability
	Business Attributes	Cost
		Reputation
	Security Attributes	Encryption
		Confidentiality
Level 2: Dynamic Semantic Composability	Behaviour	Exact
		Plugin
		Exact Post
		Plugin Pre
		Plugin Post
Level 1: Static Semantic Composability	Operation	Service ability
		Prov. & Cons. Type
		Category
		Purpose
	Message	Message Type
		Data Type
		Business Role
		Language
		Unit
Level 0: Syntactic Composability	Operation	Mode
		Binding
	Message	Number of Parameters

Figure 12 Composability Stack

The second approach is a work presented by Arpinar and et al [59], which introduces a novel algorithm named "*Interface Matching Automatic (IMA) Composition*" for web service composition. The algorithm is based on user requirements comprising input parameters and constraints, and expected output parameters and constraints. In this approach web services are navigated to find a sequence starting from the user's input parameters and go forward by chaining services until they deliver the user's expected outputs. It is claimed that the algorithm has the capability of finding a composition, which offers the best quality of service, such as shortest execution time, and also the best matching of input and output parameters.

As a continuation of the work done with IMA, Arpinar and et al in [96] present a novel technique for discovering semantic relationship between different services via identifying relationship (similarity) between their pre and post conditions. This technique uses the IMA approach to build a network of services, and establishes the semantic relations between these services. Finally, based on these relations a semantic network of services with complimentary functions is constructed. The basic idea and assumption of this work is that functionality of service can be expressed by pre and post conditions. The authors claim that this technique can find semantic relation between web services despite syntactic mismatch. It is also mentioned that since there is no strong consensus for representing pre and post conditions in a certain specific language, pre and post conditions are expressed in RDF (Resource Description Framework) triplets. Degree of similarity between two conditions can be found through comparing similarities between triplets (RDFs) of these conditions.

4.4 Summary

In this chapter the concepts of ontology, inference, and Web Services Composition are introduced. Ontologies provide distinct and clear definitions of terms, concepts and their relations for any domain of interest. Formal and structured languages help software agents (computer programs) to interpret and understand statements, and communicate with each other in an unambiguous manner. Combining these aspects with the power of logical reasoning provide an excellent candidate for facilitating automatic discovery and composition of Web Services, and in similar way for simulation model components. In this work we have employed ontologies to extend the BOM standard with semantic annotations. Inferencing through inference engines have been used to reason about composability of BOMs, and Web Services Composition techniques have inspired development of our rule based method for BOM composition that has been briefly presented in chapter 7 of this thesis. A more comprehensive presentation is given in part II, paper 9.

Chapter 5

Related technologies

In this chapter some of the technologies, which have been studied and used in the scope of this work are introduced. The main idea is not to give a comprehensive presentation of these technologies, rather some background information to facilitate better understanding of the thesis work.

5.1 DEVS

Discrete Event System Specification (DEVS) introduced by [10], [11] provides a formalism for models and simulations. It is part of a family of formalisms developed by Bernard Zeigler, Herbert Prähofer and Tag Gon Kim, which includes other formalisms such as Differential Equation System Specification (DESS) and Discrete Time System Specification (DTSS) (as a subsystem of DEVS) [10]. DEVS exists in two variations, Classic DEVS and Parallel DEVS with the main difference of handling of concurrent events. DEVS is based on dynamic system theory which aims at describing the structure and behaviour of systems. It has well-defined concepts for supporting hierarchical and modular model construction, coupling of components, and an object-oriented substrate supporting repository reuse. Advantages of the DEVS methodology for model development include well-defined separation of concerns supporting distinct M&S layers that can be independently verified and reused in later combinations with minimal re-verification.

DEVS is not a simulation language and model implementation has to be done using an executable language. Hence, DEVS models can be specified in a formal way but implemented in different languages. Furthermore, to support composability between DEVS components and execution of those components, all DEVS programs have to comply with an interface specification. Today there are different frameworks written in various languages, such as Java and C++ available for development of DEVS programs.

DEVS has not been directly used in this thesis, but its ideas have inspired the work done. In the next section the basic DEVS formalism is described and presented.

5.1.1 DEVS structure

In order to facilitate hierarchical model construction DEVS supports two kinds of models, a basic, called atomic, and a coupled model, which acts as a container for basic models [11].

An atomic model M in DEVS is defined as:

$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau)$

Where

X is the set of input values

S is the set of states

Y is the set of output values

$\delta_{int} : S \rightarrow S$ is the internal transition function

$\delta_{ext} : Q \times S \rightarrow S$ is the external transition function, where

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq \tau(s)\}$ is the total state set

e is the time elapsed since last transition

$\lambda : S \rightarrow Y$ is the output function

$\tau : S \rightarrow \mathbb{R}^{+}_{0,\infty}$ is the *time advance* function

According to the above definition, in the context of DEVS a component is described having inputs *X*, outputs *Y*, and a set of states *S*. The state of a component changes through invocation of either an internal transition function δ_{int} or an external transition function δ_{ext} [10]. δ_{int} is invoked every $\tau(s)$ if no externally induced transition occurs.

Furthermore, in the time advance function $\tau : S \rightarrow \mathbb{R}^{+}_{0,\infty}$, 0 and ∞ have the following meanings. In the case of 0 the stay in the state *s* is too short for an external event to intervene. The state *s* is regarded as a transitory state. In the case of ∞, the system will stay in *s* forever unless an external event interrupts it.

Finally, when an internal event has executed, i.e. $e = \tau(s)$, an output is generated via the output function $\lambda(s)$ and the state is changed according to $\delta_{int}(s)$.

In the above definition the state *s* represents a *phase* in which the model component resides and can be changed according to either of the state transition functions. However, the total state of the component is a combination of both *s* and other numerical values defining the component. Moreover, the atomic DEVS model can be extended to facilitate the modelling process using ports, where each port can be assigned a value. Hence, in the extended model the input and output values are replaced by (*p, v*) which represents a port-value tuple, as shown below.

$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, \tau)$

Where

$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values

$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values

S is a set of *sequential* states

$\delta_{int} : S \rightarrow S$ is the internal transition function

$\delta_{ext} : Q \times S \rightarrow S$ is the external transition function, where

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq \tau(s)\}$ is the total state set

e is the time elapsed since last transition

$\lambda : S \rightarrow Y$ is the output function

$\tau : S \rightarrow \mathbb{R}^{+}_{0,\infty}$ is the *time advance* function

As explained earlier, the DEVS formalism supports hierarchical and modular model construction, via coupling of models. The coupled models can in turn be used as components in new combinations. In order to facilitate this, DEVS defines a coupled model as:

$M = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select)$

Where

$X = \{(p, v) \mid p \in InPorts, v \in X_p\}$ is the set of input ports and values

$Y = \{(p, v) \mid p \in OutPorts, v \in Y_p\}$ is the set of output ports and values

D is the set of component names

M_d is a DEVS model with

$X_d = \{(p, v) \mid p \in InPorts_d, v \in X_p\}$

$Y_d = \{(p, v) \mid p \in OutPorts_d, v \in Y_p\}$

EIC is the set of input port couplings

$EIC \subseteq \{((N, ip_N), (d, ip_d)) \mid ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$

EOC is the set of output port couplings

$EOC \subseteq \{((d, op_d), (N, op_N)) \mid op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$

IC is the set of internal couplings

$IC \subseteq \{((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in OutPorts_a, ip_b \in InPorts_b\}$

Select is the tie-break function

According to this definition a coupled model comprises a set of input and output ports, a set of component names *D*, a set of DEVS components *Md*, input and output port couplings, *EIC* and *EOC*, a set of internal couplings *IC* connecting internal components with each other, and a so called *tie-break* function *Select*. The input and output ports can either be connected to corresponding ports of other components within the coupled model or externally connected, using another coupling model, to other components or coupled models. The *tie-break* function decides which component to proceed when two or more components have internal transitions scheduled at the same time.

Figure 13 illustrates a coupled model *CM* as a combination of models *M1* and *M2*. In this figure IP_{CM} and OP_{CM} stand for input and output ports of CM, IP_{M1} and OP_{M1} stands for input and output ports of M1, and IP_{M2} and OP_{M2} stands for input and output ports of M2.

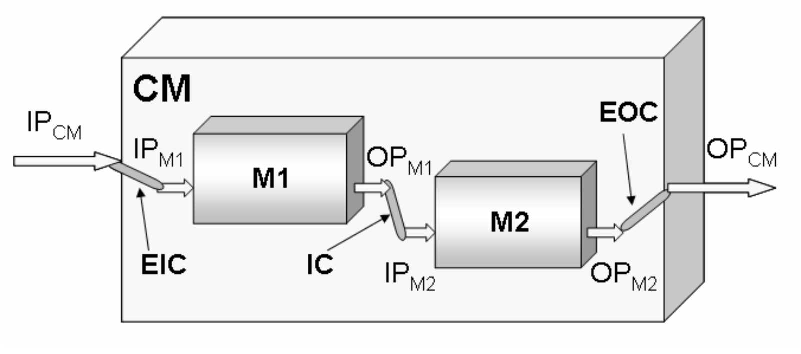


Figure 13 Composed DEVS Model

The descriptions above show that DEVS has a sound set theoretic approach and is able to describe the structure of models in a formal way. It has the potential to be included in a meta-model of a component, but only for structural description since it does not include the behaviour (the transition function specifications) [118]. The formalism facilitates reasoning about compositions and supports hierarchical model development, through coupling of components. However, such reasoning requires tools and theories for analysing the compositions, which is not available at the moment. Furthermore, there are no methods for analysing the behaviour of models and it is not possible to specify context dependencies other than through input and output. Hence, DEVS mainly supports syntactic composability, and as long as there are no methods available for reasoning about the validity of compositons, semantic composability is not an issue.

5.2 SRML

The Simulation Reference Markup Language (SRML) is a formal language for describing simulation models. It is based on XML and has been developed by the World Wide Web Consortium (W3C) with support from the Boeing Company. According to the W3C [181]:

"SRML is an XML application that can describe the behaviour for distributed simulation models, and its runtime environment is a software that is capable of executing those models."

The main objective of introducing the SRML language is to be a flexible reference standard for representing simulations with enough expressive power to model most anything for the purpose of simulation. The runtime environment serves as a *simulation engine* for running simulations described in SRML. The example below shows how a simple SRML document is structured.

```
<Simulation>
            <Script Type='text/javascript'>
            ...
            </Script>
            <ItemClass Name='Aircraft'>
                        <Aircraft type="Fighter">
                                    <Script> ... </Script>
                        </Aircraft>
            </ItemClass>
            <Aircraft Quantity='2'/>
</Simulation>
```

As any other XML-based language SRML uses a set of tags to create structured representations. The <Simulation></Simulation> tags encapsulate an entire Simulation. The first part of the document is a <Script></Script> tag-set that contains the main script of the simulation. SRML does not define the language in which this script is written in, and can expect JavaScript, VBScript or any other script-language. The default script language however, is JavaScript. The following tag-set is a number of <ItemClass></ItemClass> tags, which define the different types of items that are present in the simulation. In the above example an Aircraft is defined. Within the tags you may also define a typical instance of the "ItemClass", defining the properties of the item. The tags can also contain a script describing the functionality of the item defined. An interesting attribute that can be added in an <ItemClass>-tag is the SourceAttribute that can be used to include outside logic and or definitions. The last part of the document, after the <ItemClass>-tags defines a set of instances of items that is being used in the simulation. In our example above we have defined two aircraft. In addition to these simple constructs, the SRML document contains Events that can be used inside the script-tags, and a few other constructs that increase the detail in a simulation.

The SRML Product Development Group (PDG), is one of the PDGs created by the Simulation Interoperability Standardization Organization (SISO), working on standardization of a simulation markup language and corresponding simulation engine specification based on SRML [160]. According to the group the language specification will include; SRML concept of operations including engine description, XML tag set for SRML with descriptive text, and SRML User Guide. The engine specification comprises an Engine object model and an Application Program Interface (API) reference. For more information on SRML, see the W3C SRML website [181].

In this thesis SRML has been used for representing simulation scenarios in a formal way. The main reason has been to facilitate automatic parsing of the contents of the scenario and identifying contained entities and events for the purpose of component discovery. However, SRML includes large number of features, some of which are not applicable in the scope of this work. Furthermore, we do not wish to burden a modeller with designing complex high-level definitions of simulations. Hence, in this work only a subset of the language has been utilised. In the next section a brief introduction of this subset, referred to a SRML-light is presented.

5.2.1 SRML-light

As explained above we have specified a subset of SRML called SRML-Light [34]. SRML-Light includes only the features of SRML that are relevant for utilisation in the model discovery and composition process that is developed in this work. (The process is explained in detail in chapter 7) In the process there are mainly two major needs for an SRML definition, namely describing entities and connection between them. SRML provides this type of descriptions via the *ItemClass-tags* and the *Script-tags* respectively [181].

The *ItemClass-tag* is used in SRML to describe entities in a simulation. This is exactly what is needed for the purpose of *component discovery*. Thus, we found it suitable to adapt the ItemClass-tag in order to facilitate this activity. The ItemClass-tag in SRML can contain mote information than is needed for purpose of this work and hence, we made the following changes to limit its scope in SRML-Light. First of all, the ItemClasstag allows hierarchical structures of entities, describing inheritance of properties and logic. In SRML-Light the ItemClass is limited to being a flat structure without superclasses. If these types of relations are important to define, it will be present in the accompanying Ontology. Secondly, the ItemClass-tag can contain scripts that describe the internal logic of the item that is being defined. Although this is highly useful in a simulation, such logic is intended to be stored in the components (in this case BOM), and would be redundant in the SRML-Light document. Therefore we have left aside the Script-tag in each ItemClass, with the exception explained below. Left in the SRML-Light format are the ItemClass-tag with a name, the possibility to specify an arbitrary number of properties for the ItemClass, and an optional Source-attribute. The Source attribute of an ItemClass is interesting to keep, as it provides the possibility for a developer to name a specific BOM component that is known to fit for this component slot. This interesting since it could reduce composition time considerably and provide for more flexibility in design.

One could argue that the internal logic of a component can be useful in locating a BOM that includes such logic. Hence, it should be considered in a future version of the SRML-light to include this information, as long as it does not increase time of developing the high-level document and provide redundant information.

The second type of tag that was found interesting for this work was the *Script-tag*. Script-tag is included in the root of an SRML document as the starting point of a simulation to describe how the logical execution of the simulation takes place. As explained earlier, the Script-tag is defined to be included as a JavaScript of VBScript (or any other script-language), and for this reason a script parser have to be developed. There are a number of projects working on including JavaScript support for Java, such as Mozilla Rhino [129] or Jakarta Bean Scripting Framework [84], which could provide good libraries to use for this type of parsing. However, in SRML Light parsing and reasoning with a full JavaScript is not considered, and only the pre-defined events that are included in SRML to represent actions between components are being used. SRML defines four types of events that could be used to communicate between items, namely SendEvent, PostEvent, ScheduleEvent and BroadcastEvent. All these events are included in the Script-tags and could be used to represent inter-component communication. In SRML-Light, these events are therefore defined as the means to convey communication between components. Each ItemClass can contain a Script-tag that defines how it communicates with other components.

The PostEvent, SendEvent and ScheduleEvent are all specified within the Script-tag of each component, with a target component, event to call and a number of parameters. They can therefore directly represent an action between two components. In this work ScheduleEvent was not included, as specific timing of events is hard to translate into events in the conceptual model of BOMs. The BroadcastEvent is not straight forward to implement, as it has no explicit target. For this reason, the EventSink-tag specified by SRML within the ItemClass was included, which helps to specify that an item/entity will be listening to certain broadcasted events. It should be noted that using only these built-in events is a major limitation of use of the Script-tag, and does of course limit the expressiveness using a script-language. In a future this could be developed further as to make use of the full modelling capabilities that a script-language can provide in order to properly express inter-component communication. However, these limitations are quite justified with respect to the purpose of deployment of SRML-light in this work.

5.3 XVCL

The XML-based Variant Configuration Language (XVCL) is a general purpose mark-up language designed from the Frame technology ideas of Basset [138], with the aim of enhancing reuse in software development and supporting management of changes in software evolution. It is developed at the National University of Singapore and available as open source [184]. XVCL was initially intended to be used in a product-line fashion where similar components were used in a line of products, and where reuse of these components would reduce the development effort. However, it is today applicable wherever reuse and maintainability is of essence. XVCL is a complement to existing methodologies such as Object Oriented methodology, and not a replacement. It is software and application domain independent, thus useful for both software and non-software domains.

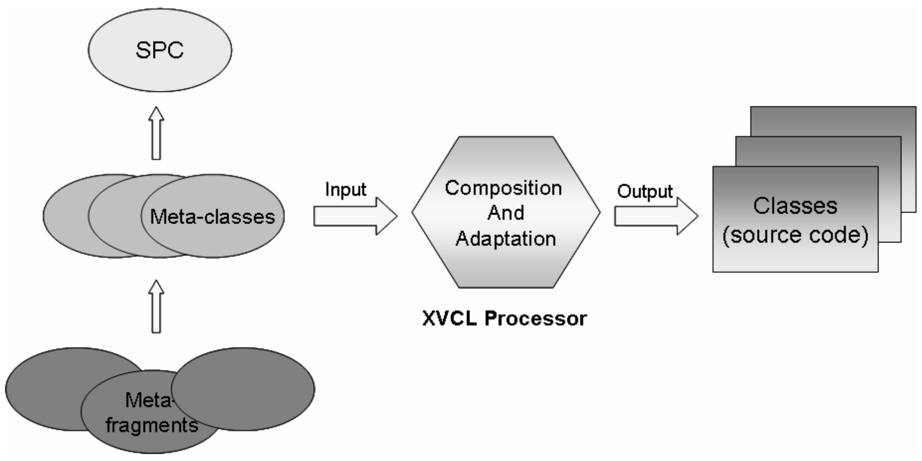


Figure 14 Constructing classes (source code) using XVCL

The idea of XVCL is to identify parts of the software products in the product-line where code is identical or very similar. These parts would then be condensed into a single XVCL *meta-component structure* that would be easier to maintain and reuse [184]. Hence, instead of maintaining several versions of a component (one for each product with small alterations that follows a generic design pattern), developers would only maintain a single XVCL meta-component structure. This meta-component structure could then be used to produce source code and reused in new applications when the same design patterns emerge. This is accomplished via employment of a XVCL *processor*, which traverses the meta-component structure, parsing the XVCL commands and producing the actual source code. The meta-component contains *commands* and *instructions*, which can be interpreted and processed by the processor in order to produce the code. Figure 14 illustrates the process of converting a meta-component into an executable component, based on a given specification.

The meta-component structures are sorted in a tree hierarchy consisting of number of *x-frames*, as can be seen in the left side of the figure. The root of this tree is called specification frame (SPC), which contains the most context sensitive or context aware information about how the system should be built. The SPC may include framed information on which sub frames that are to be included in the generation of the system.

XVCL has been studied in this work for the purpose of code generation and development of executable simulations (or code skeletons for simulations) using BOM and SOM information, as explained in chapter 7.

5.4 Software agents

Software agents are programs that can with some degree of autonomy perform tasks on behalf of a user or another program. The tasks are done based on the user's (or the program's) goals and desires. Agents have also the ability to learn and adapt their behaviour according to the user's needs in order to assist and support the users more efficiently [161].

The agent term is a relatively new research topic within AI field. According to [162] an agent is anything that can act using its effectors and perceives using its perceptors (i.e. sensors). Various definitions of software agents have been proposed by different authors [161]. Some consider any type of independent component, such as software, model, and individual, to be an agent [26]. Others mean that the behaviour of components must be adaptive in order for them to be considered agents [152]. There are also those that emphasize on the autonomous behaviour of the agents [130]. However, most of these definitions commonly include features such as, *Persistence, Autonomy*, *Social ability*, and *Reactivity*. The feature that has been considered as one of the basic features of an agent is its capability to make independent decisions, which requires active behaviour.

Agents have been used in M&S for providing answers to complex physical and social problems, e.g. for modelling intelligent and complex systems by subdividing them into subsystems that interact with each other [125]. Agent-based simulations are appropriate for domains characterized by high degree of localization, distribution and discrete decision making, such as crowd simulations [149].

Today majority of agent based systems consist of a single agent [94]. However, the need for systems with multiple communicating agents is increasing as the applications get more complex and the technology matures. These multi-agent systems (MAS) are a collection of autonomous agents that interact with each other and their environments to solve complex problems. A MAS can be regarded as a loosely coupled network of autonomous and perhaps heterogeneous agents, each solving a particular problem, that communicate with each other in order to solve problems that are beyond the individual capabilities or knowledge of each agent [27]. MASs are described to have a number of capabilities such as, solving large and complex problems (where information sources or expertise are distributed), providing interconnection and interoperability between legacy systems, solving problems that can naturally be regarded as a society of autonomous interacting agents, and enhancing performance [94]. The performance enhancement is done with regard to computational efficiency, reliability, extensibility, robustness, maintainability, flexibility, reuse, etc.

Agent based simulations (ABS) are special type of MAS, where the focus is on human social behaviour and individual decision-making, instead of developing artificial agents [15]. ABS is a collection of autonomous and decision-making agents that are able to assess their situations and make decisions according to a set of rules. An ABS can provide valuable information about the real-world system that it emulates, or reveal unanticipated behaviours. These behaviours emerge from the interaction between individual agents, which makes them difficult to understand and predict. An example is a riot situation, which results from the interactions between demonstrators practicing their civil rights expressing their opinions, police forces e.g. protecting properties, and malicious rioters taking advantage of the situation [149].

Another benefit of ABS is providing natural description of systems, which means that ABS can provide a realistic solution when simulating systems containing *behavioural* entities. For instance it is more natural to simulate how people in a building and fire-fighters are moving and acting during a conflagration, instead of setting up equations calculating the density of individuals. Furthermore, ABS is flexible and it is easy to add more agents or agent-based models to an existing ABS [15].

ABS has many areas of application such as flow simulations (e.g. traffic simulations [169]), Stock Market simulations [76], and Organisation simulations [106]. Recently, agent environments have also been used in development of symbiotic simulation frameworks [17], [52]. In the framework developed at Nanyang Technical University (NTU), symbiotic simulations are formed by combining agents, each capable of performing a basic functionality that is required for development of specific simulation scenarios [52].

5.4.1 Agent Framework

An agent framework is a development and runtime environment inspired by a specific agent architecture that is used to develop a variety of simple and complex agents and get populated by them. The agents live in that environment, they age (get mature), act and react to their environment and die. The framework can conceptually be taken as an API or programming library but it is more than just that. It is a runtime environment for complex behavioural software entities that are artificially intelligent. Agent frameworks permit creation of common-use components, which other agent-dependent components can easily make use of.

Next section presents an overview of the JACK Agent Framework [83], which has been used in this work. Different features and components of the system are being outlined and the Jack Framework workflow is being briefly described.

5.4.1.1 Jack Agent Framework

JACK Agent Framework is developed by the Agent Oriented Software Pty Ltd [83](AOS), and is an environment for development, execution and integration of multi-agent systems using a component-based approach. JACK is a third generation agent system, which according to the AOS has incorporated advances in Agent Research and Software Engineering. It provides the core architecture and infrastructure for building and running software agents in distributed applications. These software agents are called JACK Intelligent agents and model reasoning behaviour according to the theoretical Belief Desire Intention (BDI) architecture of distributed artificial intelligence (DAI) [17].

An agent model in the BDI architecture is event-driven with both reactive and proactive behaviour. It mimics simple human intelligence by having a view of the world i.e. Beliefs, goals it wishes to achieve i.e. Desires, and plans i.e. Intentions to act upon using its accumulated experience [107]. Thus, Jack Agents are autonomous software components that have explicit goals to achieve or events to handle, which are their desires. To describe how they should achieve these desires, agents are programmed with a set of plans. Each plan describes how to achieve a goal under varying circumstances. When executed, the agent pursues its given goals (desires), adopting the appropriate plans (intentions) according to its current set of data (beliefs) about the state of the world. For example when a goal or event arises the agent decides what course of action to take. If it already believes that the goal or event has been handled (as may happen when the agent is asked to do something that it believes has already been achieved), it does nothing. Otherwise, it looks through its set of plans to find those that meet the request and applicable to the situation. If it has any problem executing this plan, it looks for others that might apply and keeps cycling through its alternatives until it succeeds or all alternatives are exhausted.

The Jack Agent Framework consists of four different types of components: the *Jack Agent Language (JAL)*, the *Jack Agent Compiler*, the *Jack Agent Kernel*, and a set of *Jack Code Entities*. JAL is a Java based programming language that is used for describing a JACK agent oriented software system. A JAL program is pre-processed by the compiler to produce pure Java code, which can be compiled by the Java Virtual Machine. The Kernel provides JAL programs with agent oriented functionality and is the runtime engine for running those programs. Finally, Jack Agents need a set of code entities (code snippets), representing different aspects of the agents, to be able to run in a Jack environment, such as *JACK event*, *JACK plan*, *JACK capability*, *JACK view*, and *JACK belief set*.

5.4.1.2 JACK Framework workflow

A typical JACK project workflow starts by creating an agent given an agent *name*, a *plan* and an *event*. The agent can either post that *event*, or handle it. The agent uses the *plan* and the *event* handles the plan. This triangular relationship can be viewed in figure 15.

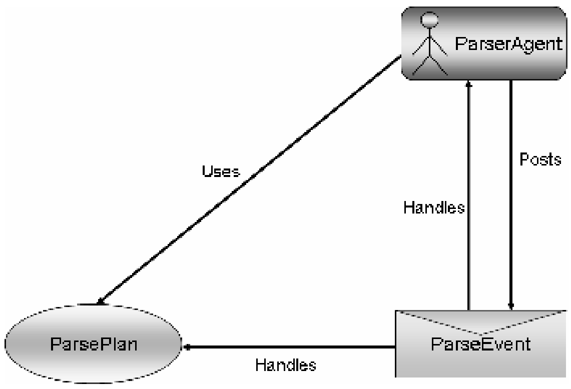


Figure 15 Jack Agent Workflow

Once the above components are in place, they are compiled as a capability component and added it in a capability library e.g., in the above example an agent is posting as well as handling a parse event. It also has a parse plan to execute. This parsing capability can be compiled in a "Parser" component.

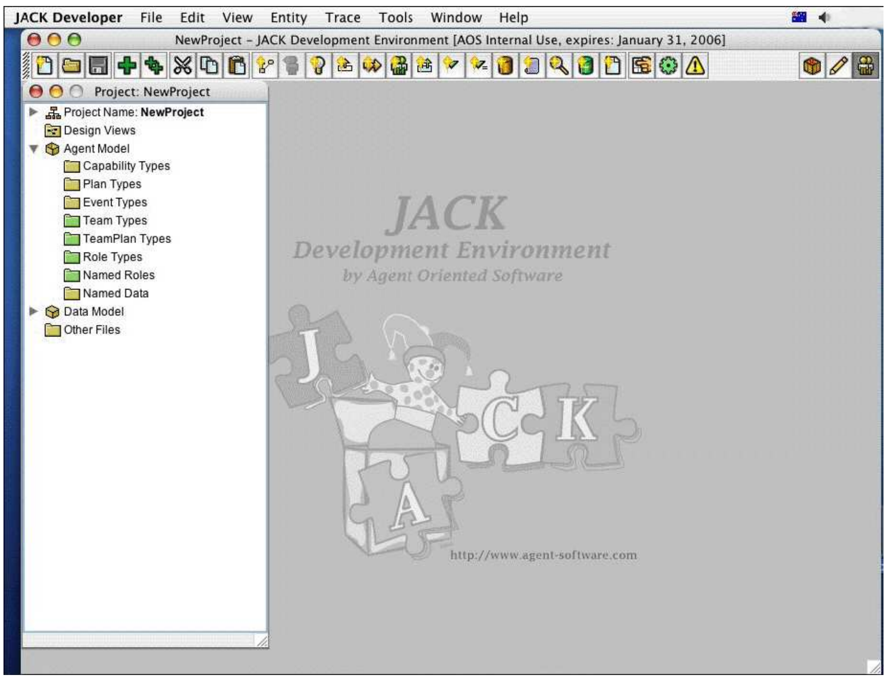


Figure 16 Jack Development Environment

In a multi-agent system, events are not usually posted and handled by the same agent. One agent posts an event and another agent (which has the capability) handles it. Once the event is handled, the agent sends/receives more events (event messages) to the requestor or to/from some other agent. This is how a process of co-operative co-ordination can be executed. This message handling can be done within the same machine or across the network. JACK kernel is capable of handling network communication.

5.4.1.3 JACK Development Environment (JDE)

The JACK Development Environment (JDE) is a cross-platform graphical editor suite for developing JACK agent and team based applications. The environment is written in Java and allows the definition of projects, aggregate agents and teams, and their component parts under these projects [83]. A screen shot of JDE environment is shown in the following figure.

5.5 Summary

In this chapter some of the technologies, which have been used or inspired this thesis, and have not been covered in other sections, are presented. The DEVS formalism provides support for reasoning about compositions and developing hierarchical model. Even though DEVS mainly supports syntactic composability, we believe that it is a well-founded and sound approach, which in combination with for instance UML can facilitate semantic composability [141].

SRML has been used for formal representation of simulation scenarios. SRML as a whole is a complete language for defining simulations and includes large number of features, some of which are not applicable in the scope of this work. Thus, we introduce a subset of the language called SRML-Light, which has been used in this thesis (part II, papers 7 and 9). The SRML-Light can be extended in the future to better make use of script-languages for expressing inter-component communications.

Even though the concept of automatic code generation has not been explored as much as we would have wanted to, we believe that XVCL is a great candidate for code generation and development of executable simulations from formal simulation model descriptions (part II, paper 7, and reference [115]).

Finally, multi-agent systems are the ideal candidates for realization of automatic or semi-automatic component based development processes that can support a modeller with identification and discovery of components as well as give feedback on feasibility of a composition (part II, paper 10).

Chapter 6

NetSim

As being pointed out in chapter 2.3, there are some issues that obstruct proper utilization of M&S, such as initial development costs, increasing system complexity, and user acceptance. Furthermore, simulation development is a multi-disciplinary process, which requires collaboration between experts from various domains, such as application domain experts, simulationists, end-users, VV&A agents (Validation, Verification and Accreditation), etc. Network-based Modelling and Simulation (NetSim) project was started in 2002 at the Swedish Defence Research Agency (FOI) funded by the Swedish Armed Forces [41], [113]. The general idea of the project, which ended in 2006, was to provide a collection of M&S services to address some of the above issues and hence, support design, development, execution and assessment of network-based/distributed simulation models. This general idea was the further revised and resulted in outlining a limited number of services with specific objectives. The common goal of these services was to facilitate utilisation of simulations in the Network Centric Defence (NCD), which is the latest paradigm adapted by the Swedish Defence.

Component-based simulation model development (CBMD) was one of these services [34], with the objective of identifying and developing methods and techniques for implementing a framework for CBMD. For this purpose we studied the BOMs standard [159], focusing on the BOM composability and BOM-based model development. Besides CBMD, other NetSim services support sharing of M&S resources, and collaborative simulation development and execution within and between organizations, which all aim at promoting increased use and reuse of simulation models which lead to increased quality of work in the M&S development process.

Figure 17 presents an overview of the service-oriented architecture of NetSim. The uppermost layer comprises various NetSim tools, dedicated to M&S-related tasks, for instance tools for composition of federations by a single user, or collaborative development of federations by a number of users. The NetSim tools derive their functionality from NetSim specific services, denoted CBMD (Component-Based Model Development), SDR (Semantic-based Distributed Repository), CC (Collaborative Core) and DRMS (Distributed Resource Management System). The DRMS provides computing capacity for reliable distributed execution of simulations. The CBMD provides a process and services for supporting component-based model development. CC provides services for collaborative work, whereas the SDR provides services for storage and look-up of available resources on a network. The NetSim specific services are based on various overlay network service technologies, such as Web Services, Grid Services and the HLA RTI. These are just examples of network technologies that could be deployed to achieve the goals of the NetSim environment.

6.1 Areas of research

The NetSim project was inspired from the latest developments within the fields of M&S and Service-Oriented Architecture (SOA). On one hand the advances in Web- and Internet-technologies are contributing to emergence of a new style of M&S architecture which is characterized by among others component-based development and execution methodology with loosely coupled simulation models. On the other hand the SOA is evolving in a sense that information, applications, tools, computers and other types of network resources are organized as services, such that they can be searched, discovered and utilised via networks. Examples of such architectures are the Network Centric Defence (NCD) and the Global Information Grid (GIG), which envisions a highly interconnected network of what can be generalized as, producers and consumers of information. The GIG is initiated by DoD (Department of Defense) and aims at:

"…enabling visibility, accessibility, sharing, and understanding of all information and services among all DoD users, as well as mission partners through well-defined interfaces" [47].

The GIG vision includes communications and computing systems and services, software (including applications), system data, security services, and other associated services. M&S has also been pointed out as one the important services to be utilized by the GIG. Hence, research has been conducted regarding methods and feasibility of employment of M&S services within the GIG [98].

However, to be able to integrate M&S into future defence systems, there is a need for developing M&S and M&S-related services within NCD. One of the objectives of the NetSim project has been facilitating employment of M&S within the NCD. The functions and services that have been designed and developed within the project are mainly:

- A framework for component-based simulation development
- A semantic-based distributed resource repository
- A distributed resource management system
- A collaborative core
- A security architecture

The project included four phases. The first phase was pre-study and researching different technologies and related ideas. During this phase we investigated different technologies and built small sample prototypes to test their feasibility [41]. The second phase of the project was design of a general service-oriented architecture for the NetSim environment [38]. Figure 17 depicts the multi-layer and modular architecture that was produced during this phase. The advantage of having such architecture was that it is scalable and could be utilized during a longer period of time in order to test and evaluate different current and future technologies, techniques and tools. New tools could be tested without needing to rebuild the whole system each time. This approach was also more flexible and would also give us the opportunity to employ different techniques at each layer, and not being limited to only one technique.

The third phase of our work was realization of the designed architecture and development of a prototype [39]. The realization was partly done by implementing each service and connecting it to the underlying layers and partly by development of an application that could join all services together. In order to make services interoperable a common information model was implemented. During the last phase of the project we tested and evaluated the whole environment using a set of scenarios and end-users assessments [40].

The following sections will give an overview of different NetSim services. These services can be deployed individually or in combination with each other. The main focus of this chapter however will be on the framework for component-based simulation model development and how the other services support this framework. These sections are mainly aimed at giving an introduction to different services, and for more detailed description of the work, interested readers are directed to different reports and papers that have been written by the project.

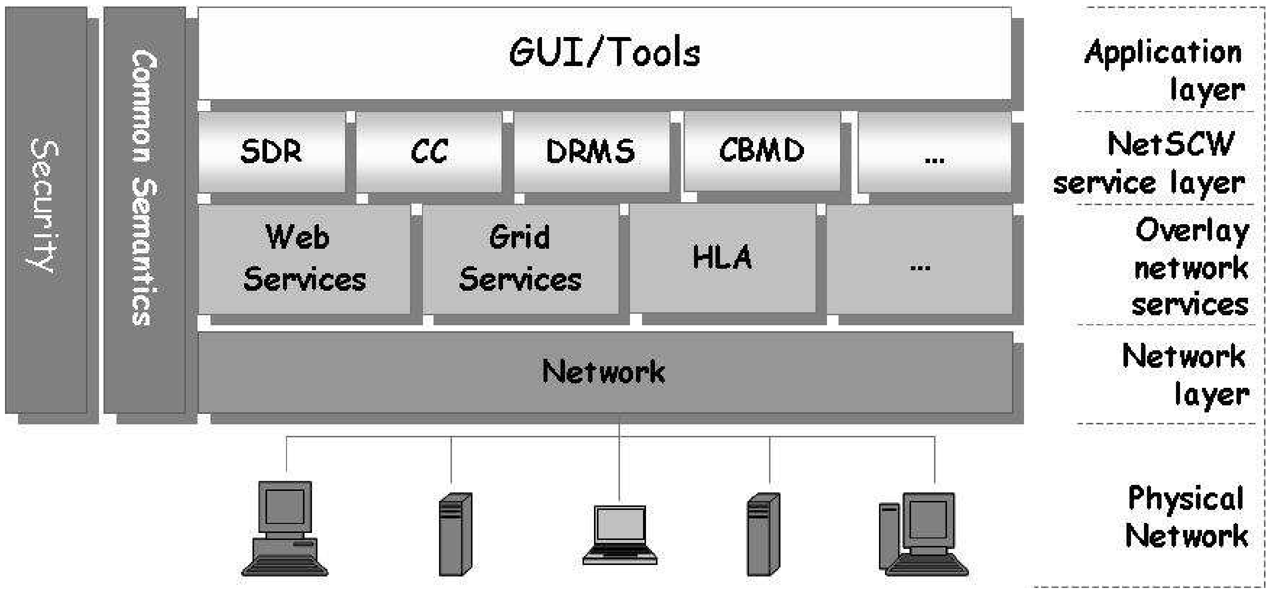


Figure 17 NetSim architecture

6.2 A framework for component-based model development

As mentioned earlier, component-based simulation model development (CBMD) is one of the services developed by the NetSim project. The objective here has been investigating and developing methods and techniques for implementing a framework for CBMD. The first step toward development of such a framework was identifying its constituents. The common denominator for all composition infrastructures both within M&S and software engineering is that they all besides the components themselves require a component library, composition and query tools and metadata standards [144]. Hence the NetSim CBMD framework consists of the following parts:

- A common data model for describing resources
- The data model, which is based on open standards, in order to promote reuse and interoperability, consists of a common terminology for describing simulation components and their relations. For this purpose we developed a general ontology for describing simulation components [108], [117]. Besides the ontology it contains structured meta-models and components specifications. In our case we utilize the BOM structure and OWL descriptions of BOM-based simulation components as described in chapter 7.
- Library of reusable components
- A semantic based repository, which can provide storage, search and retrieval capabilities is one of the main constitutes of the framework. The repository should be semantic based in order to provide more precise look-up capabilities based on the contents of the components in stead of the keywords, specifically when the number of the components in the repository is high. The repository is explained in more detail in section 6.3.
- Model composition environment
- Given a repository and a set of well-described and structured components, we need a process for search, discovery, matching, reasoning and composition of those components. This part is explained in more detail in chapter 7.

Furthermore, we have included the whole framework in a collaborative environment in order to promote validity, reliability, user-worth of the simulation models developed [81]. Finally the execution environment is employed for managing the invocation of the composite components and transferring data between the adherent components [110].

6.2.1 Different approaches for CBMD

Two different approaches for component-based simulation model development were investigated by the project. The first approach was inspired by the ideas of hierarchical model development and HLA, using HLA federates as components. The goal was to provide an environment for modellers to choose federates from a repository and compose them using a graphical composition tool. The object model descriptions of components (federates), i.e. SOMs together with ontology documents, were employed as basis for reasoning about the composability of those federates, and giving a modeller proper feedback [185]. The execution environment provided means for running the composed federations. We have developed a prototype environment based on this approach, which is functional and facilitates federate composition and federation development [41]. However, we concluded that even though SOMs provide enough information to reason about synthetic composability of composed federates, they have little to offer when it comes to reasoning about semantic validity of composition (federations). Furthermore, we noticed that federates in general are too large as components, and are usually developed for specific purposes (federations), which makes it difficult to reuse them in other combinations. Generally, a great deal of hands on work is required to adapt federates for reuse in federations other than those they have originally been developed to be part of. Deciding what has to be altered in order to adopt a federate is not trivial either, if one has not have access to the conceptual model describing the federate.

The second approach, which was perceived parallel to the first one, was slightly different. This effort was based on employment of formal description of models and automatic code generation. The first step of this track was taken in collaboration with National University of Singapore, NUS [115]. Simulation Object Models were used as a model for writing the formal descriptions of models. As for automatic code generation eXtensible Variant Configuration Language (XVCL) was used [184], explained in section 5.3.

We have developed an environment based on this approach, which proved the feasibility of the chosen path and XVCL as a technique for automatic code generation [115]. However, the conclusions regarding SOMs were identical to those from the first approach. Hence, as a continuation of this work, we decided to investigate feasibility of Base Object Model (BOM) as a basis for composition of simulation models [34], [36]. Chapter 7 gives a complete description of the work and the framework that has been developed.

6.3 SDR

Besides the components themselves one the main parts of the framework is a repository facility which provides management and storage of, and access to those components. Early in the project it was decided that a repository of such should be a distributed service which allows sharing and utilization of resources (e.g. simulation components) across organisation boundaries. We also decided that it should support advanced search and query, be robust, platform independent, decentralized, and scalable. Having these in mind we investigated available related ideas and platforms that could meet our requirements. The result is a hybrid called the Semantic Distributed Repository (SDR) [109].

The aim of the SDR is sharing, exchanging, utilization and modification of resources in a secure and (to the user) transparent way. It is designed for collaboration across organizational boundaries in so called Virtual Organizations i.e. groups of people belonging to different organizations working together, sharing and using the same or similar resources, without regard of organization or underlying environment. The SDR works in a dynamic environment where nodes and clients come and go in an ad hoc manner. The management of resources is semantic-based, which means that each resource (component) has a metadata description attached to it. This metadata is based on a common ontology. A semantic search implies that a user can look for a component based on the definition of the concepts and terms involved, and the relation between them. Through semantic search not only the number of irrelevant hits are minimized, it is also possible to identify and discover components, which one otherwise could miss if the search was only based on keywords. For instance, a search for a vehicle with four wheels, which is at least a three-sitter, will discover normal family cars (since they are a subclass of the class vehicle) and at the same time eliminate other types of vehicles such as, two-sitter sport cars, motorbikes, busses, etc.

6.3.1 Requirements

The first step of the development of the SDR was requirement analysis. The requirements were identified partly based on our investigations of available platforms and related technologies and partly based on the needs of the NetSim environment. Based on these requirements, we designed an architecture that had the potential of meeting them. These requirements include the *underlying platform* (the underlying platform should be operating system independent, use open source software and implement well known standards as much as possible), *architecture* (the architecture should be easy to rebuild and modify, be robust and capable of handling a dynamic environment and provide functionality for semantic management and search of resources), *functionality* (the most important functionality requirements besides the semantic management and search of resources were the storage and retrieval of objects), and *security* (the security requirements involve issues like integrity and confidentiality of data, user authentication, secure communication and access control of objects in the repository).

6.3.2 Design and implementation

Following the requirement analysis a general design for the SDR based on the requirements has been constructed. To enable semantic management, search and retrieval two kinds of methods, techniques and software are needed. The first kind is needed for description, management, search and identification of the repository resources. The second kind is required for localization, storage and retrieval. Furthermore, techniques for communication and security with authentication, access control and encryption are also needed. A design with these features could be seen as a three-tier architecture in figure 16.

The Data layer consists of metadata and resource data storage. The Logic layer takes care of the semantic functionality, resource location, management and transfer. The Interface layer handles communications and security issues like authentication, authorization and encryption [123].

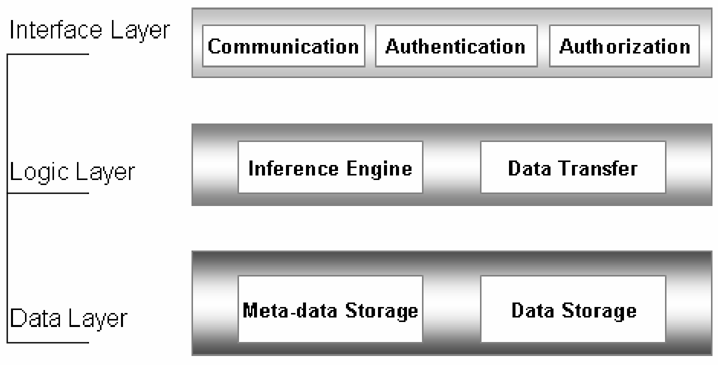


Figure 18 the three-tier SDR-architecture

The SDR has been designed as a network of nodes to which clients can connect. This design, based on a Distributed Hash Table (DHT) [3], is decentralized and no node has any administrative rights over any other node. The SDR deals with three main components, the nodes, the clients and the resources.

Each participating organization has one or more nodes that it provides to the SDR network, through which the clients/users access the network. There is no explicit limit to how many clients a node could handle. A client may be an application through which a user may explore the repository, or it may be another application that utilizes it for its own purposes. Clients interact with the SDR through a Web Service interface that allows the client to perform all available operations on resources in the SDR.

Resources in the repository have three kinds of objects associated to them, the data for the resource, the descriptions and the administrative metadata. The data is optional since it depends on the nature of the resource e.g. a computer resource is only represented by a URL. The administrative metadata is set by the SDR when the resource is created in the system. The description language for the resources is OWL. The description can be comprised of several components that have different focuses, i.e. a simulation model can be described from various aspects, purpose of the model, execution requirements, interface specifications, etc. The DRONT ontology, which we have developed within the project [108], has all of these elements in it. In this way the inference engine [107] can be used to infer conclusions about the resources.

The first prototype implementation of the SDR was done with a combination of Globus Toolkit 3.2.1 services (RLS [48], GSI [50], and GridFTP [49]) and Grid services as communication interface [48]. For the semantic framework the metadata language used was OWL, the ontology editor Protege was employed to build the OWL test ontology and Jena 2.2's internal inference engines MICRO and MINI were used [85]. The database management systems utilized were MySQL 4.1.7 for Jena and PostgreSQL 7.4.2 for RLS. The web server was Apache tomcat 4.1.31.

In the second version, SDR was ported to Globus Toolkit 4.0.1, the database systems were reduced to only PostgreSQL. Due to the improvements in GT-4, the communication interface is now handled by Web Services. The java version has also been updated from version 1.4 to 5.0. The biggest difference between these two SDR versions (besides the change to WS-interface) was the introduction of the DHT and Fortress modules. This totally changed the network topology of SDR and allowed more flexibility in the node availability.

6.4 Collaborative Core - CC

Collaborative services are another type of services provided by the NetSim environment. The main idea here is to support modellers, end-users, M&S experts, VV&A agents, etc, and provide means for collaboration during different stages of the M&S process, such as design, development and execution of simulation models. These services will not only bring different actors together and make the M&S process more efficient, they will also improve the quality and the user-worth of the models developed.

However, M&S applications are usually specialized for their purposes, and do not naturally provide support for collaborative services [80], [81], [82]. A general problem that developers of Computer Supported Collaborative Work (CSCW) applications face is the complexity of integrating management of collaboration groups and activities within an already existing application. As a result, the majority of the software used for computer-related professional tasks are single-user local desktop applications. Also the dedicated CSCW software tends to lag behind in other than CSCW functionality [45]. Especially within domain specific applications such as M&S applications, this kind of support is rarely seen. Thus we have addressed the problem through in-house design and development of the needed infrastructure, a foundation we call *Collaborative Core* (CC). In this regard we have studied techniques and methods for developing an infrastructure, which provides collaborative services, and which due to the nature of a Network Centric Defence should be distributed.

6.4.1 Requirements

The collaborative services module in NetSim mainly provides support for group management and tool sharing. Applications using the services do so without user specific complementary actions. CC leverages support and services required for a user to start, administrate, and participate in computer based collaboration groups, sharing tools and other functions in NetSim. Moreover, CC offers development support for integrating new tools for CSCW. CC comprises three main components (groups of services); *Communication Tools*, *Application Interface*, *Group Management Services*. The latter represents straight-forward services for CSCW group management and administration, including shared group status and areas. The second component provides a pluggable interface for collaborative applications, and relieves tools of the responsibility for managing user groups, and most of the responsibility for communication. Using this approach, developers can easily develop new CSCW tools, and modify existing applications to become collaborative.

The overall requirements for the CC module are as follow:

- *Distributed CSCW:* Due to the nature of a distributed defence, the environment should be distributed and not centralized. However, this does not exclude future combinations.
- *Synchronous work:* The CSCW intended here is immediate and synchronous.
- *Short Persistence Collaboration Groups:* The kind of tasks that will be performed within the NetSim collaborative work, are assumed to most often be directly task oriented, i.e. the life time of the collaboration is assumed to be short.
- *Small group sizes:* The groups are assumed to be small, 2-8 persons. Though considering scalability and regarding HLA as a candidate technology, larger groups are possible. However that requires a high level of social support, to address issues such as virtual conflicts, an issue not handled here.
- *Various client types:* Different client types are considered, such as thin clients with poor network connections. In this regard virtual worlds are too complex for the purpose, and hence we focus on shared context and space, rather than virtual place.

6.4.2 Design and Implementation

Figure 19 illustrates the general architecture for the CC. As shown in the figure, users access the environment via a NetSim client. The NetSim client is an application at the top layer of the NetSim architecture, which provides different users access to the NetSim tools, applications and other services in the environment. The CSCW services provided by the CC, namely Tools APIs, Communication Support and Group Management Services, can also be seen in the figure.

In order to avoid unnecessary development during the design phase we investigated various architectures, such as Peer-to-Peer (JXTA), Web Services and HLA, for handling communication between nods. The architecture that was finally chosen was HLA, which provides an infrastructure with essential services that beneficially could be used for CSCW, such as time management, group management (*Federation Management*), efficient information filtering (DDM), and communication management. HLA is a mature technology and standardized (IEEE 1516) used for development of various military and non-military applications [172], [186], which even have been evaluated for development of collaborative virtual environments [45], and since most simulation within our research complies with the HLA, it was a natural choice. Furthermore, advanced, flexible consistency management has been declared a lacking part in current implementations of CSCW and in existing systems [186]. Through utilization of HLA and the RTI, which provide advanced, flexible time management, this issue was addressed appropriately. However, HLA was not originally been developed for real-time applications, something that CSCW applications highly are. Thus, we evaluated the suitability of HLA for the purpose.

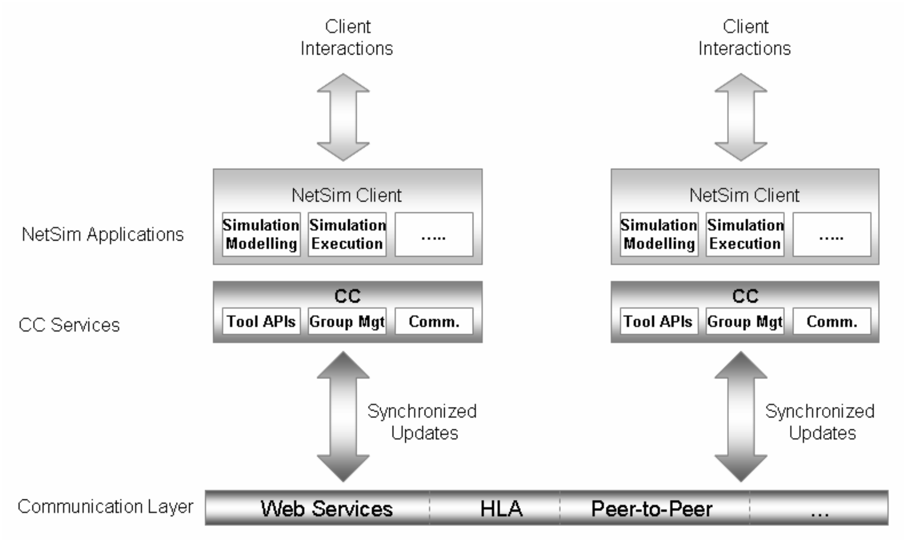


Figure 19 the general CC design

XML (the *Extensible Modeling Language*), which was perceived as suitable for structuring and handling information, was used for group definitions. XML provides a way of structuring information in a platform independent, human-readable way [132]. A very beneficial feature is that XML efficiently separates data from presentation. In CC services a lot of information is managed, such as tool specific information, collaboration group information and client information. A design choice could be to strictly follow the technology chosen (here HLA), but to accomplish a more generic structure and less technology dependent, XML was chosen. Another reason for using XML was that in our environment various client types are expected. Using XML, the same information is provided to all participants. At the client side, parsing of the XML formatted information can allow for user specific utilization and presentation.

Based on these design choices, we implemented the first CC prototype with focus on collaboration services, group services and tool APIs. A simple user GUI was created along with three simple, but demonstrative prototype applications, a text editor, a box drawing tool and a game. Additional CC features were also implemented, such as a *Participation Panel* (group specific information) and an *Activity Panel* (alerts tool activity etc.). These were used for practically evaluating the implemented CC functionality. In the second implementation of the CC communication support was also developed and tested [120]. Communication support included text, voice and video communication. As transport medium HLA was employed. The DDM services of RTI was used for efficient routing of information by directing information (*instant messaging*), and forming subgroups for communication within collaboration groups.

Applications in different nodes communicate through exchange of messages, where all messages are formatted in XML and validated towards an XML Schema created for this reason. This is provided by the Application Interface to support applications in using, parsing and reading XML.

Overall, results demonstrate feasibility of the CC infrastructure, and of the objective of extending the use of HLA to non-simulation applications. HLA proved well suited as communication structure for real-time user interactive applications, and the already built-in advanced functionality in HLA was beneficially used for CSCW services. Furthermore, XML conversion tests verified efficient use of XML for the purpose and combination with HLA for CSCW. For more information see [80], [81], [82], and [122].

6.5 Execution environment

The main purpose of the execution environment, referred to as DRMS (Distributed Resource Management System), has been provision of computing capacity for reliable execution of simulations. This is partly achieved through utilization of idle processing capacity in a network of workstations for distributed simulations, in stead of having a large number of dedicated computers as execution resources. In order to employ DRMS services desktop owners within an organization should be able to download and install a small client that under certain circumstances share resources with other connected nodes. However, this will result in an environment, where the availability of resources on the network is expected to change fast and unpredictably in an Ad-Hoc manner. So, the main question here is how to ensure reliable and fault-tolerance execution of simulations in a manner transparent to the users.

6.5.1 Requirements

To comply with the above issue and provide fault-tolerance the system should include mechanisms for migration, or movement, of federates between available computing resources during a federation execution. The dynamic characteristics of the network, also requires redundancy (replication) in storage of simulation components to gain access to the same set of federates at all times [110], [111], [112].

Furthermore, a major aspect to consider when implementing DRMS was discovery and matching of resources. The first problem relates to the basic strategy used to discover the presence of other nodes/resources on the network. Another problem is how to identify those resources that match certain requirements. The reason is that different simulations will most probably have different software and hardware requirements, such as the type of the operating system, run-time environment, CPU capacity, etc. These two issues, namely discovery and matching of resources, are addressed through utilization of SDR, by employing semantic annotation of computing resources and simulation models [110].

6.5.2 Architecture and Implementation

DRMS comprises two basic modules, namely a *worker module* and a *coordinator module* [111]. A worker is responsible for execution of one or more jobs, whereas a coordinator is responsible for the coordination of one or more workers in managing a batch of jobs. DRMS, as mentioned earlier, is dependant on a *repository service*. A repository is used by a worker to advertise its presence on the network and also its availability for execution of various jobs. Furthermore, the repository is used by a coordinator for localization of available workers. A repository also contains advertisements of other resources available on the network and is therefore used as entry point when worker services fetch resource files and executable code.

To enable uniform and semantically rich descriptions of resources within the environment, a *DRMS ontology* is needed, which is aligned with the general NetSim ontology describing different concepts in the environment and the relation between them. The DRMS ontology should comprise constructs for description of simulation models and computing resources. The main purpose of the ontology is to promote a shared view of information throughout the environment and facilitate localization and matching of resources. The choice of language for its representation is the Web Ontology Language (OWL) [178]. As discussed in section 4.1.3, the expressiveness of OWL is sufficient for representation of information required by the DRMS, and the ability to inference over information, is ideal for matching resources in the implementation.

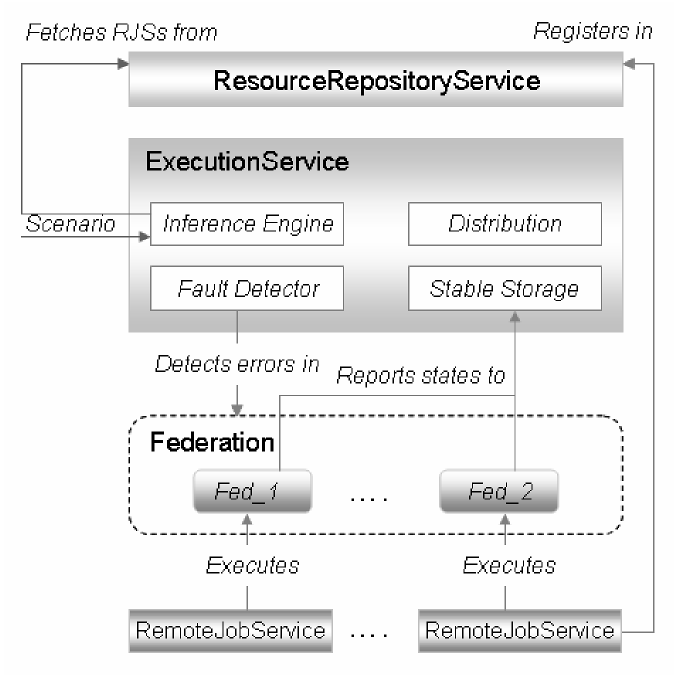


Figure 20 the general architecture of DRMS and the interrelation between different services

The general architecture for DRMS is illustrated in figure 20. For implementing a worker we have designed a *RemoteJobService*. When deployed on a workstation, this service will announce its presence on the network by registering an announcement in a repository. The announcement is represented by a meta-model, which defines the features of the RemoteJobService's host environment. This includes aspects such as the workstation's hardware configuration, OS type and version etc. The meta-model is an instance based on the DRMS ontology.

The *ResourceRepositoryService* is a representation of the SDR, which has been described in section 6.3. This service supports storage of meta-models, such as the meta-model describing the RemoteJobService's host environment. The interface of the ResourceRepositoryService includes methods for registering, deletion and lookup of meta-models. The lookup can either respond with the entire content of the ResourceRepositoryService, or a subset of registered meta-models, defined by a search query.

In order to implement the coordinator an *ExecutionService* has been designed. An ExecutionService is utilized by the NetSim environment when a single user, or group, requests execution of a scenario (federation). The main tasks of the ExecutionService are to automatically setup a federation and to monitor the federation execution.

Table 3 Service interfaces of the DRMS implementation

Service	Method
RemoteJobService	allocateJob(*Meta-model*)
	startJob(*Id*)
	stopJob(*Id*)
	getJobStatus(*Id*)
ResourceRepositoryService	addModel(*Meta-model*)
	deleteModel(*Meta-model*)
	getModels()
	getSubset(*Query*)
ExecutionService	requestExecution(*Scenario*)
	finalizeExecution(*Id*)
	getScenarioStatus(*Id*)

The implementation of DRMS is based on Web Services, the Axis platform [170], and Semantic Web technology, through use of the Jena toolkit [85]. The components of the architecture which have been implemented are: *DRMS ontology, RemoteJobService, ResourceRepositoryService,* and *ExecutionService* [110], [111].

Table3 outlines the service interface of the RemoteJobService, the ExecutionService, and the ResourceRepositoryService.

To enable fault-tolerant execution of federations the ExecutionService comprises a stable storage and a fault detector component [110]. These components are members of concerned federation through a common federate. The stable storage stores checkpoints reported from federates in the federation, whereas the fault detector detects failed federates in the federation and initiates preventive measures to resolve these errors. The error detector detects the failure of a federate by means of the HLAfederate object of the MOM (Management Object Model) [145], which is deleted if the link to the RTI is broken. As an additional measure the error detector calculates the time passed from the last reported checkpoint and if this value exceeds a pre-defined time, the federate is not longer considered active. When a test federate crashes, or its network connection is simply lost, the fault-detector initiates re-distribution of the lost component in the inference engine. The inference engine finds a new host environment for the federate under consideration, given the requirements of the federate as defined by its meta-model, and allocates the job to the RemoteJobService node. More information about DRMS can be found in [110], [111], and [112].

6.6 Summary

This chapter presents a brief introduction of the NetSim project and the general framework for component based simulation development that we have designed and implemented. The services that are developed to support the framework are also presented and discussed. Even though these services can be employed to facilitate component based design and development of simulations/simulation models, they have been designed in such way that it is possible to deploy them individually to support utilization of M&S in network centric infrastructures such as NCD.

Our second approach to development of the CBMD framework which we have briefly touched upon in this chapter is discussed in detail in the next chapter.

Chapter 7

A Framework for CBMD

As explained in the previous chapter, component based simulation model development (CBMD) was one of the services provided by the NetSim environment. In the framework that we developed for this purpose two different (still closely related) approaches were pursued. The first approach was focused on simulation (federate) reuse and composition (federation development), while the second approach was based on employment of reusable BOM-based model components and automatic code generation. This chapter will describe the work that we performed regarding BOM discovery and composition, and the framework, which has been developed for the purpose. The framework including the discovery and composition process was designed and developed iteratively during different steps. Below two distinct steps are presented, followed by a description of the agent based environment that we developed to implement the framework. However, before doing so, to make the presentations more clear and understandable, the rationale and the modelling and composition assumptions are described in the next section.

7.1 BOM-based Simulation Development

Developing simulations through reuse of existing simulation components requires a fair amount of adjustment and adaptation of the components, especially if the components are planned to be used in combinations and for purposes, other than what they have originally been intended to. This conclusion was easily drawn after our first implementation of the CBMD framework in the NetSim project [41], [38]. Furthermore, it was deducted that the larger and more complex the components get the harder the adaptation work is. And for real complex components the adaptation is almost impossible (or impractical) without proper component documentation. Hence, in order to compose a simulation out of components, the components need to contain (and expose) some information about their internal structure and how they can be used. This information is called metadata and contributes to simplified use of a component by others [93]. Generally, the concepts and terminologies used in various components may vary substantially and thus can lead to misunderstanding. Therefore the concepts and terminologies should be defined in an unambiguous way to avoid misunderstandings, particularly if the composition process is automated.

However, even having good documentation requires tedious labour to understand how a component works, how it has been implemented and what changes are required. Moreover the adjustments mainly ensure that components in a composition can communicate (i.e. syntactically) and the semantic validity of the composition can not be guaranteed. The validity is mainly checked after composition and by running the simulation and performing validity tests [144].

One approach to test and ensure the validity of composed simulation before actual composition is to provide (simulation) components with formal descriptions, i.e. structured models, which can be reasoned about and be used as a basis for selection and compositions. As mentioned in previous chapter this is the method that we used in the second approach for realisation of a component based simulation development framework in NetSim. This approach is based on model composition with the help of formal descriptions, automatic code generation and simulation development.

The first step toward this direction was taken in collaboration with National University of Singapore (NUS) [115]. In that work, since we had already worked with HLA federates and federation development in our first approach, it was natural to take a closer look at SOMs. The idea was to study SOM's feasibility for providing information for reasoning about composability of federates ahead of composing them. However, composability tests based on SOMs could easily be ruled out [39] [115]. SOM mainly provides information about the data that a federate exchanges with other federates. Even though there are some metadata available in a SOM, they can only provide very basic semantic information. Adding semantic information to SOMs in order to improve their usability is not feasible either, since a SOM is not a simulation model description and lacks the basic mechanism for providing information about the internal behaviour a simulation component [56]. And finally, automatic code generation based on SOM/FOM can only result in a code skeleton, which certainly can assist a simulation programmer, but is far from a complete simulation code [115].

Hence, it is desirable to identify another formalism for presenting models and describing simulation components. The BOM concept, as described in chapter 2, has been introduced by the HLA community and is the Modelling and Simulation community's proposed component standard. In BOMs, the interplay within a simulation or federation is captured and characterized in the form of reusable patterns. These patterns of simulation interplay are sequences of events between simulation elements [158].

Even though BOMs were created based on ideas from HLA, meaning they include HLA OMT information, BOMs may be used without this information to describe any type of simulation component, a feature which makes them even more versatile. Hence, BOMs could be suitable candidates for implementing a component based simulation model development framework. The first step to investigate the above assumption is to study the feasibility of BOM for describing simulation components and supporting semantic validity checks of compositions. In this work we also investigated whether BOMs could be used to create code-skeletons for federates and a whole federation (part II, paper 7). As BOMs contain high-level information as well as HLA OMT information it is possible to reduce the time needed to develop code that is not included in the actual simulation logic. This would be done by developing tools that use BOM information to automate the generation of such code and reduce development time and effort.

To create the above process it is required to set up some modelling and composition assumptions, which could be used as a basis for the work.

7.1.1 Modelling and composition assumptions

The main assumptions considered in this work are as followed. First of all simulation models are seen as combinations of *entities* and *events*. A composed model consists of a number of event-driven components, which communicate by sending messages. An event is something that happens at certain point in time such as, receiving information about the status of an airplane by another airplane. And an action is the service, or operation that is triggered after an event has happened, e.g. the computation done by the airplane and its change of state can be considered as the corresponding action to the above event. Basically, we define an action in terms of its effects on the environment and itself.

It is also assumed that the actions (components) are combined in a *Horizontal* manner. Two operations/actions can be combined *Horizontally* if they model a supply-chain like combination [1]. In order to describe Horizontal composition we introduced the concept of *mode*. The *mode* of an action shows whether the action initiates an interaction, via sending a message (*Out* mode), or it is invoked as a result of receiving a message (*In* mode). An action is either in *In* mode or *Out* mode and the mode can not be changed dynamically. The Horizontal composition is a combination of an *Out* action belonging to message sender entity with the respective *In* action of the entity (ies) receiving that message. Detailed information about the assumptions can be found in part II, paper 9.

In the two framework development steps that are presented in this thesis slightly different composition assumptions are considered. The composition in the first step is simpler and basically utilises the information presented by each BOM (including metadata and use history) and the accompanying ontology (written in OWL) to reason about compositions. In step two however we check compositions through matching of the Semantic BOM Attachments, which is an extension to the BOM description developed in this work. The Semantic BOM Attachments which is based on the ideas from Semantic Web and Web Services, as will be explained later.

In the next section the process that we developed during the first step and our conclusions are presented.

7.1.2 Composition framework based on BOMs and Ontology

In the composition framework we assume that a simulation developer describes the target scenario (simulation to be developed) in a formal manner using SRML-Light, described in section 5.2.1. This is followed by the development process consisting of three phases, *Discovery*, *Matching* and *Composition* (abbreviated as DMC).

The BOM Discovery phase starts by processing the provided SRML document and identifying the type of components (BOMs) that are required in the simulation. It is done by parsing out all the components mentioned in the SRML document, along with an Ontology that accompanies the SRML-Light document and stated its frame of reference. Consequently, the list of identified components together with Ontology-information is used to fetch candidate BOMs from a repository. The BOMs fetched have to be relevant to the simulation in some way, either as explicit components or components related to the simulation in some other way, such as loggers, rendering components etc. How this fetching is done depends on the BOM Repository and its interface. (In our work we originally deployed a simple file based repository, since the SDR was not available at the time.) The Discovery phase only identifies BOMs that roughly suite the intent of the simulation or match the components specified in the simulation document.

Following the first phase, the list of candidate BOMs are used as input to the BOM Matching phase. In this the fetched set of BOMs are compared and decides which BOMs might be suitable for the simulation. This is a more complex operation that needs to take into account the simulation intent, as described in the SRML document, and handles issues such as, what components fit together semantically and syntactically. In order to compare BOMs, additional information such as ontologies and reference documents is also used in this phase. The first step in this phase is to find suitable set of downloaded BOMs, and build different combinations each containing all the BOMs required by the SRML document. The reason is that we could discover more than one BOM for each identified component type in the SRML document, resulting in a number of combinations. Later each combination is analysed and their composability is checked. In our implementation the actual compatibility check utilized available metadata inside the BOMs and reference documents such as Ontologies to create a Composability score between each pair of BOMs.

This score indicates if two BOMs refer to the same types of events and entities, and if they publish or subscribe to events that had been specified in the SRML document. The score is also affected by previous use history, overlapping application domain and other general metadata, which is used to reinforce Composability Scores between BOMs. Figure 21 illustrates the logical flow of how matching of BOMs is done and a score is for each combination.

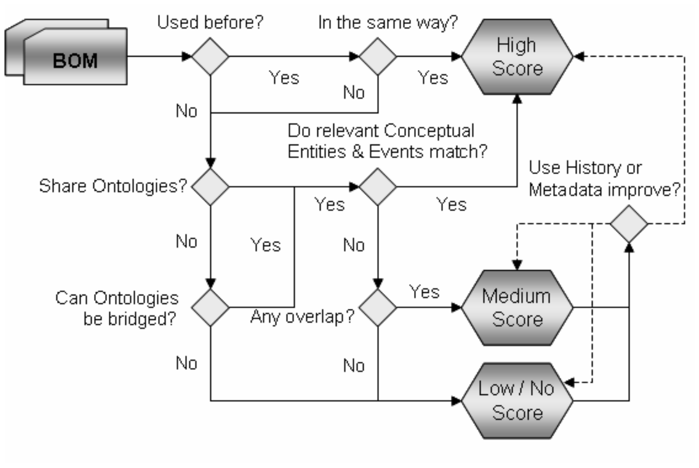


Figure 21 Matching flow of BOMs

In the last part of the process i.e. the BOM Composition phase, the selected components are assembled into a composite BOM, a BOM Assembly. This phase takes a number of BOMs as input, as well as the SRML document and mapping data to determine how to merge the BOMs together (as parsed out from the Mapping Suitability step). This information is used to create a BOM Assembly. This BOM Assembly is later intended to be used to either produce code for the execution of the simulation, or serve as a blue-print for composing BOM-implementations associated with the identified BOMs.

The component-based simulation development process, where the DMC part is highlighted, is illustrated in Figure 22.

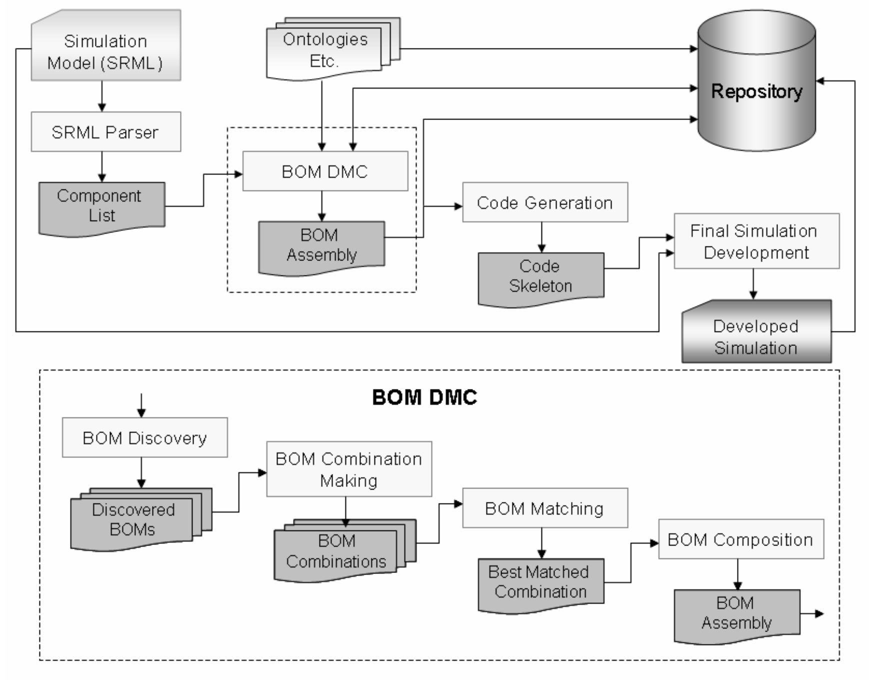


Figure 22 BOM Discovery, Matching and Composition process

Based on the above description an implementation of the BOM DMC process is made (part II, paper 7) and tested. The implementation and the test scenario in themselves are quite straight-forward, and do not represent significant discovery of any kind. For instance, determining that any simulation scenario could be modelled using SRML Light is of course impossible, but being able to formulate a simple and typical scenario from a component based view of simulation development in a formal way, and being able to make sense of it both visually and pragmatically, proves that the idea of composing simulations from components, for example BOMs, is indeed feasible.

The conclusion that is drawn from our study is that BOMs contain a great deal of information that greatly support a component based simulation process. This is due to the fact that BOMs contain numerous useful metadata and definitions of events and entities, as well as a state machine describing the internal behaviour of the components. However, we consider it essential to include semantic definitions and frame of reference (via ontologies) to avoid misunderstanding of terms and concepts. Since BOMs do not specify how this type of information should be included we have to find the best feasible way to do it.

The main weakness of the method used in this step is that the semantic matching is complete and there is no proper way of matching behaviour of the conceptual models of BOMs.

7.1.3 Composition Framework - the Rule-based Approach

The rule-based approach aims at overcoming the weaknesses of the previous framework described in section 7.1.2 by refining the development process, extending the BOM description with semantic information to avoid misunderstanding of terms and concepts, and providing a proper method for discovery and matching of BOMs (part II, paper 9).

As for refining the process, the composition framework from last phase is improved in this step, but the original idea is more or less the same. The process that is developed here contains four phases: a) *SRML Parsing* b) *BOM Discovery* c) *BOM Matching and Composition*, and d) *BOM Assembly Building*. As in the previous case, the first phase starts with a description of the target simulation written in SRML Light [181]. The simulation model contains simulation components, and events, as connectors of those components. The SRML item classes are seen as representation of BOM candidates while events (script-tag of SRML) represent actions between components. In the SRML parsing phase we parse the simulation scenario and extract information about candidate components (part II, paper 9), [36]. The output of the parsing step is a collection of entity names and their corresponding send/receive events. This collection is called *SRML Object Model*.

During the BOM discovery phase a query is built based on the SRML Object Model and is sent to the BOM repository. The repository returns a set of potential candidates corresponding to the query. Afterwards, through utilisation of a set of rules, the candidate BOMs are matched syntactically (number of parameters and event name) and semantically (parameter data type and entity type) against the SRML object model and the irrelevant BOMs are filtered.

The BOM matching and composition phase is more comprehensive and is about finding the right combination of components that satisfy the target simulation description (the scenario). Again as in the previous case this phase starts by making different combinations of candidate BOMs. Next, the composer adjusts the combinations based on received feedback from syntactic and semantic BOM matching, which is done according to the three-layer model, including a set of rules (part II, paper 9).

Finally, having in hand a right set of components, their interactions and the order of those, we enter the BOM assembly building phase during which a BOM assembly can be created from the current set of BOMs. Figure 23 depicts the BOM discovery and composition process in a flow chart format. This figure is an extension of figure 22, where the DMC process has been refined.

In chapter 4 we described the concept of ontology and Semantic Web. Semantic Web and ontologies have been pointed out by M&S experts as having the potential to support semantic composability [144], [66]. DeMO is one of the efforts that have been done in this direction, and is a taxonomy of simulation objects with the aim of supporting component based simulation development [137].

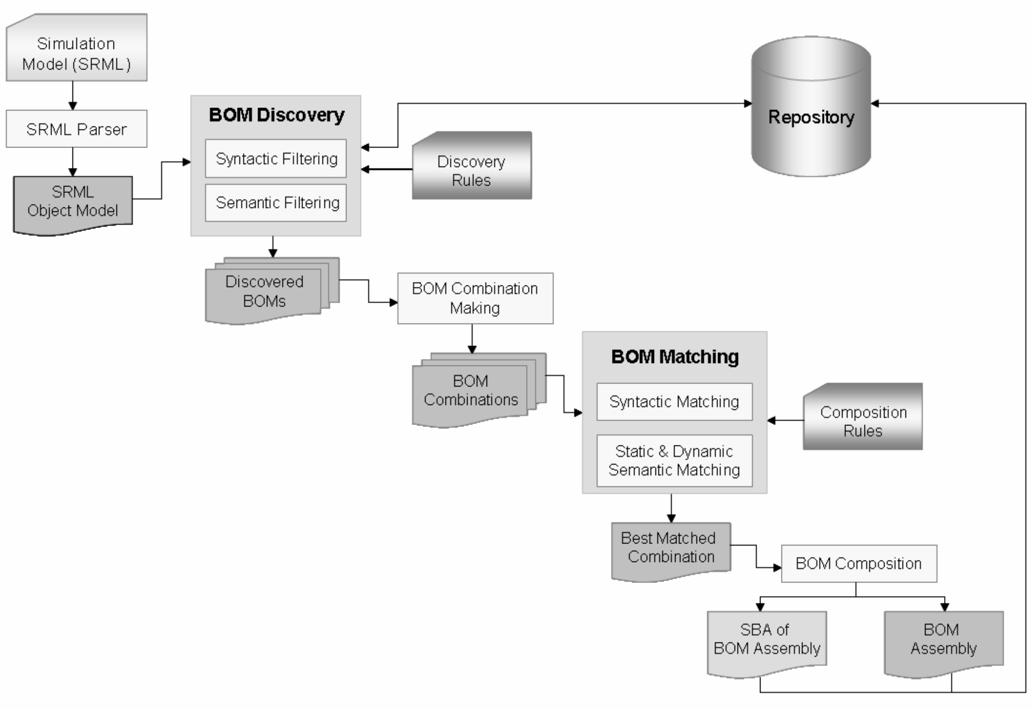


Figure 23 The BOM Discovery and Composition process

Another field of interest that is also mentioned in chapter 4 is Web Services (WS). As in the case of component based simulation development, WSs can be composed together and aggregated to deliver functionalities according to user requirements. There are well-adapted standards for publication, registration and discovery of WSs, while in the case of simulation components the M&S-community has just started to set up standards, which are not yet generally adapted. In [22] the potential of employment of WS for achieving semantic composability has been investigated and a method has been developed.

Based on the above discussion and in order to provide a proper basis for discovery and composition BOMs, we suggest an extension to the BOM description through utilization of Web Service technology and Semantic Web/OWL [35]. This extension, called Semantic BOM Attachment (SBA), provides the metadata required for discovering and composition of BOMs. In SBA the BOM description is mapped into the OWL-S upper ontology. Doing so OWL is used as the underlying language for describing BOMs. This is to support the ontology based reasoning process. Furthermore, features of OWL-S are also captured to improve the semantic expressiveness of BOMs and hence to facilitate semantic discovery and composition of them. Figure 24 depicts the items of BOM metadata converted into semantic BOM attachment plus the supportive ontology.

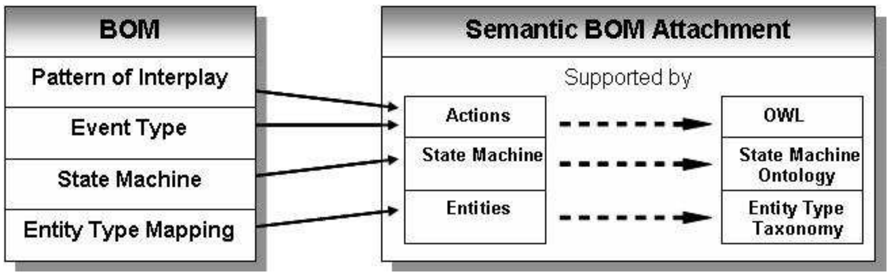


Figure 24 Semantic BOM Attachment

As for the matching of BOMs, in this step we introduce a three-layer model, utilising the SBAs of the BOMs and a set of rules for reasoning about the compositions. The model has been inspired by the work done in [9] where a composition stack for aggregating Semantic Web Services is presented. The model and the rules are part of an architecture that we proposed for implementing the component-based simulation model development process, as presented in part II, paper 9. The rules are divided into discovery and composition rules. The goal of discovery rules are to filter out irrelevant BOMs, using syntactic and semantic data.

The composition rules are divided in three layers. Each layer has a one or more composition rules verifying the composability of different items (data type, unit, component, state-machine, etc). The three layers are denoted as syntactic, static semantic and dynamic semantic. As the name suggests syntactic layer is concerned with the matching of syntactic information, such as message name, mode of action and number of parameters. Static semantic checks the entity and data types, while dynamic semantic handles matching of the components behaviour, i.e. state machines.

The mapping of the State Machine, i.e. the dynamic semantic matching, differs from the other items. State machine of each BOM represents the events that a component can send or receive in each state. It represents the internal behaviour of components, i.e. dynamic data. State Machine matching provides a means to ensure the causality of the compositions. Causality issue is considered as one of the main differences between component-based development in software engineering and M&S, since the state of simulation components might change between different time steps. Non-causal components are easier to define as one does not need to foresee how these components will be used [54]. Thus, causality introduces an important challenge when composing simulation model components. Detailed information regarding SBA and mapping different parts of BOM to OWL/OWL-S can be found in part II, paper 9.

Please note that simulation model development through composition might require some amount of component adaptation meaning that the composition process, as explained above, most probably will not present perfect matches. Hence, we need to somehow indicate how well a set of components match. This is done by defining a *composability degree* for each candidate combination. In order to calculate the composability degree, each rule is given a weight value, indicating the significance of the corresponding rule from the composer's point of view [9]. The composability degree of components is based on the composability degree of events i.e. a send event in one component with the corresponding receive event in the peer component. The composability degree for an event is computed after finding the degree of similarity at each rule and level. If the degree is greater than or equal to some threshold value, then the components are potential candidates.

Each level *i* is assigned a weight W_i, and each rule R_{ij} belonging to level *i* is assigned a weight W_{ij}. The function *Satisfied(Rule$_x$)* is defined for each pair of events and returns 1 if the Rule$_x$ is satisfied between the two events and zero otherwise. The following formula is used to calculate the Composability Degree.

$$ComposabilityDegree(ActionX) = \sum_{i=1}^{L} W_i * \left(\sum_{j=1}^{R_i} W_{ij} * satisfied(Rule_{ij})\right)$$

Where:

L = number of composability layers,

Ri = number of rules in layer i

W = weight assigned to a composability layer or a rule

The composition framework described above, has been implemented and evaluated. We have implemented the application in Java, and utilized Jena inference engine and the Jess rule engine for reasoning about matching and composition of BOMs. The BOMs are supported by Semantic BOM Attachments (SBAs) including related ontology. SBAs provide an important constitute for reasoning about the composability of BOMs. We believe that development of SBAs by employing techniques and methods from Semantic Web and Web Services is feasible, and the three-layered-scheme is promising and can improve composition of BOM-based components. However, the methods used in this approach can be refined, especially the algorithm for matching conceptual models can be further developed to handle more complex state-machines. For more details on this example see part II, paper 9.

In the next section another important part of our framework, namely the agent-based environment that has been developed to automate the execution of the proposed process, is presented.

7.2 Agent based environment

The aim of the developed composition process is to support a modeller with identification and discovery of components as well as giving feedback on feasibility of a composition. This process could be automatic or semi-automatic depending on the level of support that a modeller may require [103]. Hence, the process has to be encapsulated in an execution environment that is able to execute all the steps in the process on behalf of the modeller, and deliver required feedback at each step. In order to manage the complexity and for the purpose of maintenance the system has to be modular handling each step as a separate task communicating with other tasks.

Traditional software development approaches are difficult to use for building complex intelligent systems that posses multiple behaviours when operated in real-time environments. As presented in section 5.4, agent-oriented software engineering addresses the need for software systems to exhibit rational, human-like behaviour in their domains of expertise [162]. Software agents are autonomous entities that can communicate with each other to solve complex problems. These agents can evolve, i.e. their abilities can improve in time, and they can adapt to the environment they act within and the needs of the users [161]. Hence, we investigate the feasibility of agents as an alternative for development of the environment for discovery and composition of simulation models. In the environment that we have developed, the agents traverse through the component based simulation model development process, presented in the previous section, and execute all steps according to the needs of the modellers (part II, paper 10).

7.2.1 Design and Implementation

The agent framework that was deployed for development of the environment is the JACK agent framework [83], presented in section 5.4.1.1. The environment is designed based on a modular software development approach (part II, paper 10). The system architecture can be seen from two perspectives; *horizontal perspective*, which indicates the modules that the system contains of and *vertical perspective*, which indicates the layers system is comprised of (part II, paper 10). The business layer in the vertical perspective is the core of the system and deals with the business logic. This layer contains different agents handling the seven-step process composition process.

There are six types of agents available in our system: *Manager Agent*, *Parser Agent*, *Discovery Agent*, *Combination Agent*, *Composition Agent* and *Comparison Agent*. Manager Agent is the first agent to be called when the process is started. This agent is responsible for governing all the tasks.

The Parser Agent is the first agent to be contacted by the Manager Agent and is responsible for parsing an SRML file (simulation description) and creating a list of entity types with associated events and actions. The next agent is the Discovery Agent, which performs discovery on all the entities identified by querying the component repository and retrieving a list of BOMs. This agent also performs filtration based on the discovery rules utilising the Jena inference engine.

After the filtration, the Combination Agent creates the combinations of discovered BOMs, which are used by the Composition Agent to apply the composition rules upon and calculate their composability degrees. Jena OWL API is used inside the code body of the Composition Agent in order to traverse through the SBA of BOMs and perform semantic matching. As for matching the state machines the Jess rule engine is used. Finally, the comparison Agent sorts the list of combinations according to the logic defined by the simulation modeller and publishes the results accordingly.

The agent-based environment has been implemented and evaluated (part II, paper 10). The evaluation proved that with the help of agent technology, the entire process of BOM discovery and composition is streamlined. The approach provides support for a modeller and gives relevant feedback during different steps of the process.

The modular design of the agent framework provides a solid basis for further development and improvement of the environment, since it is possible to introduce new agent plans, beliefs and capabilities in order to deal with more complex situations, such as adding new discovery and composition rules, or more complex algorithms for combining components, without altering the architecture of the environment. Moreover, it is possible to deploy the environment on various platforms. It can be executed in a single desktop machine or can be deployed over a distributed network environment and be executed across different machines. Hence, our results indicate that the environment provides portability, adaptability and flexibility. More information about the agent based environment can be found in part II, paper 10.

7.3 Summary and future Work

This research initiative has been an effort to implement an agent based environment for automated discovery and composition of BOM based simulation components. The work includes:

- design and development of a framework for component based simulation development comprising a collaborative working environment, a semantic based distributed repository, and an execution environment
- investigating two different approaches for component based simulation development
- development of a process for component based simulation development based on BOMs
- suggesting an extension to the current BOM specification using Semantic Web and Web Services technologies
- development of a method for discovery and composition of BOMs
- development of a novel method for matching BOM state machines
- design and implementation of an agent based environment for realisation of the developed process

The work that has been conducted is not finished by any means and further development and improvements are needed. Some of the suggested improvements and areas of future research are mentioned below.

The BOM matching method needs to be validated using a formal approach such as the semantic composability theory (SCT). For instance, at the moment simple state machines, with no internal loops, are considered. Hence, the algorithm for matching needs to be further developed using more complex state machines.

It is desirable to investigate whether the collaborative modelling environment, which was used in the first approach to develop a framework for component based simulation development, can support a modeller using the agent based environment. If this is the case some effort is needed to couple the agent based environment with the modelling environment. This could include development of an automated SRML modelling GUI tool to support the modeller.

The agent system can be redesigned to achieve parallelism in the matching pipeline, hence increasing its performance. This involves a great deal of negotiation, co-ordination and communication (which are already known features of the agent technology), to serve parallel tasks.

It is also required to model more complex simulation scenarios in order to test and evaluate the system thoroughly. For this purpose new ontologies and new BOM descriptions need to be developed.

The SRML-Light document can be extended to include more features (script-tags) for representing simulation scenarios resulting in better queries thus improving the discovery of BOMs. A better description of the simulation scenario will also help identifying BOMs which do not represent (update) any entities, but provide services required by other BOMs.

More research and implementation is also required regarding the automated code generation step of the process.

Chapter 8

References

[1] A. J. Courtemanche, R. L. Wittman, *OneSAF : A Product Line Approach for a Next-Generation CGF*, Proceedings of the Eleventh Conference on Computer-Generated Forces and Behavior Representation, Orlando FL, May 2002.

[2] A. Adams Zabek, A. Wilson, and M. Fischer, *The ALSP Joint Training Confederation and the DOD High Abstraction Architecture*, In Proceedings of the 14th DIS Workshop, March. 1996.

[3] A. Ghodsi, Distributed k-ary System: Algorithms for Distributed Hash Tables, Royal Institute of Technology, KTH, 2006.

[4] A. Kleppe, J. Warmer, W. Bast, MDA Explained, The Model Driven Architecture: Practice and Promise, Addison-Wesley, 2003.

[5] A. Lehmann, *Component-based modeling and simulation - status and perspectives,* In Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Realtime Applications, Budapest, Hungary, October 2004.

[6] A. M. Law, W. David Keaton, *Simulation Modeling and Analysis,* McGraw-Hill, 2nd edition, 1991.

[7] A. Tolk, Avoiding Another Green Elephant – A Proposal For the Next Generation HLA Based On the Model Driven Architecture, in Proceedings of the 2002 Fall Simulation Interoperability Workshop, Orlando, FL, September 2002.

[8] B. Meyer, *Object-Oriented Software Construction,* Prentice Hall. ISBN 0-13-629155-4.

[9] B. Mojtahed, A. Bouguettaya, *A Multilevel Composability Model for Semantic Web Services*, Journal of IEEE Transactions on Knowledge and Data Engineering, VOL. 17, No. 7, July 2006.

[10] B. P. Zeigler, H. Prähofer, T. Gon Kim, *Theory of Modeling and Simulation,* Academic Press, 2nd edition, 2000.

[11] B. P. Zeigler, H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models*, Arizona Center for Integrative Modeling and Simulation, University of Arizona and Arizona State University, Tucson, Arizona, USA, January 2005.

[12] Base Object Model, http://www.boms.info.

[13] BOMworks, http://www.simventions.com/bomworks/.

[14] B. Powel Douglass, *Components, states and interfaces, oh my!,* The Unified Software Development Process, Addison-Wesley April 2000.

[15] C. M. Macal, M. J. North, *Toturial on Agent-Based Modelling and Simulation*, in Proceedings of the 2005 Winter Simulation Conference, Orlando Florida, USA, 2005.

[16] C. Szabo, Y. M. Teo, *On Syntactic Composability and Model Reuse*, in Proceedings of the first Asia Modelling Symposium, AMS 2007, March 2007.

[17] C. Hewitt, J. Inman. *DAI Betwixt and Between: From "Intelligent Agents" to Open Systems Science,* IEEE Transactions on Systems, Man, and Cybernetics. Nov./Dec. 1991.

[18] C. Kennedy, G. K. Theodoropoulos, *Intelligent management of data driven simulations to support model building in the social sciences*, Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part III, volume 3993 of Lecture Notes in Computer Science, Springer, 2006.

[19] C. Szyperski, *Component Software,* Addison-Wesley, 2nd edition, 2002.

[20] COM+, http://msdn2.microsoft.com/en-us/library/ms685978(VS.85).aspx.

[21] DAML-S Coalition: Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew McDermott, D. Martin, Sh. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, *DAML-S: Web Service Description for the Semantic Web*, in Proceedings of The First International Semantic Web Conference (ISWC), 2002.

[22] D. Bell, et al, *Sematic Web Service Architecture for Simulation Model Reuse*, in Proceedings of the 11th International Symposium on Distributed Simulation and Real Time Applications, DS-RT '07, Crete, Greece, October 2007.

[23] D. Brutzman, A. Tolk, JSB Composability and Web Services Interoperability Via Extensible Modeling & Simulation Framework (XMSF), Model Driven Architecture (MDA), Component Repositories, and Web-based Visualization, Technical Report, Naval Postgraduate School and Old Dominion University, November 2003.

[24] D. Garlan, R. T. Monroe, D. Wile, *Acme: Architectural description of component-based systems*, In G. T. Leavens and M. Sitaraman, editors, Foundations of Component-Based Systems, pages 47–68, Cambridge University Press, 2000.

[25] Distributed Interactive Simulation (DIS) Master Plan, Headquarters Department of the Army, September 1994.

[26] E. Bonabeau, Agent-based modelling: methods and techniques for simulating human systems. In *Proceedings of National Academy of Sciences* 99(3): 7280-7287, 2001.

[27] E. H. Durfee, V. Lesser, *Negotiating Task Decomposition and Allocation Using Partial Global Planning,* In Distributed Artificial Intelligence, Volume 2, 229–244. San Francisco, California, 1989.

[28] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF: W3C Working Draft, October 2006.

[29] E. W. Weisel , M. D. Petty, and R. R. Mielke, *A Survey of Engineering Approaches to Composability*, Proceedings of the Spring 2004 SIW, Arlington VA , April 18-23, 2004.

[30] E. W. Weisel, M. D. Petty, and R. R. Mielke, *Validity of Models and Classes of Models in Semantic Composability*, in Proceedings of the Fall 2003 SIW, Orlando FL, Sept 14-19 2003.

[31] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider: *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, Cambridge, UK, 2003.

[32] F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M. C. Shan, *Adaptive and Dynamic Service Composition in eFlow*, Proc. of the Intl. Conf. on Adv. Info. Systems Engineering, Sweden, 2000.

[33] F. Kuhl, R. Weatherly, J. Dahmann, Creating Computer Simulation Systems: An Introduction to High Level Architecture, Prentice Hall PTR, 2000.

[34] F. Moradi, R. Ayani, P. Nordvaller, *Simulation Model Composition using BOMs*, in Proceedings of The 10-th International Symposium on Distributed Simulation and Real Time Applications, DS-RT '06, October 2006.

[35] F. Moradi, *Component-based Simulation Model Development using BOMs and Web Services*, in Proceedings of the first Asia Modelling Symposium, AMS 2007, March 2007.

[36] F. Moradi, R. Ayani, G. Tan, H. Akbari, Sh. Mokarizadeh, *A Rule-based Approach to Syntactic and Semantic Composition of BOMs*, in Proceedings of the 11th International Symposium on Distributed Simulation and Real Time Applications, DS-RT '07, Crete, Greece, October 2007.

[37] F. Moradi, R. Ayani, I. Mahmood, *An Agent based Environment for Simulation Model Composition*, to appear in Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation PADS 2008, June 2008, Rome, Italy.

[38] F. Moradi, M. Eklöf, M. Garcia Lozano, D. Nordqvist, J. Ulriksson, *Nätverksbaserad Modellering och Simulering – Arkitekturen*, FOI metodrapport, FOI-R--1439—SE.

[39] F. Moradi, M. Eklöf, M. Garcia Lozano, D. Nordqvist, J. Ulriksson, *Nätverksbaserad Modellering och Simulering – Prototypen*, FOI metodrapport, FOI-R--1767–SE.

[40] F. Moradi, M. Eklöf, L. Ferrara, M. Garcia Lozano, D. Nordqvist, M. Persson, *Nätverksbaserad Modellering och Simulering – Utvärdering*, FOI metodrapport, FOI-R--2144–SE.

[41] F. Mordi, M. Eklöf, J. Ulriksson, *A network based environment for modelling and simulation*, in Proceedings of SimSafe Conference, June 15-17, 2004, Karlskoga, Sweden.

[42] F. Moradi, R. Ayani, *Parallel and distributed simulation,* Applied system simulation – Methodologies and applications, 2003, p. 457-486.

[43] F. Moradi, R. Ayani, G. Tan, *Some Ownership Management in Distributed Simulation using HLA/RTI*, Parallel and Distributed Computing Practices, Vol. 4, No. 1, June 2001, Nova Science Publishers.

[44] G. Chen, B. K. Szymanski, Component-oriented Simulation Architecture: Toward Interoperability and Interchangeability, In proceedings of the 2001 Winter Simulation Conference, Arlington, VA, USA, 2001.

[45] G. Chung, P. Dewan, *Towards Dynamic Collaboration Architectures,* Proceedings of ACM conference on Computer Supported Cooperative Work, Chicago, USA, November 2004.

[46] G. M. Pearman, Naval Postgraduate School, Monterey, CA, Comparison Study of Janus and Jlink, June 1997.

[47] GIG, *Department of Defense Global Information Grid Architectural Vision*, Prepared by DoD CIO,Version 1.0 June 2007, Available at http://www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf.

[48] Globus. http://www.globus.org/.

[49] GridFTP, http://www-unix.globus.org/toolkit/docs/4.0//data/gridftp/.

[50] Grid Security, http://www.unix.globus.org/toolkit/docs/ 3.2/security.html.

[51] G. Antoniou, F. van Harmelen, *A Semantic Web Primer*, The MIT Press, 2004.

[52] H. Aydt, S. J. Turner, W. Cai, M. Yoke Hean Low, *An Agent-Based Generic Framework for Symbiotic Simulation Systems*, Book Chapter in Agents, Simulation and Applications, to appear in 2008, by Taylor and Francis.

[53] H. A. Marshall, *SAF in CATT Training Systems, Update 1999*, Proceedings of the Eighth Conference on Computer Generated Forces and Behavioral Representation, 8TH-CGF-032.

[54] H. Praehofer, J. Sametinger, A. Stritzinger, *Building Reusable Simulation Components*, Proceedings of WEBSIM2000, Web-Based Modelling & Simulation, San Diego 2000.

[55] H. S. Sarjoughian, *Model Composability*, In Proceedings of the 2006 Winter Simulation Conference, Monterey, California, 2006.

[56] HLA at DMSO, https://www.dmso.mil.

[57] HLA Interface Specification, Version 1.3 (April 02, 1998) (http://hla.dmso.mil/hla/tech/ifspec/).

[58] HP-Lab. Jena: Semantic web framework. http://jena.sourceforge.net/documentation.html.

[59] I. B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko, *Ontology-driven Web Services Composition Platform*, Journal of Information Systems and e-Business Management, 3(2):175-199, July 2005.

[60] I. B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko., *Ontology-driven Web Services Composition Platform*, IEEE Intl. Conf. on e-Commerce Technology, San Diego, California, July 6-9, 2004.

[61] IBM, Business Process Execution Language for Web Services version 1.1, http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[62] IBM Web services tutorial. Online : http://www-106.ibm.com/developerworks/webservices/.

[63] Institute of Electrical and Electronics Engineers: IEEE Standards Computer Dictionary: *A compilation of IEEE Standard Computer Glossaries*, NewYork 1990.

[64] Interoperability Development for Interprise Application and Software, *IDEAS*, European project, 2002.

[65] J. A. Miller, G. Baramidze, *Simulation and the Semantic Web*, in Proceedings of the Winter Simulation Conference, 2005.

[66] J. A. Miller, G. T. Baramidze, A. P. Sheth, P. A. Fishwick, *Investigating Ontologies for Simulation Modeling*, Annual Simulation Symposium, Arlington, VA, USA, 2004.

[67] J. Bachman, P. Gustavson, R. Lutz, R. Scrudder, *Understanding the BOM Metadata and Making It Work For You*, 2005 Simulation Interoperability Workshop, 2005.

[68] J. Banks, Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice, Wiley Interscience Publication, 1998.

[69] J. Cardoso, *Quality of Service and Semantic Composition of Workflows*, Ph.D. Dissertation, Dept. of Computer Science, Univ. of Georgia, Athens, GA, 2002.

[70] J. Davies, D. Fensel, F. van Harmelen, *Towards the Semantic Web: Ontology-Driven Knowledge Management*, John Wiley & Sons, 2003.

[71] J. Ivers, G. A. Moreno, *Model-Driven Development with Predictable Quality*, in Proceedings of Conference on Object Oriented Programming Systems Languages and Applications, Montreal, Quebec, Canada, 2007.

[72] J. Ivers, N. Sinha, K. Wallnau, *A basis for composition language CL,* Technical report, Carnegie Mellon University, Software Engineering Institute, September 2002.

[73] J. Kim, E. Sosa, *Metaphysics: An Anthology*, Blackwell Publisher, 1999.

[74] J. L. Fiadeiro, A. Lopes, M. Wermelinger, *A mathematical semantics for architectural connectors*, Lecture Notes in Computer Science, 2793:190–234, 2003.

[75] J. Misra, *Distributed discrete event simulation*, ACM Computing Surveys, Vol. 18, No. 1, pp. 39--65, March 1986.

[76] J. O. Kephart, J. E. Hanson, A. R. Greenwald, *Dynamic pricing by software agents*, Computer Networks Volume 32, Issue 6, Pages 731-752, May 2000.

[77] J. S. Dahmann, R. M. Fujimoto, R. M. Weatherly, *The department of defense high level architecture,* In Proceedings of the 1997 Winter Simulation Conference, 1997.

[78] J. Siegel, *CORBA 3 - Fundamentals and Programming*, John Wiley & Sons, ISBN 0-471-29518-3.

[79] J. Ulriksson, F. Moradi, O. Svensson, *A Web-based Environment for building Distributed Simulations*, in Proceedings of the European Simulation Interoperability Workshop, 02E-SIW-036, June 2002.

[80] J. Ulriksson, R. Ayani, *Consistency Overhead using HLA for Cooperative Work,* 9:e IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005), Montreal, Kanada, oktober, 2005.

[81] J. Ulriksson, R. Ayani, F. Moradi, *Collaborative Modelling and Simulation in a Distributed Environment.* Proceedings of European Simulation Interoperability Workshop, Stockholm, Juni 2003.

[82] J. Ulriksson, F. Moradi, M Liljeström, N Montgomerie-Neilson, *Building a CSCW Infrastructure based on an M&S Architecture and XML*, Published in the Springer lecture notes of the 2:nd conference on Cooperative Design, Visualization and Engineering (CDVE2005), Palma de Mallorca, September, 2005. Springer ISSN: 0302-9743.

[83] JACK Agent Framework, Autonomous Decision-Making Software, http://www.aosgrp.com/products/jack/index.html.

[84] Jakarta Bean Scripting Framework, http://jakarta.apache.org/bsf/.

[85] JAMES II, Java-based Agent Modeling Environmnet for Simulation II, http://wwwmosi.informatik.uni-rostock.de/mosi/projects/cosa/james-ii.

[86] Jena - Verktyg (API) för hantering av RDF och OWL strukturer. http://jena.sourceforge.net.

[87] Jess web site, http://herzberg.ca.sandia.gov/jess/.

[88] Joint Chiefs of Staff Publication, Department of Defense Dictionary of Military and Associated Terms, No 1-02, 1999.

[89] K. Aberer et al., *The essence of P2P: A refernce architecture for overlay networks,* In Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing, August 2005, Konstanz, Germany.

[90] K. C. Wallnau, Volume iii: A technology for predictable assembly from certifiable components, Technical report, Carnegie Mellon University, April 2003.

[91] K. Chandy, and J. Misra, *Distributed simulation: A case study in design and verification of distributed programs*, IEEE Transactions on Software Engineering SE-Vol. 5, No. 5, pp. 440--452, Sept. 1979.

[92] K. Houston, D. Norris, *Software Components and the UML*, Component Based Software Engineering: Putting the Pieces Together, Addison-Wesley, 2001.

[93] K. Morse, M. Petty, P. Reynolds, W. Waite, P. Zimmerman, Findings and Recommendations from the 2003 Composable Mission Space Environments Workshop, 2004.

[94] K. P. Sycara, *Multiagent Systems*, The American Association for Artificial Intelligence, 1998.

[95] L. Bass, J. Ivers, M. Klein, P. Merson, *Reasoning frameworks*, Technical report, Carnegie Mellon University, Software Engineering Institute, July 2005.

[96] L. Lin, I. B. Arpinar, *Discovery of Semantic Relations between Web Services*, IEEE International Conference on Web Services, ICWS 2006, Chicago, IL, September 2006.

[97] L. Mellon, and D. West, *Architectural optimizations to advanced distributed simulation*, In Proceedings of the 1995 Winter Simulation Conference, Arlington, USA, pp 634-641, December 1995.

[98] L. Winters, A. Tolk, *The Integration of Modeling and Simulation With Joint Command and Control on the Global Information Grid*, Proceedings of the Spring Simulation Interoperability Workshop, San Diego, CA, IEEE Press, Apr 2006.

[99] M. Abadi, L. Cardelli, *A Theory of Objects*, ACM, 1996.

[100] M. D. Petty, *Semantic Composability and XMSF*, XMSF Technical Challenges Workshop 2002, Monterey CA, August 19-20 2002.

[101] M. D. Petty, *Simple Composition Suffices to Assemble any Composite Model*, Proceedings of the Spring 2004 Simulation Interoperability Workshop, Orlando FL, April 18-23 2004.

[102] M. D. Petty, Eric W. Weisel, *A Composability Lexicon*, Proceedings of the Spring 2003 Simulation Interoperability Workshop, Orlando FL, April, 2003.

[103] M. D. Petty, E. W. Weisel, R. R. Mielke, *Overview of a Theory of Composability*, Virginia Modeling Analysis & Simulation Center, Old Dominion University, 2004.

[104] M. Hofmann, *Introducing pragmatics into VV&A*, in Proceeding of the European Simulation Interoperability Workshop, ESIW 2002, London, UK.

[105] M. P. Papazoglou and D. Georgakopoulos, *Service Oriented Computing,* Communications of the ACM, 46(10):25–28, 2003.

[106] M. Prietula, L. Gasser, K. Carley, Simulating Organizations: Computational Models of Institutions and Groups, MIT Press, Cambridge, MA, 1998.

[107] M. Wooldridge, *Reasoning About Rational Agents*, MIT Press, 2000.

[108] M. Garcia Lozano, M. Chenine, V. Kabilan, *A Pattern for Designing Distributed Heterogeneous Ontologies for Facilitating Application Interoperability*, Proceedings of 18:th Conference on Advanced Information Systems Engineering 2006, Caise'06 Luxembourg, June 5-9, 2006.

[109] M. Garcia Lozano, F. Moradi, R. Ayani, *SDR: A Semantic Based Distributed Repository for Simulation Models and Resources*, in Proceedings of the first Asia Modelling Symposium, AMS 2007, March 2007.

[110] M. Eklöf, F. Moradi, R. Ayani, *A Framework for Fault-Tolerance in HLA-based Distributed Simulations*, Proceedings of the 2005 Winter Simulation Conference, M.E. Kuhl, N.M. Steiger, F.B. Armstrong and J.A. Joines, eds. Orlando, USA, December, 2005.

[111] M. Eklöf, R. Ayani, F. Moradi, *Evaluation of a fault-tolerance Mechanism for HLA-Based Distributed Simulations*, Proceedings of the 20th Workshop on Parallell and Distributed Simulations, PADS, Singapore 2006.

[112] M. Eklöf, M. Sparf, F. Moradi, R. Ayani, Peer-to-Peer-Based Resource Management in Support of HLA-Based Distributed Simulations, SIMULATION, Vol. 80, p. 181-190, maj 2004.

[113] M. Eklöf, J. Ulriksson, F. Moradi, *NetSim – A Network Based Environment for Modelling and Simulation*, NATO Modeling and Simulation Group, Symposium on C3I and M&S Interoperability, Antalya, Turkey, 2003.

[114] M. Eklöf, R. M. Gustavsen, A. Hjulstad, O. M. Mevassvik, *An Execution Enviroment for Distributed Simulations*, Collaborative project between FOI and FFI, FOI/rapport, FOI-R-2114--SE.

[115] Master thesis: A Software Engineering Approach to HLA-Based Simulation Development, HLA – XVCL, Andersson, Carl-Johan, Krantz, Mattias, KTH 2005.

[116] Master thesis: *Carbonara - a semantically searchable distributed repository*, Baymani S, Stridsfeldt E, Institutionen för tillämpad IT. KTH Kista, Maj 2005. http://www.imit.kth.se/~rassul/exjobb/rapporter/sima-emil.pdf.

[117] Master thesis: *Distributed Repository Ontology and a Design Pattern for Application Ontologies*, Chenine M, Royal Institute of Technology, Stockholm, Sweden, 2006.

[118] Master thesis: *Component-based Modelling and Simulation*, H. G. Moser, KTH, January 2006.

[119] Master thesis: Generic XML-based Interface for Computer Supported Collaborative Work in an HLA Environment, M Liljeström, FOI-S--1776--SE March 2005.

[120] Master thesis: *Kommunikationsmedel för datorbaserad distribuerad samverkan*. L. Ferrara, Institutionen för tillämpad IT. FOI-S- -0232—SE, KTH Kista, September 2005.

[121] Master thesis: En implementation av PKI-baserad Single Sign-On för Web Services, A. Frey, KTH 2005.

[122] Master thesis: Collaborative software infrastructure based on the High Level Architecture and XML, N. A. Montgomerie-Neilson, , FOI-S--1775--SE March 2005.

[123] Master thesis: *Authorization in Semantic based Distributed Repository*, Z. A. Shah, Institutionen för tillämpad IT, 2006.

[124] Master thesis: *Merging RLS-based Resource Indexing and Management with a DHT*, D Torres, J Pan, KTH Kista, December 2006. Available at: http://web.it.kth.se/~rassul/exjobb/rapporter/jun-pan.pdf.

[125] M. Luck, et al, *Agent Technology: Computing as Interaction*, European Coordination Action for Agent Based Computing (IST-FP6-002006CA) AgentLink III, ISBN 0854328459, September 2005.

[126] M. C. Daconta, L. J. Obrst, K. T. Smith. *The Semantic Web: A Guide to the Future of XML,Web Services andKowledge Managment*. John Wiley & Sons, 2003.

[127] MDA Guide Version 1.0.1, Object Management Group 2003, http://www.omg.org/docs/omg/03-06-01.pdf.

[128] Model Driven Architecture in Wikipedia, The Free Encyclopedia, Date of last revision: 13 January 2008, http://en.wikipedia.org/wiki/Model-driven_architecture.

[129] Mozilla Rhino Home Page, http://www.mozilla.org/rhino/.

[130] N. R. Jennings, On agent-based software engineering, *Artificial Intelligence,* 117:277-296, 2000.

[131] OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/.

[132] Object Management Group, *eXtensible Markup Language*, http://www.w3.org/XML/.

[133] Object Management Group, *Unified modeling language: Superstructure*, URL, http://www.omg.org/cgi-bin/doc?formal/05-07-04, August 2005.

[134] Object Management Group, *Meta-Object Facility*, http://www.omg.org/technology/documents/formal/-mof.htm.

[135] Object Management Group, *XML Metadata Interchange*, http://www.omg.org/technology/documents/-formal/xmi.htm.

[136] P. A. Fishwick, Simulation Model Design and Execution: Building Digital Worlds, Prentice Hall, 1995.

[137] P. A. Fishwick, J. A. Miller, *Ontologies for modeling and simulation: Issues and approaches*, in Proceedings of Winter Simulation Conference, Washington, DC, USA, 2004.

[138] P. G. Basset, Framing Software Reuse – Lessons from the Real World, Prentice Hall, 1997.

[139] P. Carlisle, W. Babineau, and R. Wuerfel, *The Joint Simulation System (JSIMS) Federation Management Toolbox*, Proceedings of the Fall 2003 Simulation Interoperability Workshop, 03F-SIW-048.

[140] E. H. Page, B. S. Canova, J. A. Tufarolo, *A case study of verification, validation, and accreditation for advanced distributed simulation*, ACM Transactions on Modeling and Computer Simulation, Valume 7, Issue 3, pp. 393-424, 1997.

[141] P. K. Davis, R. H. Anderson, *Improving the Composability of Department of Defence Models and Simulations*, Prepared for the Defence Modeling and Simulation Office, RAND 2003.

[142] P. Davidson. *Multi agent based simulation: Beyond social simulation,* Lecture Notes in Computer Science, 1979:97–107, 2000.

[143] P. Gustavason, L. Root, Ph. Zimmerman, Ch. Turrell, *Conceptual to Composable: Driving Towards Rapid Development of Simulation Spaces*, In proceedings of the Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), Orlando, USA, 2003.

[144] R. G. Bartholet, D. C. Brogan, Paul F. Reynolds, Jr., Joseph C. Carnahan, *In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design*, Proceedings of the 2004 Fall SIW, Orlando, FL, September 2004.

[145] R. H. Nelson, A Next-Generation Federation Management Tool: Using the Management Object Model (MOM) and FOM-specific Data to Monitor an HLA Federation, in Proceedings of the Fall Simulation Interoperability Workshop, 00F-SIW-155, Fall 2000.

[146] R. H. Wallace, *Practitioner's Guide to Ada*, McGraw-Hill, NY, 1986.

[147] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Parallel and Distributed Computing. Wiley-Interscience, 2000.

[148] R. R. Ropelewski, *SIMNET Training Concept Hones Battlefields Skills*, Armed Forces Journal International, June 1989.

[149] R. Suzić, *Embedded Simulation Systems for Network Based Defense*, User Report FOI-R--1828--SE, Swedish Defence Research Agency, Stockholm, Sweden, 2006.

[150] R. Weatherly, D. Seidel, and J. Weissman, *Aggregate abstraction simulation protocol*, In Proceedings of the Summer Computer Simulation Conference, Baltimore, Maryland, USA, 1991.

[151] S. Dustdar, W. Schreiner, *A survey on web services composition*, International Journal of Web and Grid Services, 1(1), 1 - 30, 2005.

[152] S. Mellouli, G. Mineau, et al., *Laying the foundations for an agent modelling methodology for fault-tolerant multi-agent systems*, in Fourth International Workshop Engineering Societies in the Agents World, Imperial College London, UK, 2003.

[153] S. Narayanan and S. Mcllraith, *Simulation, Verification and Automated Composition of Web Services,* Proc. of the 11th Intl Conf. on WWW, Hawaii, 2002

[154] S. R. Ponnekanti, and A. Fox, *SWORD: A Developer Toolkit for Web Service Composition*, Proc. of the 11th Intl Conf. on WWW, Hawaii, 2002.

[155] S. Shlaer, S. J. Mellor, Object-Oriented Systems Analysis: Modeling the World in Data, Yourdon Press, 1988.

[156] S. W. Ambler, T. Jewell, E. Roman, *Mastering Enterprise JavaBeans*, John Wiley & Sons, 2002.

[157] Semantic Web, http://www.w3.org/2001/sw/.

[158] Simulation Interoperability Standards Organization (SISO), *Guide for Base Object Model (BOM) Use and Implementation*, SISO-STD-003.0-DRAFT-V0.11, SISO, 2005.

[159] Simulation Interoperability Standards Organization (SISO), *Base Object Model (BOM) Template Specification*, SISO-STD-003-2006, 31 March 2006.

[160] Simulation Interoperability Standards Organization (SISO), *SRML Product Development Group*, http://www.sisostds.org/index.php?tg=articles&idx=More&article=441&topics=104.

[161] Software Agents in Wikipedia, The Free Encyclopedia, Date of last revision: 13 December 2007, http://en.wikipedia.org/wiki/Software_agent.

[162] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence, 1995.

[163] T. Bultan, X. Fu, R. Hull, and J. Su, Conversation Specification: A New Approach to Design and Analysis of E-Service Composition, in Proceedings of WWW Conference, 2003.

[164] Tim Berners-Lee, Mark Fischetti, *Weaving the Web*, Chapter 12, HarperSanFrancisco, 1999.

[165] T. Erl, *Service-oriented Architecture: Concepts, Technology, and Design, Upper Saddle River,* Prentice Hall PTR, 2005.

[166] T. Gruber, *What is an Ontology?*, Stanford university pages.

[167] The DIS Vision, *A Map to the Future of Distributed Simulation*, prepared by the DIS Steering Committee, Comment Draft, October 1993, Institute for Simulation and Training, University of Central Florida.

[168] The DIS Vision, prepared by the DIS Steering Committee, Institute for Simulation and Training, University of Central Florida, May 1994.

[169] Transportation Analysis Simulation System, *TRANSIMS*, http://transims.tsasa.lanl.gov/.

[170] U. Saleem, *Developing java web services with AXIS*, 2004, Available via http://www.developer.com/java/web/article.php/3443951.

[171] V. K. Handley, P. M. Shea, and M. Morano, *An Introduction to the Joint Modeling and Simulation System (JMASS)*, Proceedings of the Fall 2000 Simulation Interoperability Workshop, Orlando FL, 00F-SIW-018.

[172] S. Vuong, C. Scratchley, C. Le, X. J. Cai, I. Leong, L. Li, J. Zeng, S. Sigharian, Towards a Scalable Collaborative Environment (SCE) for Internet Distributed Application: A P2P Chess Game System as an Example [online], Available via http://www.magnetargames.com/Technology/DAIS-Vuong-Chess-230603R.doc. Last accessed February 2005.

[173] W. Schneider, *SIMNET, a breakthrough in combat simulator technology*, International Defense Review, No. 4 1989.

[174] W. S. Means, E. R. Harald, *XML in a Nutshell: A Desktop Quick Reference*, O'Reilly 2002.

[175] W. T. Councill, G. T. Heineman, *Definition of a Software Component and its Elements*, Component Based Software Engineering: Putting the Pieces Together, Addison-Wesley, 2001.

[176] W. T. Ng, S. J. Thio, Ch. H. Teo, *A MDA-Based Translation Approach to Component-Level Reuse,* in Proceedings of the Spring 2004 Simulation Interoperability Workshop, Arlington, VA, April 2004.

[177] World Wide Web Consortium, *Resource Description Framework*, http://www.w3.org/RDF/.

[178] World Wide Web Consortium, *Web Ontology Language*, http://www.w3.org/2004/OWL/.

[179] World Wide Web Consortium, *Web services architecture requirements,* http://www.w3.org/TR/wsa-reqs.

[180] World Wide Web Consortium, *Web Services Description Language (WSDL) 1.1*, http://www.w3.org/TR/wsdl.

[181] World Wide Web Consortium, *Simulation Reference Markup Language*, http://www.w3.org/TR/2002/NOTE-SRML- 2002121.

[182] WSDL-S, World Wide Web Consortium Member Submission on Web Service Semantics, http://www.w3.org/Submission/WSDL-S/.

[183] X. Yi and K. Kochut, Process Composition of Web Services with Complex Conversation Protocols: a Colored Petri Nets Based Approach, in Proceedings of Design, Analysis, and Simulation of Dist. Sys. Symposium, 2004.

[184] XVCL Project Homepage, http://xvcl.comp.nus.edu.sg/.

[185] Y. Hu, G. Tan, F. Moradi, *Automatic SOM Compatibility Check and FOM Development*, in Proceedings of 7th IEEE Distributed Simulation and Real-time Applications, Delft, The Netherlands, October 2003.

[186] H. Zhao, N. D. Georganas, *Collaborative Virtual Environments: Managing the Shared Spaces,* Networking and Information Systems Journal, 3(2), 1-23 2003.