PHD DISSERTATION

Strengthening Collaboration, Integration and Modelling in System of Systems Engineering

by Claus Ballegaard Nielsen

AARHUS
UNIVERSITY

Strengthening Collaboration, Integration and Modelling in System of Systems Engineering

Strengthening Collaboration, Integration and Modelling in System of Systems Engineering

PhD Thesis by

Claus Ballegaard Nielsen

Aarhus University Department of Engineering, Denmark

AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

River Publishers

ISBN 978-87-93237-15-5 (e-book)

Published, sold and distributed by:
River Publishers
Niels Jernes Vej 10
9220 Aalborg Ø
Denmark

www.riverpublishers.com

Contents

Abstract

As people and businesses experience the advantages and capabilities enabled by increased connectivity and integration between computer systems, a higher demand arises for the functionality and services that systems must provide. This necessitates the creation of a type of highly complex systems that themselves contain systems, which have a high degree of interconnectivity and interactions between them. This type of system is known as a System of Systems, which denotes a system that itself contains a group of heterogeneous constituent systems that via the interactions between them create a synergistic collaboration through which a higher common goal can be achieved. The development of a System of Systems is challenged by a high degree of complexity in the form of diverse stakeholders, geographical distributed development, a constant system evolution, an unpredictable emergence of behaviour, and the heterogeneity between the constituent systems. These challenges are being addressed in the field of Systems of Systems Engineering. The methodologies and tools used for analysis and development are however still in their infancy. This dissertation is focused on strengthening the Systems of Systems Engineering field by using a combination of Software Engineering and Systems Engineering, specifically with a focus on formal model based engineering techniques and tool-support hereof. The dissertation has three focus areas: managing integration challenges, enhancing the capabilities of System of Systems modelling and enabling collaborative development in formal modelling. The result is: a range of classification dimensions that characterise System of Systems; a System of Systems classification for design patterns; and multiple approaches for improving the tool-support for formal modelling techniques in System of Systems Engineering.

Resumé

Som flere folk og virksomheder oplever fordelene og de forbedrede egenskaber der bliver muliggjort af forbedrede forbindelser og integration mellem computer systemer, opstår der et øget krav til funktionaliteten og de tjenester som systemerne skal levere. Dette nødvendiggør skabelsen af en type meget komplekse systemer, som selv indeholder systemer, der har en høj grad af sammenkobling og interaktion mellem dem. Denne type system er kendt som et System of Systems (System af Systemer), og angiver et system som i sig selv indeholder en gruppe af heterogene konstituerende systemer der via samspillet mellem dem skaber et synergetisk samarbejde, gennem hvilket et højere fælles mål kån nas. Udviklingen af et System of Systems udfordres af en høj grad af kompleksitet i form af mange forskellige interessenter, geografisk distribueret udvikling, en konstant system evolution, en uforudsigelig fremkomst af adfærd, og forskelligheden mellem de konstituerende systemer. Disse udfordringer bliver adresseret af System of Systems Engineering, men de metoder og værktøjer, der anvendes til analysering og udvikling er dog stadig i deres tidlige stadie. Denne afhandling er fokuseret påat styrke System of Systems Engineering feltet ved hjœlp af en kombination mellem Software Engineering og Systems Engineering, specielt med et fokus påformelle metoder og model baseret udviklingsteknikker, samt værktøjer til støtte heraf. Afhandlingen har tre fokusområder: håndtering af udfordringerne i integration, forbedring af mulighederne i System of Systems modellering og at mulig- gøre kollaboration i udviklingen af formelle modeller. Resultatet er: en mœngde af dimensioner der karakterisere System of Systems; en System of Systems klassifikation for design mønstre; samt flere tilgangsvinkler til forbedring af værktøjer der bruges til formelle modelleringsteknikker i System of Systems Engineering.

Acknowledgements

There are a number of people who have been important in the completion of this dissertation, as their support, guidance and assistance made it all possible. To all of these, and many others, who contributed academically or personally, I owe a great deal.

First I want to thank my supervisor Professor Peter Gorm Larsen from the Department of Engineering, Aarhus University. He has had a remarkable impact on both my research and my personal development. I am highly thankful for all he has taught me, both consciously and unconsciously, through his guidance, time and ideas. His never ending support and enthusiasm is both contagious and motivational.

I want to acknowledge my colleagues in the COMPASS project whom have contributed strongly to my research by providing a foundation on which the research could be performed. They have contributed to my research with valuable ideas and feedback, for which I owe them my strongest appreciation.

I also owe a strong recognition and thank you to Kenneth Lausdahl and Nick Battle for they support and work with the tool development efforts. Their impressive technical insights and expertises have been invaluable in the technical aspects of this dissertation.

I want to express my gratitude to John Klein, James Ivers and Ray Obenza from the Software Engineering Institute, Carnegie Mellon University for giving me the opportunity to visit them in Pittsburgh, PA, USA, and for the Foundation Idella for providing funding to support this trip.

My current and past colleagues at the Department of Engineering, Aarhus University has provided me with aid, support and knowledge that has helped me to develop as a researcher and an engineer. It has been a great pleasure working with them and among them. I would especially like to thank Sune Wolff, Joey W. Coleman, Augusto Ribeiro, Luís Diogo Couto, Anders Kaels Malmos, Rasmus W. Lauritsen, José Antonio Esparza Isasa and Peter W.V. Jørgensen for their camaraderie and good spirits.

For their collaboration on joint work, I thank my co-authors; Peter Gorm Larsen, Kenneth Lausdahl, Klaus Kristensen, John Fitzgerald, Jim Woodcock,

Jan Peleska, Rick Kazman, Klaus Schmidt, John Klein, Claire Ingram, André Didier, Stefan Hallerstede, Uwe Schulze and Andrew Galloway.

I would also like to thank Nick Battle for his valuable feedback and review of this PhD dissertation.

I want to thank my parents Alice Nielsen and Anders B. Nielsen for their for love, endless encouragement and support of me in all my pursuits. I also thank my sister Gitte B. Nielsen for her never-ending support and reassurance. Knowing I always have my family to count on is invaluable and something I deeply cherish. Without them, this dissertation would never have been written.

The best thing that has happened to me over the course of my PhD has been meeting my beautiful girlfriend, Luminita Ciocoiu. For her love, encouragement, understanding and patience, I owe my deepest and most sincere appreciation and gratitude. She, more than anyone, knows the work and effort that has gone into completing this dissertation.

Claus B. Nielsen
Aarhus, April 2014

Part I

Summary

1

Introduction

Computers are useless. They can only give you answers
Pablo Picasso *1881 - 1973*

Based on this statement is should be fair to derive that Picasso was neither a systems nor a software engineer. Not because of the assertion that computers are useless, but because the creativity and critical reasoning that Picasso deemed the computers of lacking, actually strives very well among engineers. Any Systems or Software engineer knows that computers can be a source of immense complexity and ambiguity, which lead to the development of systems and computers raising a lot of questions that require creativity and reasoning at a higher level. Picasso was right about computers on their own only supply answers, but what he might have overlooked is that it is not the answers, it is the questions that make the computer a tool for the creative and eager mind. Computers will faithfully supply answers to the questions they are asked, but it is the task of the engineer to ask the computer the right questions.

Computers, in the modern sense of the word, has existed since the late 1940's [214]. The electronic computers were built in university projects with funding from government and military sources, leading to the initial steps of the computer engineering field [107]. In the 1950's established and new companies started to enter the field, initiating a race towards improving and expanding the capabilities of computers. In the late 1950's the change from vacuum tube to transistor based computers was a big leap in terms of higher efficiency and smaller physical size. From the mid-sixties integrated circuits enabled a new generation of computers due to the advantages it delivered in cost and performance. Ever since there has existed a constant push on finding new boundaries for technology in terms of scope, performance, functionality and complexity. This not only applies to the physical computers, but also greatly to the programs the computers execute: the software.

The development of the software field naturally followed the advancement of computers, leading to increasing functionality and complexity. The possibilities and flexibility of software allowed for a near endless number of possible usages and applications. As a result of the growing number of software applications needed, software became products in their own might [119]. As people, machines, and businesses became dependent on the reliability and functionality of software, it raised demands for having methods and tools for designing, building, and maintain this software, eventually leading to the field of Software Engineering [182, 20]. Ever since, the computer and software have gone hand in hand as tools for solving problems and provide value for businesses, governments, as well as everyday life. The technology has moved from taking up a government-run multi-storey house to becoming personal and multi-purpose [31].

It is clear that since Picassos passing in 1973, computers have come a long way. They have become more resourceful [197], more ubiquitous [246], more connected and interacting [7] and more accessible [29]. The continued growth of the sheer number of systems and the expansion of functionality they provide, naturally leads to advances in the field of engineering. It requires more sophisticated development techniques and creates new engineering paradigms and ways of thinking.

One such way, is to think in the way of systems. Thinking of the interactions and connections between components and sub-systems, as well as understanding the different parts in relation to the entire system, and seeing both relations and problems as relevant for the whole [5, 150]. The notion of a ”System“ can be traced all the way back to the ancient Greeks in their attempt to form a political entity [76]. When talking about systems within the field of engineering, a system is a structure or collection of heterogeneous elements that have been joint together in one form or another, in order to produce results that are beyond those of the individual elements [170]. As systems thinking is concerned with the different elements of the system in relation to the whole of the system, it is to a large degree concerned with establishing a mind-set that can combine the world of theory and the world of practice of many different science fields, through the use of interdisciplinary processes [22].

As computers themselves are systems and systems can contain computers, the computer engineering field already has close ties with systems thinking. Having a systems thinking approach to engineering therefore works as a complement to traditional computer engineering, by providing a wider scope than that of pure computational thinking [249]. This allows for an engi-

neering approach where systems are at the centre of attention, leading to an "engineering system's thinking" and consequently the Systems Engineering field [87, 140]. Systems Engineering is concerned with establishing and performing interdisciplinary processes to aid the design, construction and maintenance of complex systems [78].

In the same way as a system can consist of computers, a system can also consist of systems. This has led to a paradigm within Systems Engineering known as System of Systems (SoS), which is a type of system that itself contains groups of heterogeneous constituent systems that through the interaction and synergy between them achieve a functionality for the overall systems that is greater than what can be achieved by the systems separately [22, 168]. As a natural result of the challenges that exist in SoS design and construction, the area of System of Systems Engineering emerged [127]. The SoS Engineering field is however still in its infancy and the field has many challenges that need to be addressed. Since Systems Engineering is focused on creating an interdisciplinary engineering field, it is only natural to search for solutions and methods in other engineering fields, as a way of approaching the challenges. Combining Systems Engineering with the methods and tools used in software engineering [199], business management [216], modelling [95] and collaborative environments [174] has been proposed.

This dissertation provides a summary of the research performed during a PhD study on the use of Software Engineering and particularly modelling in the field SoS Engineering. The research takes its starting point in identifying key SoS characteristics through a survey of the field's literature, which provides a basis for determining future research challenges of Model-Based SoS Engineering (Chapter 2). These challenges combined with observations made from case studies provide the basis for several approaches and tools being developed in order to strengthen the field of SoS Engineering (Chapter 3). Specifically, the research has been performed with a focus on structuring integration challenges, enhancing the capabilities of systems modelling and using collaborative development methods, all within the SoS Engineering field.

The remainder of this introduction will provide a deeper foundation for the dissertations central subjects: Systems Engineering (Section 1.1), Systems of Systems Engineering (Section 1.2) and Modelling and Simulation (Section 1.3). This is followed by an account of the Motivation (Section 1.4), Research Method (Section 1.5), Research Objectives (Section 1.6), and Evaluation Criteria (Section 1.7) for the performed research. Finally, an overview of the Published Work (Section 1.8) of the author is given, before the Outline and Reading Guide (Section 1.9) completes the introduction.

1.1 Systems Engineering

History is full of examples where projects and systems have failed with more or less disastrous results, which have had an impact on both economy and humans [37, 51]. Many of these were a result of growing customer needs, products getting increasingly complex, and a rapid increase in technology and markets. The challenges and failures faced when doing large projects involving complex systems, brought about a need for establishing a better understanding of Systems Engineering. In an attempt to deal with the complexity of systems and enable the realisation of successful systems, Systems Engineering offers tools, processes and mind-sets focusing on the overall system design, from the conceptual design to its disposal. It also focuses on embracing the many diverse fields of engineering often involved in the system life-cycle [108]. As such Systems Engineering is focused on choosing the most appropriate methods and tools for a given systems development project and on applying these to improve the decision-making process and ensuring the correctness of system specification and design [230].

1.1.1 Short History

One of the first records of the "Systems Engineering" term is found in an article from 1949, where it is used to explain the engagements of the U.S. Navy Electronics Laboratory in turning new ideas into practical equipment [43]. A deeper account of the history of Systems Engineering is given by Brill [96] and summarised by Gorod *et al.* [88]. According to Brill's chronicle, the first direct accounts of Systems Engineering originate from the early 1950's in the United States. More specifically, Systems Engineering lectures were given at Massachusetts Institute of Technology, US, by G.W. Gilmam of Bell Laboratories. The focus on the term increased during the 1960's and 1970's, of which a few significant contributions can be mentioned:

The Systems Engineering pioneer A.D. Hall presented a methodology for Systems Engineering which was focused on; acquiring knowledge of the many facets of systems development, concentrating on the customers objectives, and on the systems engineer to examine a system within three areas: physical or technical; business or economic; and social [94]. All with the purpose of establishing the operational, economic and performance goals of the system, to allow for a technical plan to be drafted and executed. Hall presented the question: "Has mankind evolved to a point that there exists, or that with creative additions and re-combinations of modest proportions, there can be shown to be available, a common systems methodology, in terms of which

we can conceive of, plan, design, construct, and use systems (procedures, machines, teams of people) of any arbitrary type in the service of mankind, and with low rates of failure?". This expressed the need and possibilities in a common and collective systems methodology to plan and construct systems on a generic level [103].

G.M. Jenkins advocated for an expansion of the view on systems, from being purely on physical systems to also include stakeholders, such as governments organisations, management and individuals, by describing a system as a "complex grouping of human beings and machines". Jenkins also presented four phases of systems development: Systems Analysis, Systems Design, Implementation and Operation, that covers everything from environment, sub-system interactions and using models to simulate operational conditions [115, 111].

In 1976 A.W. Wymore placed a focus on interdisciplinary teams and the importance of human behaviour in Systems Engineering and presented a methodology that focused on effective communication in teams and management [255]. Wymore was also instrumental in establishing modelling and simulation as way of analysing desired system behaviour, within the Systems Engineering field. His 1967 book "A Mathematical Theory of Systems Engineering" [254] formed the foundation for what three decades later were presented as "Model-Based Systems Engineering" [256].

In 1990, as a consequence of a lack in qualified engineers who were capable of doing systems thinking, 35 individuals from numerous US-based companies and organisations formed the first professional organisation for Systems Engineering aimed at championing the definition and understanding of Systems Engineering [105]. Originally, launched as a US-based organisation it fairly fast expanded beyond the borders of the US, and is now known as the International Council for Systems Engineering (INCOSE). The purpose of INCOSE is to bring together all who have an interest in systems being developed, produced and run more efficiently, and to provide better processes, methods and tools to systems engineers.

1.1.2 Role in the Dissertation

The research described in this dissertation has its foundation in the author's background in Software Engineering and in the modelling of complex systems, as well as the author's interest in interdisciplinary engineering teams and the challenges and possibilities this presents.

As long as Systems Engineering has existed it has had close ties with Software Engineering. This has two reasons: 1) the tools used in Systems Engineering and 2) the amount of software in systems today. An important aspect of the Systems Engineering field is the tools available for supporting the engineering process, and a large part of these tools are developed in software [21].

As technology has developed over time, software takes up a bigger and bigger role in complex systems, for instance in the military industry where software was responsible for 8% of a fighter jet's functionality in the 1960's, a percentage that had grown to 80% in 2000 [62]. This is another reason software is important in Systems Engineering, and this highlights the interesting aspect that computers and systems themselves have become absolutely essential in the development of systems and computers.

The close interconnection entails a tighter integration between the fields of Systems Engineering and Software Engineering [18], and it makes it relevant to investigate how methods of one field can be used in the other.

1.2 System of Systems Engineering

At the time of this writing the Internet search giant Google had just acquired Nest, a manufacturer of mass-market smart home appliances for $3.2 billion[1]. Nest produces products that will enable the "smart home", via devices such as thermostats and smoke alarms, that are connected to the internet. The acquisition gives Google a corner of the growing infrastructure of sensors, devices and systems that are getting interconnected around us. Mobile phones, televisions, cars and household fridges are capable of communicating with other systems, creating a highly complex system of interconnectivity and interactions. The advantages and possibilities of connectivity between systems that people and businesses are experiencing, give constant growing demands for the functionality and services that systems must provide.

Machine to Machine interaction and connectivity between computing systems have practically been on the technological research agenda for nearly as long as the computer itself. In the same way as for computers, there has been a constant push for developing and improving computer communication [250]. Evolving from local networks over national to international and intercontinental networks, it initially had a military and academic purpose [57]. The

[1] Google press release: Google to acquire Nest `https://investor.google.com/releases/2014/0113.html` - Last accessed 19-04-2014.

further development of computer networks led to many important contributions to the field of computer systems, arguably the establishment of the internet as the most significant [157]. The capabilities and value of computer networks have resulted in systems becoming increasingly more connected and interacting, progressing to a degree where they are considered as part of a distributed system [233]. In a distributed system multiple computational entities communicate in order deliver and make use of services that can solve computational problems. The participating systems can range from being high-end servers to a mobile phone or small sensor. The development in the field of computers has naturally led to further advances of distributed systems with some of the key subjects currently being: ubiquitous computing [246, 220], cloud computing [240, 260] and "Internet of Things" [7].

The growing demands for interconnectivity have establish a need for large-scale and complex systems to be engineered. Many of these interconnected distributed systems can be regarded as SoS as they: 1) express a large degree of autonomy as a result of having diverse stakeholders; 2) they are independently designed and not directly meant to collaborate with other heterogeneous systems; 3) are separated by great geographical distances; 4) have detached development cycles and evolution of functionality; but 5) have an overall common goal that is dependent on the synergy between them.

Examples of System of Systems are often information intensive systems, such as Air Traffic Management [85], Emergency Management [159], or more concrete the Global Earth Observations System of System [28] which is a decision-support tool linking the information from the space agencies of 88 nations.

The many development facets of these systems have pushed the engineering effort towards its boundaries, which consequently led to the SoS Engineering field. SoS Engineering covers the processes and practices in the analysis, design and development of SoS, where especially the dynamism and emergent operational behaviour, combined with the integration of distributed independent systems are key challenges [127].

1.2.1 Short History

The term "System of Systems" was first used in the mid 50's in relation with system theory and the arrangement of theoretical systems [22]. It was not until the late 1980's that the term was used to describe the challenges and characteristic arising in relation to joining independent systems together [236].

From here the field saw an increase in both the industrial and academic research taking place. Based on the publications in the literature this increased during the 1990's reaching an all-time high midways in the first decade of the new millennium [128]. The field of SoS Engineering was not formally established until 2003 [127].

A more detailed historical account is given in Section 2.1.

1.2.2 Role in the Dissertation

SoS Engineering is a cross-disciplinary engineering field that has to address the challenges inherent in developing a type of system that express both technical and socio-technical challenges. As such it is a field that has many challenges, of which many remain unsolved as a result of its youth. This makes it a field with a large potential for the curious engineer to research and develop methods and tools that can solve some of these challenges.

During this PhD project the author was working together with a research group from Aarhus University that works on enhancing existing industry tools and practices in SoS engineering, through the EU Seventh Framework Programme research project: "Comprehensive Modelling for Advanced Systems of Systems"(COMPASS)[2]. The COMPASS consortium consists of five universities and three companies that research the field of SoS in order improve the engineering thereof by using a combination of Model-Based SoS Engineering and guidelines for SoS requirements, architectures and integration.

1.3 Modelling and Simulation

The use of modelling and simulation as a means of approaching and handling complexities is instrumental to most fields of engineering.

Models are some type of representation of an object that they are meant to replicate or express some part of. Generally speaking, models have the purpose of enhancing the understanding of certain aspects of the object it represents and can be used to facilitate and document details and specifications [162]. As such models are often used as a point of reference and for communicating design specifics to stakeholders. Models enable engineers to describe designs and concepts in a way that make it easier to study the functionality and complexities of the modelled object. Therefore, they allow

[2] `http://www.compass-research.eu/` Last accessed 14-04-2014.

engineers to discover possible design flaws and provide better predictions for architecture and design qualities [99]. Models can have many types of representations, such as: physical, mental, mathematical or graphical portrayals of the object they represent.

A simulation is an estimated imitation or reflection of the behaviour that a simulated object will have in a given situation. A simulation is performed on the basis of a model, and it can be used by engineers to show the operation and performance of a modelled object [258].

Modelling and simulation is used in Systems Engineering, often under the term Model-Based Systems Engineering (MBSE), to support system requirements and design as well as perform analysis, verification and validation throughout the development of the systems [256]. Models in Systems Engineering is for instance used to: 1) depict physical systems in which physical laws often play an important role; 2) to describe a process or an algorithm in which events occur in a step-by-step fashion; or 3) represent structures and relationships [26]. The first is denoted continuous systems and are typically described using differential equations, while the second is known as discrete-event which describes a sequence of events in time [179, 259]. The third is generally described in a graphical modelling language, such as SysML [232].

In the context of this dissertation a model is a computer model described in a textual notation that has a mathematical foundation and is used for describing discrete-event systems.

It has been proposed that Systems Engineering should adopt the rigorous modelling and the development of models from the Software Engineering field [199]. In the software engineering field models are created of software design using mathematically founded techniques unified under the term "formal methods". Mathematically founded modelling languages are used to introduce a higher degree of rigor and reliability as means of dealing with the complexity and unpredictability of software systems. The languages have unambiguous semantics, such that inherent properties about the models can be reasoned about. The development of formal methods and their application in development projects have been widely encouraged and extensively researched in both academia as well as in industry [38, 253].

Formal methods are highly suitable in the engineering of SoS characteristics such as autonomy, evolution and emergence. In connection with both system specification and design these are all aspects that involve a great degree of vagueness and unpredictability. A stronger analysis of these aspects can be achieved by using formal methods. In recent years the application of formal methods in SoS development has been recommended multiple times

in the SoS literature [30, 58]. This has led to research and development in concrete formal SoS modelling notations [251, 86, 73].

1.3.1 Short History

Giving a historical account of the general field of modelling and simulation is difficult because of the many diverse fields in which it is used and because of the wide range of approaches and methods that the term cover. Consequently, the account given here focuses on giving a brief overview of computer modelling and simulation aimed at Software Engineering and Systems Engineering.

Some of the earliest applications of computers were used to perform simulations of models within the development of military systems during World War II. The use of modelling and simulation was expanded to non-military application in physics and engineering in the early 1950's [173]. Quickly thereafter computers were being used for creating models and simulations of computers themselves [219]. Models and simulations of computer software got a firm base with the establishment of formal methods. In nearly five decades formal methods have been used in the analysis and validation of software systems [74]. Around the same time mathematically based approaches for Systems Engineering were being developed [254]. Later on, development methods in which modelling and simulation was at the core of the engineering effort started to emerge. These matured into development methods such as Model Driven Development [77] and Model-Based Systems Engineering [256].

A deeper explanation of the history of modelling and simulation is outside the scope of this dissertation, but more information on modelling in Systems Engineering has been covered by Dickerson [55]. A more detailed account on the use of modelling in SoS Engineering is given in Section 2.3

1.3.2 Role in the Dissertation

The author has a background in the formal modelling and simulation of software systems, and the use of modelling and simulation has been recommended as an effective means for grasping the complexity of SoS [202]. This forms the basis for a key part of the research described in this dissertation.

The research makes use of two formal modelling notations: the Vienna Development Method and the COMPASS Modelling Language, both are described in further detail below. The two notations are supported by two open-

source tools which give this PhD project a basis on which approaches can be build and tested. The tools are described in further details in Section 3.1.

1.3.3 Vienna Development Method - VDM

The Vienna Development Method (VDM) is a well-established formal method for specifying, modelling, and evaluating software systems [15, 120, 67]. It has its origin in the Vienna Definition Language (VDL) developed at the IBM laboratories in Vienna in the beginning of the 1970's. Through the development of the process, with the addition and combination of multiple techniques, the approach was defined and named as the Vienna Development Method in 1973 [121, 16]. Since then VDM has been applied succesfully to a range of industrial projects [70].

VDM models builds on type definitions that are constructed from simple abstract types, such as booleans, natural numbers, characters, as well as type constructors for union, product set, sequence and map types. Types can be restricted by predicate invariants that are enforced by performing run-time type checks via tool-support.

Typed variables can be used to establish a persistent state, that also can be restricted by invariants, and operations for modifying the state can be defined implicitly using standard pre- and post-condition predicates or explicitly using imperative statements. Additionally, functions can be defined in a fashion similar to operations, but cannot refer to state variables.

The VDM Specification Language (VDM-SL) notation has a formally defined syntax and language semantics defined in an ISO Standard [206, 142, 149]. VDM-SL is a language that primarily is aimed at modelling functional specifications of sequential systems [68]. In order to meet new technology and the latest industrial challenges VDM has developed over time by introducing several new language dialects with extended functionality. VDM++ is an object-oriented extension of VDM, in which the models consists of collections of classes [69]. The research project "VDM++ In a Constrained Environment" (VICE) introduced a timed extension to VDM++ in order to model real-time systems with respect to time. Research revealed that neither the existing VDM++ dialect nor the extension made with VICE was sufficient when modelling distributed real-time systems [241]. As a result an extension was proposed to enable the modelling of distributed real-time embedded systems in VDM++ [244, 106]. The extension introduced the notion of CPUs, busses, specific time delays and asynchronous operations. The extension made it possible to deploy individual distributed systems on separate

CPUs, which could by connected by busses. The extension named VDM-RT (as a replacement of the VICE notation) was implemented and validated by multiple case studies [66, 229, 242].

The existing VDM dialect are:

VDM-SL an ISO standardised sequential language for defining software functionality.

VDM++ which includes the fundamental functionality of VDM-SL but extends it with concurrency and object oriented design.

VDM-RT which extends VDM++ by adding timing constraints, CPUs and busses as well as distributed system design and topology.

In their foundation these languages are not executable, as they allow implicitly defined functions and operations as well as types with infinite domains, however an executable subset exists for all dialects making it possible to interpret and simulate them [146, 148].

To ensure the validity and consistency of a system specification VDM models can be validated by analytic methods, ranging from type checking to execution of the model, via tool support. Further details are given in Subsection 3.1.1.

1.3.4 COMPASS Modelling Language - CML

A key goal of the COMPASS project is to strengthen Model-Based SoS Engineering by using formal modelling to precisely describe the complex structures and interactions of SoS [71]. This is achieved by creating a language called the COMPASS Modelling Language (CML), which has a formal semantic foundation through which SoS architectures and contracts can be expressed and simulated.

CML combines the strongest elements of the VDM state-based formal method [68] and the CSP/Circus process-based formal methods [213, 198] to create a language capable of expressing both the structure, behaviour and state information of SoS and their constituent systems.

In its foundation, the COMPASS Modelling Language (CML) is built around collections of types, values, functions, operations, classes, processes and channels. The type system as well as the values, functions and classes originate from VDM, while channels are taken from CSP/Circus. The *process* is the main building block in CML and it acts as the constituent system

in the SoS. An SoS is formed by processes getting connected via channels. *Channels* are globally defined and they are used for message passing. Processes can react to activities taking place in the overall system model by reacting to events. An *Event* means a communication occurring on a channel that processes can synchronise on. State and operations for modifying the process state can be defined within a process, and so can *actions* that are used to express the reactive behaviour of a process, such as communicating on a channel.

An example of a very basic CML model is given in Listing 1.1. This shows the definition of a channel c that can carry a nat type, and three small systems represented by the processes P, A and B. Process P is a composition of two processes A and B. In this model, process P is the one that composes the system at the highest level and as such it can be seen as the one describing the SoS. This is known as the *top* process of the model.

```
channels
 c : nat

process P = A [|{c}|] B

process A =
 begin
  @ c.1 -> Skip
 end

process B =
 begin
  @ c?x -> Skip
 end
```

Listing 1.1: Parallel system of A and B.

The top process P defines a parallel composition between the processes A and B, meaning in this model they can be considered as two constituent systems running in parallel. When making a composition, a range of channels can be given to denote the channels that the composed process can synchronise on, based on the events occurring on the channels. In the given example the composition defines that the processes will synchronise on events occurring on channel c. Process A and B have a very simple functionality, where A is a system that defines an action (indicated by @) to synchronise the value

1 on channel c and then terminates its execution, while B is a system that synchronise (indicated by ?) a value placed on channel c, assigns it to x and terminates its execution.

CML is supported by the open-source Symphony tool platform that builds directly on the Eclipse IDE platform [41] providing project explorer, editor, interpretation and debug functionality in addition to a range of plug-ins, such as a model checker, a proof obligation generator and a theorem prover.

1.4 Motivation

Although the "System of Systems" term has been appearing for more than five decades, the engineering of the field is still in its early stages. With an increased focus on the field there has been a rapid growth in research and it has created a voluminous literature [128]. This has led to a rich set of descriptions and SoS characteristics that makes use of a substantial vocabulary, but there is still a lack of a common ground for a definition and understanding of SoS. The increased focus on the field from both academia and industry has also led to the creation of an SoS Engineering field and the study of this field has resulted in a large number of research challenges being identified [127, 211, 237].

As any other field of engineering, SoS Engineering has a need for approaches and tools that engineers can use to approach the design and construction of SoS in the best possible way. A range of methods, processes and approaches has been transferred from other engineering fields, such as "Software Engineering" and "Systems Engineering", in order to create a strong foundation for the field [126]. One such example is the use of systems modelling and in recent years the application of formal methods in SoS development has been proposed multiple times in the SoS literature [30, 58]. New research projects have been started that are aimed at providing tools and methods with a basis in formal modelling for the engineering of SoS. However the application of formal modelling techniques will face the same challenges with industrial adoption in the field of SoS Engineering as it has had it other fields [132].

Experiences from the formal modelling field does show that strong tool support of formal techniques and proper incorporation into existing development practices, greatly improve the use and acceptance of formal methods [130, 245].

The overall motivation of this PhD project is a wish of getting a deeper understanding of the challenges in the SoS engineering field and to improve

the approaches and tools used in SoS engineering, especially with a focus on formal modelling.

1.5 Research Method

This section describes the research methods that has been applied in the PhD project:

This PhD project focused on identifying challenges of the SoS Engineering field through a combination of literature surveys and case studies. Once a challenge of particular interest was identified, the existing literature was surveyed for similar challenges and any corresponding solutions. In some cases the solutions identified through the survey were adapted or extended in order to conform to the concrete challenge, while in other cases a new solution was developed. In order to evaluate a proposed solution they were, in most cases, followed by a case study as a proof of concept. Using the case study the solution would be adjusted and improved until the solution was shown to provide a contribution to the identified challenge. Contributions were then submitted as publications to either workshops, conferences or journals for peer-review and validation of results.

The basis for the PhD project was established through a survey of the SoS literature with a focus on SoS definitions and Model-Based engineering techniques for SoS Engineering. Additional input was gained through smaller case studies and from the observations and input from the industrial case studies in the COMPASS project. This knowledge was used to develop methods, classifications and tool support aimed at improving the SoS Engineering field.

1.6 Research Objectives

The objectives of this PhD project is to improve the field of SoS Engineering by contributing to a strengthened SoS understanding, with classifications aimed at certain SoS challenges for which stronger tool-support for formal modelling can be developed.

It is the central proposition of this PhD Project that:

The SoS field consists of a large number of building blocks, from which it is still trying to create a foundation on which it can establish itself, and as a developing field it has the challenges of attempting to incorporate, expand and evaluate methods, processes and tools from other engineering fields,

while trying to develop SoS Engineering specific techniques as well. As such the SoS Engineering field has not addressed all of its challenges and is still lacking the methods and tools support need to create a strong foundation and improved engineering of SoS.

It is the hypothesis of this PhD that:

Convergence of the existing knowledge in the SoS field can be used to identify key challenges in SoS, and the field of SoS Engineering can be strengthened by introducing formal modelling techniques with strong tool-support focused on these key SoS challenges.

It is the hope that the research results from this PhD will aid the developers in SoS Engineering in asking the right questions such that the computers can supply useful answers.

1.7 Evaluation Criteria

The research contributions made in this PhD project will be evaluated using the follow criteria:

Clarification of the SoS field A clarification of the characteristics must be provided that makes it possible to specify key dimensions for SoS. A clarification will make it easier to approach the SoS Engineering field and identify primary challenges.

Integration The integration and interactions between constituent systems in SoS shall be studied to identify specific challenges, and methods must be provided that enhance the SoS Engineering approach to integration.

Collaboration The role of human stakeholders in SoS Engineering shall be studied and methods shall be developed that can address the challenges identified.

Modelling The application of formal modelling in the SoS Engineering field shall be examined and barriers for adopting formal models into the field shall be addressed.

Improved tool-support for SoS Engineering Approaches for stronger tool-support that will strengthen the use of formal modelling in SoS Engineering shall be developed.

The evaluations of the research contributions are described in Section 4.3, where the level of fulfilment of the individual criteria is assessed. The extent to which the criteria is fulfilled will be illustrated using a chart as exemplified in Figure 1.1, which shows the research contributions and the fulfilment of criteria.

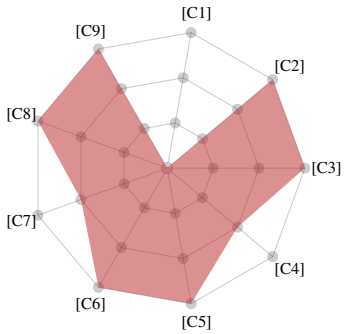


Figure 1.1: Example comparison chart.

1.8 Published Work

The research and work performed in this PhD project has resulted in a number of publications. Subsection 1.8.1 contains the publication that forms the foundation of the PhD project and is included as part of the dissertation. Subsection 1.8.2 lists publications that are related to PhD project, but has not been included in the dissertation. Finally publications that are not related to the PhD project are listed in Subsection 1.8.3.

1.8.1 Publications

The publications listed here are included in this dissertation in Part II.

[P190] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. *Combining VDM with Executable Code*. In Abstract State Machines, Alloy, B, VDM, and Z, volume 7316 of Lecture Notes in Computer Science, pages 266– 279, 2012. Springer-Verlag. ISBN 978-3-642- 30884-0.

[P187] Claus Ballegaard Nielsen and Peter Gorm Larsen. *Extending VDM-RT to Enable the Formal Modelling of System of Systems*. Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012, July 2012. 978-1-4673-2974-3

[P189] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock and Jan Peleska. *Model-based Engineering of Systems of Systems*. Submitted to ACM Computing Surveys, September 2013.

[P125] Rick Kazman, Klaus Schmid, Claus Ballegaard Nielsen and John Klein. *Understanding Patterns for System of Systems Integration*. 8th International Conference on System of Systems Engineering (SoSE), IEEE SoSE 2013, 141-146, June 2013.

[P186] Claus Ballegaard Nielsen, Claire Ingram, André Didier, Uwe Schulze, Stefan Hallerstede, Andrew Galloway and Peter Gorm Larsen. *Challenges in Collaborative Formal Modelling of System of Systems*. Draft paper to be submitted to the International Journal of System of Systems Engineering, 2014.

[P188] Claus Ballegaard Nielsen and Peter Gorm Larsen. *Collaborative Formal Modeling of System of Systems*. IEEE SysCon 2014, March 2014. IEEE Best Student Paper Award

[P192] Claus Ballegaard Nielsen, Kenneth Lausdahl and Peter Gorm Larsen. *Distributed Simulation of Formal Models in System of Systems Engineering*. 4th IEEE track on Collaborative Modelling and Simulation in IEEE WETICE 2014, June 2014.

[P154] Kenneth Lausdahl, Claus Ballegaard Nielsen and Klaus Kristensen. *Including Running System Implementations in the Simulation of System of Systems Models*. Submitted to Software Engineering and Formal Methods (SEFM) 2014, March 2014.

1.8.2 Other Publications

[P191] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. *Using the Overture Tool as a More General Platform*. In Franco Mazzanti, editor, iFM 2012 & ABZ 2012 - Proceedings of the Posters & Tool demos Session, pages 1–34. CNR-ISTI, June 2012.

[P124] Rick Kazman, Claus Ballegaard Nielsen and Klaus Schmid *Understanding Patterns for System-of-Systems Integration*. Technical Report, CMU/SEI-2013-TR-017, Software Engineering Institute, Carnegie Mellon University, 2013.

[183] Claus Ballegaard Nielsen. *Towards Dynamic Reconfiguration of Distributed Systems in VDM-RT*. Proceedings of the 8th Overture Workshop: Semantic Issues in VDM: a BCS-FACS and Overture Workshop, September, 2010.

[184] Claus Ballegaard Nielsen. *Modelling Dynamic Topologies via Extensions of VDM-RT : A Case Study of an Evolving System*. Aarhus University, School of Engineering, Technical Report, ECE-TR-9, July, 2012.

1.8.3 Publications Outside the Focus of PhD Research

[42] Joey W. Coleman, Anders Kaels Malmos, Claus Ballegaard Nielsen and Peter Gorm Larsen. *Evolution of the Overture Tool Platform*. Proceedings of the 10th Overture Workshop 2012, School of Computing Science, Newcastle University, 2012.

[185] Claus Ballegaard Nielsen, *Utilizing VDM Models in Process Management Tool Development: an Industrial Case*. Proceedings of the 9th Overture Workshop, Electrical and Computer Engineering, Aarhus University, ECE-TR-2, 2012

[110] José Antonio Esparza Isasa, Peter W.V. Jørgensen and Claus Ballegaard Nielsen *Modelling Energy Consumption in Embedded Systems with VDM-RT*. Proceedings of State Machines, Alloy, B, VDM, and Z 2014 (ABZ 2014), April 2014.

1.9 Outline and Reading Guide

The dissertation is divided in two parts: Part I contains an introduction and gives a summary of the research that has been performed during the PhD project. The part also supplies an overview of the research contributions on the basis of the publications that has been produced during the project. Part II contains a subset of publications by the author with co-authors, on which the research contributions are based.
For clear identification, contributions are numbered e.g. [C1], and framed:

Contribution 1: Description

In addition to an introduction to the research field, Part I gives an overview of the performed research on the basis of publications. It 1) introduces background information on the premise and reasoning behind the performed research and 2) provides a summary of the publication, and 3) it draws lines to related work and state of the art. Part I introduces ten publications, eight of which are included in the dissertation and described in more detail. For easier identification these ten publications are prefixed with "P" e.g. [P189].

Part I is structured as follows; after this introductory chapter, Chapter 2 gives an overview of the research that has been performed in exploring SoS and supplies details on the three main themes of this dissertation: modelling, integration and collaboration. The chapter presents the publications: [P189, P125, P186] and contains the contributions that primarily focuses on the identification, challenges, classification and structuring of the SoS engineering field. Chapter 3 introduces the tool-support that has been developed in order to improve and strengthen the SoS engineering field on the basis of the challenges identified in Chapter 2. This chapter is based on the publications: [P187, P188, P192, P190, P154]. Finally, Chapter 4 draws conclusions and discusses research contributions as well as future work.

Part II, lists a range of publications written by the author of this PhD thesis in collaboration with co-authors. Each chapter presents the bibliography entry for one publication, followed by the publication in its original published or submitted form. Part II contains the publications: [P190, P187, P189, P125, P188, P192, P154, P186].

2

System of Systems Research

This chapter summarises the research that has been performed in exploring the System of Systems (SoS) field and how the outcome of the exploration has been used to identify challenges in the field. The chapter also identifies and details the three main themes of this dissertation: modelling, integration and collaboration. Finally, a breakdown of the research that has been performed within these themes during the PhD project are provided.

To provide the formal basis for the research, a survey of the existing SoS literature was performed in order to establish a better understanding of the concepts, terminology and application of the field [P189]. A historical perspective on SoS is given in Section 2.1, followed by an identification of key dimensions that characterise SoS in Section 2.2. The identification of the dimensions made it possible to establish many of the challenges in SoS Engineering, specifically with a focus on the use of Model-Based techniques, as presented in Section 2.3.

In addition to having a focus on Model-Based techniques, the identification of challenges in SoS Engineering and Model-Based techniques was also combined with the experiences gained through the work performed in analysing and modelling SoS. This led to the research expanding into the integration challenges between constituent systems and into the socio-technical area of collaboration between engineering teams. The research performed on the former is presented in Section 2.4, while the latter was examined through a case study focused on the collaborative aspects of formal modelling in SoS Engineering, as presented in Section 2.5.

2.1 System of Systems Historical perspective

The field of SoS is still in its infancy and as an emerging field there is a great deal of diversity and varying foci when it comes to definitions, development methods and applications in the field [114].

In [P189] we performed a survey of the SoS literature, which were focused on the many attempts of defining and characterising SoS. The survey studies 30 different definitions, characterisations or taxonomies for SoS, and attempted to group the meanings on the basis of their focus (e.g. SoS terminology) or the background (e.g. Military). The survey showed that the literature offers a rich set of descriptions of SoS characteristics and make use of a substantial vocabulary with many either overloaded or ambiguous meanings and classifications. This suggests that it is possible to draw fine distinctions between various type of SoS applications, but also that many of the entries share common terms, but use them for different concepts, or share concepts but use different terms for them. This ambiguity and imprecision presents an obvious risk for misunderstandings. The survey is used to create a set of dimensions that can be used to group and describe the many terms. These dimensions are described in further details in Section 2.2.

The survey takes its start from the first mentioning of "System of Systems" in the technical literature, and despite the field still being emerging the idea and notion of SoS has been around for a substantial time. The first citing of the "System of Systems" term is used in the mid 50's by one of the pioneers of General Systems Theory, K.E. Boulding [22]. Boulding's classification differs from the more modern SoS classifications by being focused on using a hierarchy of complexity for the arrangement of theoretical systems and constructs. As such it is focused on creating structures for analysis of methodologies and theoretical systems, not on the relationships between constituent systems nor their operational behaviour as part of a larger system. Boulding's classification does however still contain elements that bear a strong relation to the more modern descriptions of SoS. The system's anatomy has a dynamic dimension that changes over time, the systems can adapt on the basis of the information it receives and there is a division of labour between the systems that are both differentiated and mutually dependent.

With the basis for SoS laid down early, the use of the term and the way it was applied took various directions in the following 30 years. It was used in such diverse fields as urban city planning [14], the structuring of systems science [4] and for most systems found in biology [112]. The establishment of SoS as an engineering concept with relations to joining independent systems together, came with the United States' Strategic Defense Initiative (SDI) [236] in the late 1980's. A concrete establishment of the field of SoS Engineering took another 15 years [127], but in the decade that followed the field got an increased awareness and the SoS research intensified in both industry and academia.

Numerous entries in the literature gives definitions and descriptions of SoS, either seen from a theoretical point of view or as part of an introduction to a concrete systems engineering problem, but a central milestone in the definition of SoS came with M. W. Maier's paper on "Architecting Principles for Systems-of-Systems"[1] [167]. In performing the survey it became apparent that this paper remains one of the most cited in the SoS literature. With a lack of a shared agreement on a SoS definition and characteristics, the paper attempts to use five principal features to characterise SoS, sometimes referred by the acronym "OMGEE".

Operational Independence Any system that is part of an SoS is independent and is able to operate serviceably if the SoS is disassembled.

Managerial Independence Despite collaborating with the other members of the SoS, the individual systems are self-governing and individually managed so that they "not only can operate independently, they do operate independently."

Geographic Distribution The parties collaborating in an SoS are distributed over a large geographic area. Although the geographic area is defined vaguely, it is stressed that the collaborating systems can only exchange information and not considerable quantities of mass or energy.

Evolutionary Development An SoS' existence and development are evolutionary in the sense that objectives and functionality can be under constant change, as they can be added, modified or removed with experience. Thus an SoS never appears completely formed.

Emergent Behaviour Through the collaboration between the systems in an SoS a synergism is reached in which the system behaviour fulfils a purpose that cannot be achieved by, or attributed to, any of the individual systems.

In Maier's terms, SoS are distinguished from monolithic systems by the constituent systems' independence, and the evolutionary nature and emergent behaviour of the SoS as a whole.

Table 2.1 gives an overview of some of the key historical entries in the literature with a point of view on SoS definition and characterisation.

In the years that followed many authors used Maier's five principle features as a reference and point of departure for describing and exploring SoS. From the mid 90's the number of publications related to both characteristics

[1] In the SoS literature, [168] is highly influential and widely cited. However, the characteristics originate in [167] which also exists in a white paper version published online [169]. All these versions share the title "Architecting Principles for Systems-of-Systems".

Author(s)	Main characteristics
Boulding 1956 [22]	Static structure of "open systems" from different disciplines
Ackoff 1971 [5]	System science organising systems into structured framework
Eisner *et al.* 1991 [61]	Meta-systems engineering framework combining independent systems
Shenhar *et al.* 1994 [226]	Taxonomy with technological uncertainty and scope
Noam 1994 [196]	Telecommunication infrastructure moving from "network of networks" to an SoS
Manthorpe 1996 [172]	Focus on the jointness between C4I in a defence setting
Maier 1996 [167]	Most influential paper defining the OMGEE characteristics of SoS

Table 2.1: Initial historical literature, up until Maier's SoS characteristics paper [168].

and the engineering of SoS started to increase and in the middle of the decade that followed the number increased rapidly. This development has also been shown by Kemp *et al.* [128], as illustrated in Figure 2.1.

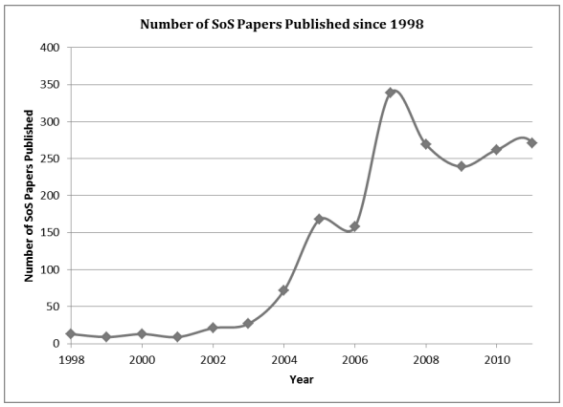


Figure 2.1: Re-produced from Kemp *et al.* [128]

Table 2.2 shows some of the key literature in SoS research from the mid 90's and into the first decade of the new millennium.

Author(s)	Main characteristics
Kotov 1997 [133]	Large-scale concurrent and distributed systems
Lukasik 1998 [164]	The importance of educating of engineers to deal with evolving self-organising systems
Roe 1999 [212]	Systems engineering process for military SoS
Cook *et al.* 1999 [44]	SoS as a systems methodology for military systems with concerns for hierarchy, emergence, and C2
Krygiel 1999 [138]	Focus on interoperability of information and data sharing
Pei 2000 [204]	SoS as a defining factor in future battlefield scenarios
Carlock *et al.* 2001 [33]	Enterprise Systems Engineering point of view
Sage *et al.* 2001 [215]	Use of the strategy "new federalism" for organisational structuring
Chen *et al.* 2003 [35]	Focus on the SoS environment with a core in architecture interoperability and dynamic behaviour
Keating *et al.* 2003 [127]	SoS as a meta-system of interrelated complex subsystems
Bar-Yam *et al.* 2004 [11]	Derives SoS characteristic from the fields military, biology and sociology
Crossley 2004 [47]	SoS as a multidisciplinary research in interoperability, individual behaviour and human behaviour
De Laurentis et al. 2005 [54]	Three dimension taxonomy for SoS analysis and design
Abbott 2006 [1]	Open at the top, open at the bottom and continually evolving, but slowly
Boardman *et al.* 2006 [17]	The alphabet characteristics: autonomy, belonging, connectivity, diversity and emerging
Boehm 2006 [18]	Software-Intensive SoSs
Cocks 2006 [39]	An SoS may not be as much about the system mission, but about the architecture of the selected solution
Fisher 2006 [63]	Composition of autonomous systems is an SoS
Sharawi *et al.* 2006 [225]	Independence, interoperability and global goal are essential SoS concepts for modelling and simulation
Karcanias 2010 [122]	Evolution of the notion of Composite Systems to include autonomy and independence.

Table 2.2: Key literature around the new millennium.

Of these it is worth taking a note of Boardman *et al.* [17] that presented the alphabet characteristics: Autonomy, Belonging, Connectivity, Diversity and Emerging as a way of characterising SoS.

R. Abbott that takes a slight different view than the majority of the literature, by seen SoS as an environment in which other systems reside, instead of a hierarchy of component systems. This shows that even with the rapid

increase of literature, a degree of variation and discrepancy still exists within the field with respect to terminology and terms.

Nearly, all of the newer publications dated after the Strategic Defense Initiative (SDI) [236] in 1988, does however still have the same fundamental traits of SoS. For instance the autonomy and diversity characteristic by Boardman *et al.* maps to Maier's independence and evolution characteristics. In the same way Abbott describes SoS as being continually evolving and as consisting of a collection of systems that participate when the investment is likely to be worthwhile, which also can be related to Maier's independence and evolution characteristics. This is the same for all of the characteristics and the pattern repeats throughout the literature. It is this pattern that the survey attempts to influence by creating the dimensions of SoS Engineering.

The survey showed that some authors have a focus on making a definition for SoS through a textual description of its behaviours and characteristics. Others take the standpoint that the concept is to complex and widespread for a one or two line description and instead attempt to create a taxonomy of terms that can characterise an SoS. A third group of authors focused more on the challenges and possibilities of establishing an SoS. Common to them all is that SoS Engineering is made difficult by: 1) the large degree of uncertainty caused by the autonomy and independence of the constituent systems, 2) the distribution and diversity of stakeholders with separate main goals, 3) the interoperability and integration challenges between the constituent systems, 4) challenges in dealing with a constant changing and evolving systems, and 5) the unpredictability of emergent behaviours occurring as the constituent systems start working together.

Literature surveys of the SoS field have been performed numerous times, either with the purpose of defining and characterising SoS or to identify the challenges and future research goals [126, 113, 225, 141, 88, 114]. For example, C.B. Keating looks at a minor part of the literature to determine five points of convergence for SoS Engineering [126]. Keating finds that the field is concerned with the integration of multiple independent systems in order to create a higher level system that has capabilities and goals beyond that of the independent constituent systems. M. Jamshidi lists numerous textual descriptions of SoS definitions from various sources, but does not correlate them or provide a direct point of convergence[113, 114]. Sharawi *et al.* propose a taxonomy based on the independence of operation and development of heterogeneous constituent systems in which characteristics either are considered

essential or desirable with respect to the modelling and simulation of SoS. The concepts independence, interoperability and a global goal are essential, while a characteristic such as distribution is considered desirable.

Categorizations of SoS

The US Department of Defense makes use of four different categorisations for SoS [202]. These are divided into:

Directed The SoS is built to fulfil specific purposes. They have the ability to operate independently, but they are managed to satisfy a concrete purpose.

Collaborative The SoS are not compelled to follow a central management, instead the constituents system voluntarily participate in a collaboration to fulfil the goal.

Acknowledged The SoS recognises a common purpose and goal, while the constituent systems retain independent control and objectives. Evolution of the common purpose is based on collaboration between the SoS and the constituent systems.

Virtual The SoS is without either managerial control or a common purpose. This makes the behaviour and the fulfilled goals highly emergent, but it also entails that the exact means and structures producing the system functionality are intangible and indistinguishable.

Three of these categories were originally defined by Maier [167], while the "Acknowledged" type was later added by Dahmann *et al.* [48].

The "Directed" SoS practically embodies a form of planed emergence, because all constituent systems are centrally managed. The other types of SoS have little or no centralised managerial control. The "Collaborative" type has the notion of a centralised management but with very limited or no powers to enforce decisions, while the "Virtual" type is without any degree of management. The "Acknowledged" type describes a scenarios found in many military systems, where there is a focus on establishing collaborative management at the SoS level, while keeping the managerial and technical independence at the constituent level. The goal is that autonomy and ownership is maintained, while at the same time ensuring that changes can be collaborative decided upon on the basis of some common objectives.

The work presented in this dissertation mainly focuses on the Collaborative and Acknowledged category of SoS.

2.2 Dimensions of SoS Engineering

As revealed by the survey, presented above, the literature makes use of a wide range of terms and concepts in the description of SoS, leading to a substantial vocabulary. This makes it difficult to grasp the field and it makes it difficult to place systems and applications within the field.

2.2.1 Research Conducted

In [P189] we grouped and converged the terms and concepts into eight dimensions that we proposed as a way of positioning a systems engineering problem in the SoS field. In combining the terms, the contexts in which they were used, were taken into account. For example the term "independence" may denote independence of operations for self-governing constituents [167, 47], the independence of capabilities [123], meaning the variances in resources of constituents, or it can refer to independent optimisation of the constituents [215]. The mapping from the dimensions to the literature in which the terms originate is listed in [P189]. The eight dimensions are listed in subsections below.

Autonomy of Constituents

Autonomy is the extent to which a constituent system's behaviour is governed by its own rules rather than by others external to the constituent. The property of managerial independence identified by Maier entails that constituents perform their own functions in accordance with their own rules, while also participating in the SoS. Cook acknowledges the need for independence in Maier's sense, and also identifies a requirement for constituents to be 'purposeful' and set their own goals [45].

Given the heterogeneity of an SoS, there is likely to be considerable variation in the autonomy exhibited by constituents. Modelling and analysis techniques need to permit the expression of a range of actions and behaviours that an autonomous constituent may perform, but which may not be possible or appropriate to share with all constituents at the overall SoS level. For instance for competitive or confidentiality reasons.

Independence

Independence is the capacity of constituent systems to operate when detached from the rest of the SoS. This is Maier's 'operational independence'

characteristic, also identified by Krygiel as the capability for independent action [138] and by Jamshidi as the extent to which systems are 'independently operable' [114].

Independence implies that a given constituent system may offer a range of behaviours of which only some are related to its role in an SoS. This is a challenge for the SoS engineer as the independence makes the constituent system and their stakeholders less committed to participating in the overall SoS and may result in reluctance to sharing information with other stakeholders. This presents challenges when modelling SoS, as the usefulness and quality of models depends on having information and details on the systems being modelled.

Distribution

Distribution refers to the extent to which constituent systems are dispersed so that some form of connectivity enables communication or information sharing. Distribution may denote a geographical distance such as Shenhar's "arrays" of systems dispersed over wide geographical areas [226], and Maier's geographical distribution characteristic [167].

Modelling frameworks that support distribution require the ability to assign constituent system processes to a computational infrastructure, linked by a communication medium. Descriptions of concurrency, communication, and potential failures of communication should therefore be a central element in SoS modelling.

Evolution

Many SoS are long-lasting and subject to change, whether in the functionality delivered, the quality of that functionality, or in the structure and composition of constituent systems. Maier identifies evolutionary development as a key characteristic [167], and Abbott emphasises that an SoS is "continually evolving, but slowly" [1].

Model-Based approaches to SoS engineering require support for gaining assurance of the preservation of specified properties under evolution steps. Evolution may be manifested as updates to constituent systems or system topology, which entails that both modelling notations and development processes need to incorporate progression over time and system flexibility.

Dynamic Behaviour

Dynamic behaviour is the capacity of an SoS to undertake changes to its structure and composition, typically without planned intervention. Several authors identify the ability to undertake this kind of real-time change as an important characteristic, especially in ensuring resilience of an SoS to faults and other threats. [17] identify the need for "dynamic determination of connectivity", requiring the autonomy of the constituent systems to deliver the functions required to disconnect and reconnect constituent systems. [47] regards SoS as dynamic entities, while [222] discuss approaches to the use of runtime safety models to enable dynamic reconfiguration of open SoS.

In contrast with evolution, which refers to the capacity to support planned changes on a slower scale, this dimension refers to the technical abilities an SoS has to change its composition during operation, such as on-the-fly swap-in and a pluggable architecture. To support the dynamic behaviour, SoS models must have abstractions for the dynamic modification of architectures and interfaces, and the capacity to reason about such changing structures.

Emergence of Behaviour

Emergence refers to the behaviours that arise as a result of the synergistic collaboration of constituents. Reliance is typically placed on the delivery of some emergent behaviour in order to deliver a higher functionality than delivered by the constituents separately. Several significant papers, including [167] and [17], refer directly to the need for emergence.

The reliance placed on emergence raises the demands of SoS modelling and analysis methods [257]. Emergence is difficult to predict and capture because emergence cannot directly be designed. Having executable models allow model simulations to reveal the behaviour of the constituent systems and the interactions between them, and as result will show emergent behaviour of the SoS.

Interdependence

Interdependence refers to the mutual dependencies that arise from the constituent systems having to rely on each other in order to fulfil the common goal of the SoS. If the objective of a constituent system depends on the SoS, then the constituent system itself may have to sacrifice some of its individual behaviour in order to meet the requirements of joining SoS.

Including both "Independence" and "Interdependence" may appear contradictory. However, some authors take the view that an SoS requires tradeoffs between the degree of independence in the constituent systems and the interdependence required to reach the common goal [215, 11]. So while the individual constituent systems are independent, the relations and interoperability between requires some degree of interdependence. [47] notes that the US Department of Defense's differentiation between independence and interdependence: A "family" of systems is seen as "a set or arrangement of independent (not interdependent) systems that can be arranged in various ways to provide different capabilities", in contrast to the interdependence that is seen as a characteristic of what is termed an SoS.

Modelling and analysis techniques should allow for the explicit identification of interdependence, the tracing of mutual dependencies, and the ability to use these links to assess the impact of constituent system changes.

Interoperability

Interoperability refers to the ability of the SoS to incorporate and create interactions between a range of heterogeneous constituent systems. This involves the integration and adaptation of interfaces, protocols and standards to enable bridging between legacy and newly designed systems. The interoperability concept appears in the literature as integration of capabilities [172], interoperability and integration [138], heterogeneity [33, 45], "open at the bottom" [1] and diversity [17].

The need for interoperability places requirements analysis methods used in SoS Engineering. Methods are needed that will enable SoS engineering teams to approach system integration in a systematic way that will keep SoS characteristics in focus.

Contribution 1. Converged terms of the literature to identify dimensions that enable SoS to be positioned in the SoS Engineering field

2.2.2 Related Work and State of the Art

Taxonomies are a known way of characterising and differentiating the properties of complex systems in order to identify the system types and the Systems Engineering methodologies needed to develop them [10].

Shenhar *et al.*[227] present a taxonomy for Systems Engineering that can be used to distinguish between different types of systems. The purpose of taxonomy is to make it easier to take decisions on which engineering practices and tools to use for different system types. The taxonomy classifies systems on a four-level technology range from Low-Tech to Super High-Tech, as well as on a scope range from single-purpose to System of Systems. Shenhar *et al.* does not consider a taxonomy directly for Systems of Systems.

D.A. DeLaurentis presents a taxonomy that is aimed at assisting in the design and development of SoS [53]. A lexicon containing structures for categories of systems and levels of organisation is used to establish the taxonomy, through which SoS challenges can be classified. The taxonomy position the SoS challenges on the basis of "the predominant features in their structure and behavior" by characterising them in three categorisations that also are denoted "SoS dimensions". The dimensions are: System Type, Control of Systems and Connectivity of Systems, which relate to: the proportion between machines and human in the SoS, the degree of connectivity, and whether the control is centralised or distributed, respectively.

The SoS dimensions mentioned here can be considered as axes along which systems can be slid and positioned in a three dimensional space, while the dimensions presented in Section 2.2, more can be considered as groupings that each represent a certain characteristic of SoS.

2.3 Modelling and Simulation in SoS Engineering

Having a specific focus on Model-Based SoS Engineering the survey also included a study of the current state of the practice in Model-Based techniques [P189] within the SoS field. These were related to the eight dimensions in order to identify the current challenges in performing Model-Based SoS Engineering and to uncover future research goals for strengthening the use of modelling in SoS Engineering.

2.3.1 Modelling

The survey showed that the use of modelling as an engineering tool has naturally spread to the SoS Engineering field as a result of modelling being applied for many years in other engineering disciplines [32]. In most circumstances it is the case that existing modelling notations are being transferred directly from other fields and attempted applied to SoS engineering [72]. The concern with using a modelling notation intended for another engineering do-

mains or task is that these notations often are domain-specific languages that are aimed a solving concrete problems [238]. This means that the notation often will embody domain knowledge and be tuned to work at the right level of abstraction for the problem domain [137, 136]. When starting to apply domain-specific modelling notations to tasks outside the intended domain, difficulties occur as a result of the engineering task, to which the notation is applied, not mapping to the notations targeted domain. This entails challenges in: maintaining models, encounters with constraints of the notation, and difficulties in reasoning about the model as the system size increases [89].

One of the main challenges for modelling languages aimed at SoSs is to ensure that they have well-founded semantics; as many of the modelling languages in the current practice do not. As SoSs have a high degree of *Distribution*, the modelling languages must be able to express the notion of multiple independent execution platforms that can be interconnected. A particular focus should be the languages' ability to include *Interoperability* as part of modelling the individual constituent systems and their relationships.

In the current modelling efforts agent-based approaches are used to model the *Autonomy* of SoS. A main challenge of doing SoS modelling is to embed this aspect into more modelling languages by including constructs that enable users to describe autonomous behaviour; in particular of how humans act in relation to the SoS. Likewise, modelling languages need constructs that can describe the *Dynamicity of Behaviour* in order for the models to express the dynamically changing infrastructures which enables the constituent systems to initiate and break their interrelationships.

As such it makes sense to define modelling languages dedicated to the modelling of SoS, but there has been few attempts in the literature [71, 86, 163]. Besides the CML notation, described in Section 1.3, another key contributions is a modelling formalism in which components form the basic structural elements of the SoS and the behaviour of the SoS is described by parameterised state charts [86]. This formalism is concentrated on statically as well as dynamical characteristics and has a strong focus on adaption and evolution of SoS. Graph grammar and transformations rules are used to describe reconfigurations and changes in the overall system structure and as such addresses the adaptivity of SoS [102].

As the integration and relationships between the constituent systems is a fundamental part of SoS, an important part of modelling in SoS Engineering is systems architecture and composition. However, in the existing literature the direct usage of the terms 'architectural model" and "architectural modelling" in relation to SoS is rather sparse, and only few authors

have a direct focus on SoS architecture, and mostly in relation to Enterprise Architectures [161].

Selberg and Austin demonstrates how using more standardised interfaces enable an easier integration of new constituent systems into the SoS architecture as it evolve [224]. Given a lack of formal methods supporting proper engineering of SoS, there is a recommendation for using formal models to provide analysis capabilities as a way of dealing with the otherwise unmanageable increase of complexity.

Kilicay-Ergin *et al.* has a focus on SoS architectural modelling, but see a limitation in using static frameworks and methodologies for architecture development as they do not provide a way to analyse dynamic evolution of system state or behaviour [129]. In order to reveal and analyse the behaviour of the overall SoS architecture they advocate the use of executable models.

2.3.2 Simulation

Executable models or model simulation is one of the most frequently used forms of model analysis. In order for the simulation of an SoS model to be efficient it must be able to show the characteristics of an SoS. If the main characteristics can be incorporated into the model, the volatile characteristic emergent behaviour has a much better chance of being detected through simulation of the model.

One SoS characteristic that is particularly well suited for simulation is the aspect of *Autonomy*. Simulating an SoS will show system behaviours and the constituents' interactions, which would be very difficult to predict by merely analysing the models or systems statically.

An SoS simulation environment should also support the *Independence* of the constituent systems, by allowing both stand–alone and combined simulation of models. The constituent systems may be described in separate models that need to be simulated both individually and in combination with models of other parts of the SoS.

An SoS simulator should be capable of capturing the *Dynamicity of Behaviour* that occurs as a result of the dynamically changing system topology, such that the dynamic changes can be communicated to stakeholders.

In the existing literature, the focus of model simulations of SoSs is primarily on using the models for training purposes for different kinds of personnel or for modelling human behaviour in the SoS. Agent-based technology is typically included when it is desired to include the human in the simulation

loop [8]. Thus, the use of agents are often used to explore human behaviour and socio-technical aspects in an SoS scenarios [92, 166].

Examples can be found that attempt to create a general modelling and simulation approach aimed directly for SoS. One such example is presented by Kotov where a C++-based library is presented for modelling and simulation in an SoS setting [133]. A similar generic approach extending the Unity language to an SoS setting can be found in Gamble and Gamble [82]. Sahin *et al.* presents a framework for the architectural representation and simulation of an SoS based on Discrete Event System Specification (DEVS) and the exchange between constituent systems defined in XML [217].

Part of the literature on SoS simulation has a focus on interoperability between simulations and models running in distributed simulators. The majority of the literature is focused on High-Level Architecture (HLA) simulation interoperability [156]. HLA has primarily been used for coupling different existing high-fidelity, defence-related simulators together. Distributed Simulation is described in more details in Section 3.4.

2.4 Integration

When considering the basis of SoS as being a system that composes other systems which interact and exchange data between each other, it is evident that system integration is a central challenge in SoS engineering. The SoS dimensions: *Interoperability*, *Interdependence* and *Independence* all have a relation to integration. They have implications and dependencies on the engineering practices being applied in joining and assembling complicated systems. The *Interoperability* involves composing and enabling interactions between heterogeneous systems through the adaptation of interfaces, protocols and standards. The *Interdependence* dimension sets forth demands to the integration methods being applied as it has to deal with mutual dependencies between the constituent systems being integrated. The complexity in the interdependence comes from the constituent system themselves potentially having to sacrifice some of their individual behaviour in order to enable the integration that will fulfil the common goal of the SoS. The dimension of *Independence* entails that the constituent systems have the capability to operate separately from the SoS and such may not be fully committed in their participation in the SoS. Therefore, they may difficult to integrate as it may not be feasible to adjust their data types or interfaces.

The importance of integration and well-functioning interactions between the constituent systems is well known in the SoS Engineering field, as such

it is a topic for which research and development is considered highly important [221, 158]. Mane *et al.* focuses on the role system interdependencies structures in achieving successful SoS integration [171]. M.J. DiMario address the SoS integration challenges by looking at three different types of interoperability: at the programmatic, constructive, and operational levels [56]. These relate to: system acquisition, linkages between engineering teams and organisations, and linkages between systems seen from a operation or technical point of view, respectively. M.J. DiMario sees a primary interoperability challenge between legacy and new systems, and calls for an increasing understanding of the interoperability and improved engineering methods to establish SoS centric systems.

Doing systems integration successfully and creating interactions between heterogeneous systems has been a major challenge for several decades and is known as being difficult and complicated [138]. Integration depends on: the available technologies ability to allow for connectivity between systems; how well-defined the semantics of the transferred data are; how easy the systems architecture can provide sharing of data and the functionalities between systems; and finally it depends on how transparent the integration is to the end user [195].

The challenges of systems integration is well-known in software engineering projects and this has led to various methods and approaches targeted at strengthening integration [209, 12]

2.4.1 Research Conducted

In [P125] we took a common development technique from the Software Engineering field and adjusted it such that it could be applied in an SoS context. We looked at the use of software design patterns and developed a way of systematically identifying and applying them in order to support SoS engineering in addressing integration challenges. The use of design patterns is a common practice in software engineering, where a generic solution that is continuously applicable to a specific common problem is detailed in a type of solution template [83, 60]. The value of design patterns is that they represent the distilled wisdom of thousands of projects that have gone before and as such offer a solid foundation to solve the problem at hand.

In the paper we presented an approach that can characterise design patterns for SoS integration. Taking an architectural perspective on SoS design, the paper provides: 1) a classification for the design context the SoS is developed in and 2) an identification of which properties that characterise the SoS.

The purpose of the classification is identifying which patterns are relevant for solving the integration problems for a particular SoS.

The classification is built around a set of categorisations that enable the scope and context of the current SoS to be classified, such that the relevant integration patterns can be identified. A template has been created that defines the classification and is used to characterise integration patterns. The classification makes use of four categorisations for determining the SoS development context and five categorisations for determining the technical context. An overview of the classification template is shown in Table 2.3

Attribute	Value Range
Name	⟨ Short descriptive text ⟩
SoS Scope	System of Systems, Systems
Development Context	Greenfield, Brownfield, Closed Source
Integration Purpose	One-Directional, Information Exchange, Bi-Directional, Information Exchange, Control, Negotiation
LISI	Isolated, Connected, Functional, Domain, Enterpise
PAID Attributes	Procedures, Applications, Infrastructure, Data
Integration Level	Information Exchange, Basic Behaviour Interaction, Complex Behaviour Interaction, User Interface Sharing
Data Abstraction Level	Structural, Syntactic, Semantic
Data Level Integration	File-Transfer, Message-exchange, Streams, Common Data
Interaction Style	Send, Call, Call-Return, Call/Call-Back, Time-Based, Multi-Call Protocols
Quality Attributes of Integration	Reliability, Performance, Security, Availability, Interoperability, Scalability, Manageability, Consistency
Pattern defined by	⟨ Reference to pattern literature ⟩

Table 2.3: Template for SoS integration pattern classification

The SoS development Context categorisations are concerned with the setting in which the SoS is developed and is focused on the scope and context of the system. The categorisations are:

System of Systems Scope determine if the scope of the SoS has been firmly defined.

Development Context specifies the development stage of the SoS, e.g. an entirely new system, modification of existing systems, integration of legacy systems.

Integration Purpose details the flow of data and how decisions are taking in the SoS between the constituent systems being integrated.

LISI and PAID allows for a relation to the Levels of Information System Interoperability (LISI) and Procedures, Applications, Infrastructure, and Data (PAID) interoperability models [90].

The technical context is concerned with the specific technical aspects of the integration and they allow for more fine-grained details of the integration to be applied.

Integration Level determines how tightly coupled the constituent systems need to be.

Data Abstraction Level details how and how well shared data is understood between constituent systems.

Data Level Integration specifies how data technically is being shared.

Interaction Style describes how data flows and interactions between constituent systems occur.

Quality of Integration contains quality attributes such as performance, integrity and scalability.

This template was applied to a range of patterns found in a survey on software literature, and a collection was gathered [P124]. The purpose of this was to create a repository of software integration patterns and attach them to a classification aimed at determining their applicability to a specific SoS integration challenges. Having an understanding of the characteristics of the constituent systems, the infrastructure, and the data in the system is critical in dealing with SoS integration and as such the purpose of the approach is to present a way of considering and evaluating patterns for SoS integration.

Contribution 2. Classification for identifying design patterns that aid the integration challenges in SoS Engineering

2.4.2 Related Work and State of the Art

There is a vast amount of material in the literature that focuses on integration challenges in software and on using patterns to aid the field of software engineering. While the existing approaches and software pattern collections

for addressing integration problems are useful, they focus on the integration problem in a generic way that is not specific to an SoS-context, or make implicit assumptions that are not directly applicable in SoS Engineering. On the other hand, the literature focused on using software patterns for dealing with SoS integration challenges is rather sparse.

Morris *et al.* have a specific focus on the challenges in making both systems and organisations work together [181]. They present the System of Systems Interoperability (SOSI) model which aims at identifying interoperability problems for which solutions or partial solutions are possible. In the same way as presented above, the SOSI model also make use of the LISI and PAID models as part of its system characteristic. The SOSI model differs in taking a broader view on SoS integration, as while it has fewer characteristics in the technical integration it also includes characteristics for program management and system operation. The SOSI model is not focused on identifying concrete design patterns, but attempts to identify engineering methods at a higher level.

Ingram *et al.* [109] presents a study on modelling patterns and architectures for SoS, with the purpose of establishing a better understanding on how patterns can support architectural reuse and increase system comprehension. The study identifies the requirements for an SoS architectural design pattern and surveys the literature in order to identify existing patterns. For each identified pattern there is a focus on the background, aims and structure of the pattern from which a justification for the patterns use in an SoS context is derived. The approach differs from the one presented in [P125] in the way the patterns are selected. The approach in [P125] uses four categorisations for determining the SoS development context and five categorisations for determining the technical context, while the approach by Ingram *et al.* selects the patterns on the basis of architectural principles with a focus on Maier's five SoS characteristics [167] (presented in Section 2.1).

2.5 Collaboration Study

A key factor in advancing the SoS Engineering field is the existing development practices and methods being driven in the direction the challenges of SoS, especially in relation to its technical aspects. Formal methods are, for instance, well-suited for addressing technical challenges related to specifications and analysis of structures and behaviours. The SoS Engineering field does, however, face socio-technical challenges as well, as a result of the numerous stakeholders that are involved in SoS development [160]. It is

well-known that humans play a role in the challenges and solutions of the SoS Engineering field, as the establishment and development of interactions between the constituent systems rely on human to human interaction [221].

The challenges arise partly because of 1) the complexity of SoS and partly because of 2) human-behaviour [52]. The complexity of SoS can make the systems difficult to grasp, and the many facets and details of the system leaves a large space for interpretation in which each person can take their own unique angle. When constituent systems are being connected during the creation of an SoS it requires some type of convergence in order to establish interaction and interoperability. Human-behaviour can however make this a complicated task. In such a process, humans can express a large degree of self-interests and conflicting perspectives, which can cause uncertainty and ambiguity in the system design. Out of a need for handling incompatibility, design conflicts and stakeholder interests; human decision–making is an important part of SoS Engineering [177], although it comes with some challenges.

Parts of these challenges are expressed in the *Autonomy* and *Independence* dimensions, as they make it challenging to anticipate the system behaviour. They entail that the constituent systems have different stakeholders, have been developed individually and are self-contained. This makes it more difficult to reach agreements on design, anticipate the behaviour of system and handle system changes and evolution. The heterogeneity of the constituent systems means that diverse stakeholders need to exchange information and negotiate agreements on how the constituent systems are going to interact. This is challenged not only by the self-interests of the stakeholders, but also by the *Interoperability* dimension which entails the adaptation of interfaces, protocols and standards. The interaction between the human stakeholders is also hindered by the *Distribution* dimension that entails that both systems and humans involved in the development may be globally dispersed, which places a necessity for precise and efficient information exchange to support the engineering of the SoS.

Having geographically distributed stakeholders that have to work together on a common project is not a new challenge in the field of engineering. Since the mid-1980s research and development has been carried out in order to make computer applications support groups and organisations [91]. Grouped under the term Computer-supported Cooperative Work (CSCW) it involves a range of different methods for collaboration, such as electronic meeting software, video-conferencing systems and collaborative authorship applications. While there is a general focus on supporting and coordinating

collaborative activities through the use of computers, there is one part of CSCW that mainly focuses on tools and techniques for collaboration, and another part that has to do with the social and psychological aspects of collaborative work [34, 40]. The work presented here does however focus on the tool aspects of collaboration.

The Software Engineering field has seen an increase in the use of globally dispersed development teams, which has led to an increased focus on the challenges of doing distributed development [101]. As development activities are distributed across different geographical locations and organisations, issues begin to appear with respect to precise communication and effective coordination between the distributed collaborators. These issues are connected with the way engineering teams work, seen from a technical and social point of view, as well as on the availability of tool-support that can support collaboration [6]. The socio-technical challenges arising from software development being performed in a distributed manner has led to the development of the engineering paradigm Global Software Engineering (GSE) [100]. GSE is focused on the coordination and collaboration challenges of distributed software engineering and researches how distributed software engineering can be improved [104]. GSE has a close relationship with CSCW and has many of the same focuses, just with an emphasis on Software Engineering.

The challenges in collaboration are not unknown to the Systems Engineering field either, and the development of distributed collaborative environments for Systems Engineering has previously been proposed in the literature [175]. The research on collaborative development is sparser within both the SoS Engineering and the formal modelling field, meaning that the literature contains a gap in the area of collaborative development of formal models for SoS.

2.5.1 Research Conducted

In [P186] we presented the results of a case study that was focused on examining the challenges related to doing collaborative development of formal models of an SoS. The purpose of the case study was to obtain more information on the challenges, processes, advantages and disadvantages of collaborative SoS modelling. The goal was to gain knowledge on how the modelling of SoS can be approached and what the challenges are of developing them collaboratively.

The case study involved an emergency management SoS in which different emergency services were involved. The emergency management SoS was

responsible for providing the needed information exchange between various emergency services in the case of an emergency. The study was of an academic nature and as such the role of each emergency service was played by the university partners in the COMPASS project. The case study was however inspired by an emergency management system developed by the COMPASS project industrial partner: Insiel. The study involved a total of five partners from four countries stretching across two continents.

In the case study the different emergency services needed to collaborate in order to solve a given emergency situation. The services had different roles (e.g. Police, Fire and Rescue, Ambulance services, etc.) that each provide and require different capabilities and types of information. Each emergency service represented a constituent system and the combination of these formed the overall emergency management SoS. Consequently, each involved partner represented a collaborator which was the stakeholder for one emergency service constituent system. The emergency situation gave some information about the emergency that was globally known and some "private information" that was only known by the individual emergency services. This information may be irrelevant to others, it may be confidential, or it may be information that is vital for some of the other services.

Each emergency service was asked to determine certain information that only could be obtained via some of the other collaborators in the emergency management SoS. This created a scenario, where each collaborator had to consider the implications and side effects of participating: concerns for interoperability, data sharing across organisational boundaries of the various services, and in determining the necessary level of information sharing and collaboration.

In the study the collaborators were asked to create a CML model of their respective constituent system, and to create a joint CML model for the overall emergency management SoS. The collaborators were asked to create the: channels, interfaces, data types and interactions needed in order to solve a given emergency.

As part of the rules for the study some limitations were introduced as to how the collaborators were able to communicate. The main limitation were that the "private information" only known by the individual collaborators should not be shared directly through e-mail or similar means, but only be expressed and exchanged through channels and data types defined in the CML models.

The study ran over the course of three months in which the collaborators discussed system functionality, interfaces, data types, etc. in order to create

and evolve their models. There was no group attrition during the study. In addition to the collaborators the study had an observer who did not take part in the modelling effort. During the study the considerations and negotiations taking place were shared with the observer as a form of passive participant observation and as such the collaborators were aware of the data collection [223]. The observer was never present during the modelling effort itself, but only notified of the electronic or verbal exchange between collaborators, in order to be as unobtrusive as possible.

During the study the models developed by each of the collaborators were handed-in at certain deadlines, and as such the models functioned as a type of trace data from which the progress and approaches could be seen. Finally, self-reported data collection was performed during two joint physical meetings where the study was discussed between all collaborators and the observer.

Consequently, the study is based on observation data of a small project study. While a degree of participant reactivity is inevitable, it is considered to be of a minor degree; partly because the focus was on practical engineering challenges and not on the performance or behavioural facets of the collaborators, and partly because the study was of an academic nature. This meant that the study was not directly related to the professionalism or personalities of the involved collaborators, as it may be in more sociological studies with a stronger focus on the social processes.

The study showed that the collaborators experienced the most problems when it came to: 1) exchange of model information, 2) agreeing on design decisions. In connection with the first problem the collaborators found that it was difficult to exchange the CML models as there was no process or tool-support for handling this. This meant that the models mainly was transferred via e-mail, which caused further challenges with versioning and traceability as well as the hassle of the models being manually moved in and out of the modelling tool. In sharing the models they also encountered the problem of inadvertently sharing information that was not meant to be shared, as some of their private information was encapsulated in the model. With the second problem a large part of the effort spent by the collaborators was on ensuring the interoperability between the constituent systems, as a joint model for the overall SoS needed to be established on the basis of the CML models of the constituent systems. As it was expected, with the knowledge from CSCW and GSE, collaborating and agreeing on design decisions was a difficult, time-consuming and error-prone process. While it was possible to converge and reach a level of agreement through e-mail discussions, it became troublesome

at the modelling level as discrepancies and inconsistencies would appear between the individual models, especially on specifics such as shared data types and naming of communication channels and type names.

The experiences and discoveries of the study was used as the base for further research, which is presented in Section 3.3.

Contribution 3. Identified challenges with collaboration in relation to using formal modelling for SoS Engineering

2.5.2 Related Work and State of the Art

Collaboration and working in groups have been intensively studied within many professional fields, and with the investigation being carried out by many different research fields, among which psychology and sociology are the most dominant. Giving details on these general studies are outside the scope of this dissertation, but a review of the research on team formation, socialisation, development and processes can be found in Kozlowski *et al.* [134] that also provides survey of the psychological research on team effectiveness in [135].

Instead this section will focus on the studies that have been performed on collaboration within the fields of SoS Engineering and formal modelling. Working collaboratively is not new in the field of engineering [50], but it has not had a strong focus within SoS Engineering nor in the formal modelling field, where the research on distributed development teams working collaboratively is limited.

A design framework has been presented that can define the role and process characteristics of collaboration in SoS [98]. The framework focuses on the differences in opinions of stakeholders and on the policies define between them to achieve a satisfactory collaboration. The policies and processes of the collaboration can be described in a collaborative meta-model and be analysed using software agents [97]. The framework is focused on the business aspects of SoS development and the meta-model is more to be considered as type of a taxonomy than a formal model.

Bryans *et al.* use the formal modelling technique the Vienna Development Method (VDM) (presented in Subsection 1.3.3) to map and arrange a type of collaborative alliances known as “dynamic coalitions” [24]. Dynamic coalitions are similar in concept to “virtual organisations” and both terms describe a type of collaboration between individual systems or stakeholders that form temporary partnerships in order to achieve specific goals. Their

associations with each other are unstable and they can vary in architecture, scale and complexity. As such they express many of the same characteristics as SoS, although the they do not express the same degree of joint evolution as SoS. Bryans *et al.* use a case study on collaboration between chemical firms to investigate aspects such as membership, ownership and trust [25].

3

Tool-support for System of Systems Engineering

The field of engineering has always been aided by tools to help in the analysis and construction of systems [84]. In the field of software engineering Computer-Aided Software Engineering (CASE) tools are used by software engineers to organise, model and study the systems and software being developed [27]. The use of software tools for aiding the engineering process has been adopted from the hardware manufacturing process as a way of adding a more disciplined and structured development process [201]. This chapter focuses on research performed in this PhD project on providing software tools that can strengthen the field of System of Systems (SoS) Engineering. The research in tool-support is focused on addressing the challenges identified in Chapter 2 and the developed methods and tool-support is aimed at covering the dimensions presented in Section 2.2. The tool-support is provided through modifications and extensions of existing software tools and modelling language notations.

The tool-support has partly been added to the Overture Tool that is aimed at supporting the Vienna Development Method (VDM) modelling language and the partly to the Symphony tool suite that is aimed at the modelling and analysis of SoS in the COMPASS Modelling Language (CML) notation.

Section 3.1 presented the tools in further details. Section 3.2 presents an approach for adding dynamic reconfiguration capabilities to the VDM-RT notation and the Overture tool in order to enable the description and simulation of the *Evolution* and *Dynamic Behaviour* dimensions. A tool-extension for the Symphony tool which enables it to become a Collaborative Development Environment (CDE) with the purpose of aiding the challenges of collaboration between SoS engineering teams (identified in Section 2.5) is presented in Section 3.3. An additional tool-extension of the Symphony tool that allows collaborating engineering teams to perform distributed simulations is presented in Section 3.4. Finally, an extension of the Overture tool that allows model to be integrated with externally defined code libraries, and

an extension of the Symphony tool that enables SoS models to be connected and simulated with externally running systems, is described in Section 3.5.

3.1 The Overture and Symphony Tool Suites

The developed tool extensions have partly been implemented in the Overture tool suite which supports VDM models, and partly in the Symphony tool suite which supports CML models. At the beginning of the PhD project the Symphony tool did not yet exist. Consequently, the first tool-extensions were developed for VDM models and the initial research was studied with the use of tool-extensions for Overture tool. During the course of the PhD project the Symphony tool came to life and was continuously developed to an increasingly mature tool. When the Symphony tool had reached a level of maturity where SoS models could be analysed and simulated, the research moved to the new platform. As such the later research was performed and studied in tool-extensions that were developed for the Symphony tool. This section provides a brief introduction to both tool suites.

3.1.1 The Overture Tool

The Overture Tool is a tool suite for the modelling technique VDM, aimed at supporting the modelling and analysis of computer-based systems[1] [144]. The Overture Tool consists of an Integrated Development Environment (IDE) build on top of the Eclipse platform [59, 180] and a core part that has its origin in VDMJ [13], all written entirely in Java. The IDE provides the graphical user interface with file manager, editors with syntax highlight, debug functionality and all the plugins that gives access to all the different types of validation and verification functionality delivered by the tool. The core of Overture provides the main part of the model validation and verification as it contains functionality such as the type-checker, the interpreter and the proof obligation generator, etc. An overview of the Overture Tool is provided in Figure 3.1.

The Overture tool is an open source initiative initiated in 2003 [69] with the purpose of creating an industrial-strength tool that could support the development and analysis of all of the VDM dialects. The purpose of the tool being open source is to allow researchers and other interested forces to make the modifications and tool extensions needed to perform experiments.

[1] http://www.overturetool.org/

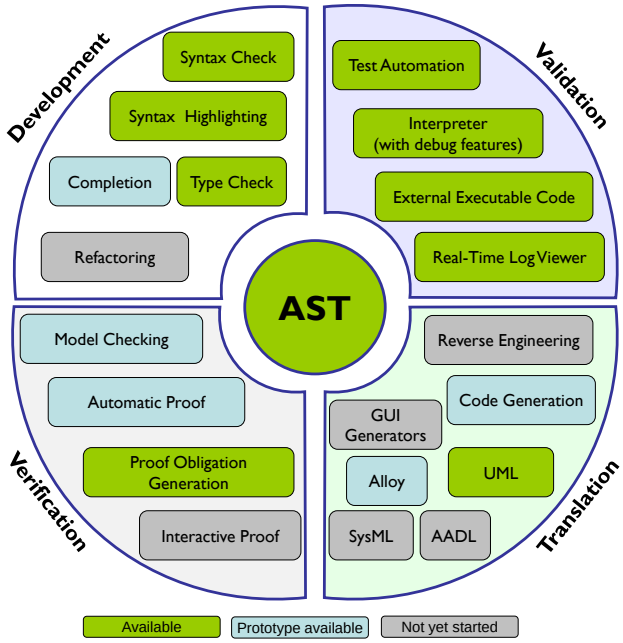


Figure 3.1: Overture Overview - adapted from [151]

The Overture tool makes use of an Abstract Syntax Tree (AST) that is central to the core components of the tool [193]. As part of the Overture project, the AstCreator tool exists that can perform automatic code generation of the Java classes that make up the Overture AST on the basis of an AST script file. The AstCreator tool includes a functionality that allows the Overture AST to be extended to allow reuse of existing features. Having the central AST enables tool developers to access the VDM models contained in the tool, in a uniform way. The open source aspect of the tool allowed for active research in the tools for VDM and a range of advances have been made for the tool [147, 145, P191, 42] including a translation between VDM models and UML [153] and a translator for VDM to Alloy [152].

The source of Overture tool being open was also utilised in the research performed in this PhD project. Having the ability to modify the source code made it possible to make adjustments to the interpreter which is used to perform the simulations of the executable VDM models. The changes made to the interpreter was made to enable the dynamic reconfiguration capabilities

described in Section 3.2 and to allow VDM models to be integrated with externally defined code libraries, as described in Section 3.5.

3.1.2 The Symphony Tool

The Symphony tool suite is developed as part of the COMPASS project to provide a platform that can support the systematic engineering of Systems of Systems[2] [72]. The Symphony tool provides an extendible platform for tools and plugins aimed at model construction, simulation, proof obligation generation, static analysis by model-checking, and test automation[3] [41]. In addition to this the tool integrates with the RT-Tester[4] tool to provide test automation and with Artego Artisan Studio[5] such that the abstract structures of the overall SoS can be described at an architectural level using SysML. A Transformation Development Kit in Artego Artisan Studio has been used to develop a transformtion from a SysML model into a CML model. An overview of the Symphony Tool is provided in Figure 3.2.

The Symphony tool is a close cousin of the Overture tool and parts of the functionality of the Symphony tool builds directly on parts of the Overture implementation. In fact the AstCreator for the Overture is also used to generate the AST for the CML notation. The extension feature of the AstCreator, as mentioned in Subsection 3.1.1, is used and the VDM AST lays the basis for the CML AST. Specifically, the VDM-derived elements that exist in the CML notation are type checked using the Overture type checker. Having a close relationship between the Overture and Symphony tools allowed for the existing competencies gained from the development of the Overture tool, to be utilised in the development of the Symphony tool.

As with its close relative the Symphony tool is also an open source project based on the Eclipse platform, and several of the basic components of the Symphony IDE have their origin from the Overture tool. As such the tool has a strong focus on extensibility and within the COMPASS project the rich plug-in functionality of the Eclipse platform is used to add the tool support in modular manner. For instance all of the analysis modules, such as the Proof Obligation Generator, the theorem prover and the plug-in for test automation, are added as plugins.

---

[2] http://www.compass-research.eu/

[3] https://github.com/symphonytool/symphony.org/

[4] https://www.verified.de/products/rt-tester/

[5] http://www.atego.com/products/artisan-studio/

The dynamic plug-in structure of the Eclipse platform was used to add the CDE to the Symphony tool, as described in further details in Section 3.3. In the same way as with the Overture tool, having the ability to change the source code, on a separate branch, made it possible to modify the simulator in order to enable distributed simulation, as described in Section 3.4.

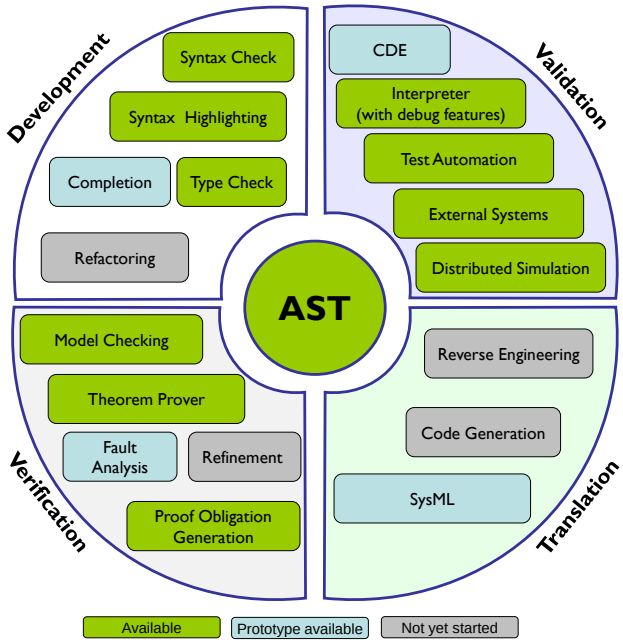


Figure 3.2: Symphony Overview

3.1.3 COMPASS Modelling Language - Symphony Internal Simulation Representation

When the Symphony tool is used to analyse and simulate CML models it has an representation of the model it uses internally. This subsection supplies the fundamentals of the representation needed for understanding the Symphony tool changes that are presented in the following sections.

During simulation of the CML model the progress occurs by the CML processes being triggered by events occurring on channels. Inside the simulator the term *behaviour* is used to represent processes and actions during simulation. It is the actions in processes and the current events on channels

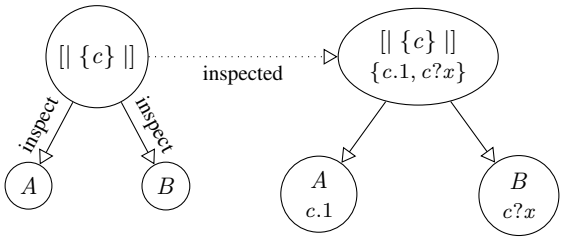


Figure 3.3: The system before and after the children of top process has been inspected

that determine which events will be available next. The simulator inspects the behaviours in order to determine the next steps that can be taken in the simulation. Consequently, an inspection produces a collection of next possible events that can be executed. Once a step is taken the simulator will execute the related behaviour, potentially creating events that trigger the synchronisation between processes. This leads to new model state and an inspection is performed again to find the next possible events. Internally, the simulator continuously call inspect and execute commands in order to perform the simulation.

Within the simulator, a CML model is represented as a tree structure of behaviours. The top process is used to define the entry point for the simulation and the simulation is performed by traversing the tree top-down to inspect and execute underlying behaviours. A composed process could for instance have two children representing behaviour A and B. A parent in the tree structure functions as a coordinator for its children, meaning that it will filter the collection of events available to the children and thereby control the possible execution. The inspection of the composition defined for process P will result in the collection of events to be gathered, as shown in Figure 3.3.

3.2 Dynamic Reconfiguration

The Evolution and Dynamic Behaviour dimensions present challenges for SoS engineering as they set forth demands for handling and analysing the changes that will occur in the SoS. Not only must the SoS architecture itself have the ability to dynamically reconfigure the system topology in response to evolving needs [221]. It also raise demands for the capabilities of the development tools being used. Having a dynamic and adaptive system requires some additional aspects of the development methods, than the engineering of

a static system architecture where the structural elements remain the same. Having a dynamic system means that the system topology may only be stable for a short period and that the behaviour will change over time [176]. When looking at complex dynamic systems it is known that the overall behaviour of the system is dependent on its structure and internal relationships, not just on the behaviour given by the constituent systems [75]. As such the dynamic behaviour and evolution of an SoS not only cause changes in the system composition it also changes the behaviour of the SoS. This may entail loss of performance, changes in the emergent behaviour or even system failure,

3.2.1 Research Conducted

In response to a lack of proper tool-support for describing and analysing the evolution and dynamic behaviour of SoS we presented an extension of the Overture tool that enables dynamic reconfiguration to be expressed and simulated in VDM-RT [P187]. The purpose of the extension was to provide a tool that could increase the certainty and confidence in the design and the development process of an SoS. Being able to model and analyse the behaviour occurring in a dynamically evolving SoS would make it easier to demonstrate the consequences of design decisions to different stakeholders

VDM-RT is aimed at modelling and simulating system architecture and behaviour of a distributed system by defining a system consisting of CPUs and specifying the network topology connecting between them [243]. VDM-RT was chosen because the modelling of SoS architectures require a modelling notation that is aimed at describing distributed systems with the interconnection and communication between the constituent systems. However, the original VDM-RT assumes a static set of hardware and software configurations, which is a troublesome limitation when modelling systems with highly dynamic behaviours.

The extension was two-fold: 1) minor changes were made to the notations of VDM-RT in order to better express the terminology of SoS field and dynamic architecture changes, 2) a change to the simulator to allow for dynamically changes of the network topology, the number of channels and the number of constituent systems during the run-time execution of a model. The intention is to enable a VDM-RT model to capture the fundamentals of dynamic reconfiguration in an SoS and not to incorporated specific reconfiguration techniques.

With the changes to the modelling language it became possible to describe an overall system architecture of the SoS in a **System** class that contains

a number of **Constituent** systems and a number of **Channel** types that establish communication links between the constituent systems. A constituent system is a processing entity to which modelled applications can be deployed. The dynamic reconfiguration of the system architecture is initiated through a number of static reconfiguration operations that can be called on the system class. When simulating the model the reconfiguration operations can add and remove constituent and channels dynamically, as well as change the network topology dynamically by connecting and disconnecting constituents to and from communication channels.

The effect of the extension was examined through a road safety case study. The studied SoS uses an intelligent traffic infrastructure that enables communication and data exchange between vehicles in order to increase the traffic information available. As vehicles pass each other on the road they will create wireless networks through which they can co-operate. The system is highly dynamic as the constant movement of the vehicles result in a constant change in connections and relations. As there is no central entity and the behaviour of the overall system is based on an unrestrained web of relations, the study was used to examine how well the tool extension enables the modelling of such an SoS. The modelled system with the connections and exchange of traffic information is visualised in Figure 3.4. The visualisation is made possible by the research presented in Section 3.5.

Having the ability to express the dynamically changing and evolving nature of the SoS directly in the model made it possible to better reflect the systems altering structure and reveal the behaviour of the SoS as the vehicles moved around.

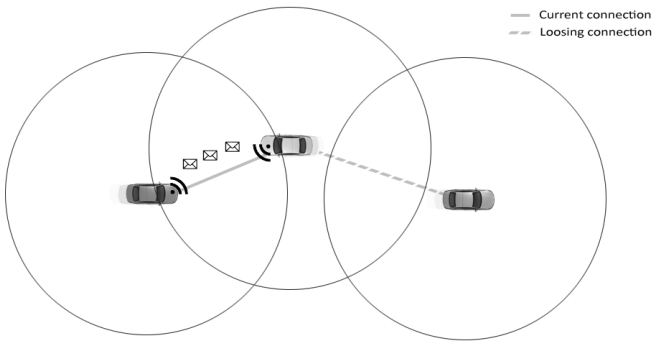


Figure 3.4: Communication and network between vehicles

Contribution 4. Approach for expressing and simulating the SoS dimensions of Evolution and Dynamic behaviour in formal SoS models

3.2.2 Related Work and State of the Art

The dimensions of *Evolution* and *Dynamic Reconfiguration* has been researched before, both in connection with SoS and formal modelling.

A. Meilich focuses on net-centric environments [194] as means of introducing adaptive architectures that enable systems to have run-time flexibility and be composed over time [178]. This flexibility comes at the price of predictability, but model simulation presented as a way of addressing this challenge as it enable experiments to be performed as the system evolves.

Sahin *et al.* presents a framework that uses a combination of Discrete Event System Specification (DEVS) and XML for making architectural descriptions and simulations of SoS [217]. DEVS is a discrete-event formalism for building hierarchical systems by defining components and their interconnection, and simulating the interactions between them, including the run-time addition/removal of couplings and components. DEVS is used to represent the structure and behaviour of the SoS, while XML is used for the data communication between the modelled constituent systems. DEVS and VDM are both capable of modelling discrete event systems but they use different approaches. DEVS has an origin in systems theory and has a strong focus on state transitions, while VDM has a formal methods origin with a set-theoretic foundation.

Kotov presents an object-oriented framework aimed at the modelling and analysis of SoS [133]. The Communicating Structures Library (CSL) framework describes an SoS as a range of system components represented by nodes that are connected through nets. Nodes can represent a range from microprocessors to computer clusters, and nets can represent anything from peer-to-peer network to busses. Procedures known as Processes are used to send data between nodes and as such function as the driving force of the model simulation. The framework is aimed at modelling and simulating system behaviour, interactions, message flow and system scalability. As such the CSL framework is comparable with the VDM-RT and Overture extension, but differs in the way that VDM-RT has the main processing and behaviour enclosed in the systems themselves and there is a stronger focus on changing the topology dynamically, than found in CSL. CSL on the other hand provides

capabilities for compositional system structures through the use of sub-nodes, which cannot currently be performed in VDM-RT.

From a formal modelling perspective Johnsen *et al.* presents an approach for the high-level modelling of distributed systems with diverse quality networks with the purpose of analysing properties of heterogeneous nets [117]. The work is an extension of the executable object-oriented modelling language Creol [118] to enable the modelling of dynamic reconfiguration of networks between components. A system class defines the network topology and the class can be used to reconfigure the topology, an approach that is very similar to the approach presented here. In contrast to adding new operations to perform the reconfiguration, Creol introduces new language keywords and guard predicates are used to determine the configuration changes, additionally Creol uses a notation that designates the entire system as the system, while the constituents systems are placed in components. Creol is however not focused directly on SoS Engineering.

3.3 Collaboration

Through the results and experiences gained from the study presented in Section 2.5, it was made clear that the SoS Engineering field is challenged by collaboration required to establish agreements and interactions between systems and stakeholders. The collaboration required between both the independent systems as well as the stakeholders of the systems raise challenges with both technical and human factors. SoS engineering is dependent on the quality of the collaboration between individual developers and owners of the constituent systems in the SoS.

The *Autonomy* dimension of the constituent systems makes it difficult to anticipate their behaviour, as they only are governed by their own rules and details which they may not be willing to share. The dimension of *Independence* means that the constituent systems are capable of operating independently as they often have different owners and are developed individually. Consequently, including these dimensions in SoS engineering entails both a socio-technical element and a technical interoperability element. The varied ownership means that humans need to negotiate an agreement on how the constituent systems are going to collaborate. Properties such as interfaces, data types, resource consumption and the degree of confidentiality need to be negotiated between geographically distributed development teams. Additionally, the *evolution* of SoS entails that the individually owned constituent systems will encounter changes or change requests from other con-

stituent systems. These are decisions that require human decision–making and negotiation between the owners of the constituent systems.

3.3.1 Research Conducted

The original approach for CML was limited by the aspects of diverse owners and distributed development teams not being incorporated in the tool. SoS models were developed in one large joint model, where the different developers had access to all parts of the SoS model. As such stronger tool-support was needed to combine the rigor and level of precision made possible by formal methods, with the collaborative efforts needed between the human stakeholders involved. In [P188] we presented an approach that enables the collaborative development of formal models of SoS via stronger tool-support. The goal was to develop an approach that could keep a focus on core technical aspects of SoS Engineering, while incorporating socio-technical factors. An approach that not only would allow the analysis of structure, behaviour and interactions of systems; but also of the behaviour and interactions between stakeholders. A central element was to keep the technical aspect, such as system structures and interfaces as the core of the development process, but also ensure that the development steps support that different stakeholders have different ideas and wishes for system structures and interfaces. The approach builds on the idea that the development of SoS to a large degree depends on the establishment of a successful collaboration between both systems and humans.

The approach makes use of the Symphony tool to establish a Collaborative Development Environment (CDE) that enables the design and analysis of the SoS topologies and interfaces to be developed and evolved collaboratively between stakeholders. In a nutshell the CDE allows the Symphony tool to work with other globally dispersed instances of itself and enables a complete SoS model to be made up of distributed sub-models describing the individual constituent systems. The CDE allows *collaboration groups* to be created, in which stakeholders can work together on SoS design and development. Each stakeholder has the CML model of their own constituent system encapsulated in their instance of the tool, and can use the CDE to share certain parts of their models, such as interfaces and data types. The CDE allows a range of events to be communicated between the group members to form a modelling session in which the SoS design can be described. Some of the possible events include: proposals for establishing connections between specific constituents' system sub-models; the negotiation of interfaces; and the ability to

identify and attend to inconsistencies between the different versions of the sub-models. All collaborators that are in the Collaboration Group can send events which ensure that the authority is distributed and no one has the control of the group.

The CDE use shared data structure where a Collaboration Project contains the Collaborators, i.e. other Symphony instances and a list of Configurations. A Configuration consists of a range of files that make up the shared sub-model which describes selected parts of the SoS. Each Configuration represents a certain state of the sub-model. Collaborators can adjust their sub-models locally before it is shared with the other collaborators. When the model is set to be shared the current state of the model is locked in a Configuration that is signed with the Collaborator's name and the Configuration is forwarded to the other collaborators. The shared model is considered as a proposal for a particular part of the SoS architecture, and collaborators can use the CDE for approving, rejecting or to evolving the model further. In evolving a model a collaborator can make adjustments to the received model and re-share it with the collaboration group. This enables a type of negotiation to occur between collaborators as models continuously are adjusted and shared.

The connection between the Symphony instances is established using the open-source Eclipse Communication Framework (ECF)[6]. The framework is specifically aimed at establishing communication between distributed applications on the Eclipse platform using various common communication protocols.

Contribution 5. Tool-support for collaborative development of formal models in SoS Engineering via a CDE

3.3.2 Related Work and State of the Art

The use of Computer-Supported Cooperative Work (CSCW) and collaborative development between engineering teams is well researched in the literature [210, 248].

Software Engineering has a focus on Model-Based collaboration tools to support the negotiation between stakeholders in the requirements engineering process [19]. The approaches do however generally use a modelling notation such as UML or a type of structured templates in their collaboration. The focus on collaborative development methods for formal methods in the lit-

[6] `http://www.eclipse.org/ecf/`

erature is however limited. The collaborative development approaches that directly focus on SoS development are also sparse in the current literature. J. Whitehead does however specifically note the need for stronger support in multi-project and multi-organisation collaborations, as the current state is said to raise "an emerging concern in increasingly large systems-of-systems [248].

The use of Distributed Collaborative Environment for Systems Engineering has been proposed by W.K. McQuay that provides a vision for an infrastructure for a system, which combines whiteboards, instant messaging, visualisations, modelling and simulation to improve knowledge sharing between globally dispersed teams [175]. The proposed framework is however at the conceptual level and has not been realised.

Geddes *et al.* propose a way of fostering collaboration in an SoS on the basis of some foundational elements. It is noted that the collaborators in an SoS have an interest in agreeing on a common goal, despite them having their own interpretations and individual goals. Their willingness to participate in the overall SoS serves as a confirmation that they will be willing to contribute to the overall goals, as it will benefit their own individual goals. A key condition for the collaboration is that information is interpreted in the same way and that it is possible to establish a consensus on decision-making. A concrete approach for achieving this is however not provided.

An approach for the distributed collaborative design of SoS with a specific focus on evolution has been proposed [208]. A framework is created out of web services that software agents can use to communicate and exchange information on their interfaces and data types. A process is defined for the information exchange that allows the interfaces and data types to be evolved collaboratively. The approach does not make use of formal models for describing system behaviour, but instead relies on dynamic UI generation, virtual white-boards and a search service for finding the necessary agents for the collaboration.

3.4 Distributed Simulation

The *autonomy* dimension entails that each constituent system and their stakeholders are self-governing and can make individualistic decisions, which makes it difficult to anticipate the systems' behaviours. The behaviour of a system can be revealed through the use of model simulation, and an SoS dimension such as *Emergence* is to a large degree based on system behaviour and the interaction between constituent systems. Stakeholder of the constituent systems may however have a concern for using and sharing models for simulation pur-

poses as precise specifications of the systems may reveal the implementation of their systems. For competitive or confidentiality reasons stakeholders may have an interest in keeping the implementations of their system hidden.

As such the traditional approach for simulation where complete models are simulated on one separate machine is not feasible for all types of systems. As complex systems, such as SoS are represented by systems of other interacting systems with various stakeholders, the use of a confined and secluded model becomes insufficient. A way of approaching these systems is through the use of distributed simulation, where individual stakeholders are responsible for simulating certain parts of the overall simulation [49, 80].

3.4.1 Research Conducted

In [P192] we presented an approach where the connection made between the collaborators in the CDE (presented in Section 3.3) is used to initiate a distributed simulation session between Symphony tool instances. The approach utilises the CDE functionality that allows stakeholders to only share parts of their model with the other collaborators. While the collaborators only share parts of their model, the entire executable models of the constituent systems are available on the different CDEs. This means that the entire SoS can be simulated by the stakeholders simulating their individual parts of the overall SoS model. This enables the stakeholders of the individual constituent systems to contribute to the model simulation without having to share models containing the details of their system's internals.

When initiating and performing the distributed simulation, one CDE instance will act as the *coordinator*, while the other CDE instances have the role of *clients*. The coordinator is responsible for initiating the distributed simulation and for directing how the distributed simulators are to progress. The coordinator has the role of selecting the top process that describes the SoS at the highest level of composition, as well as selecting which processes that are to be simulated by which collaborators. This is done by mapping a process name to a collaborator name using a graphical configuration interface.

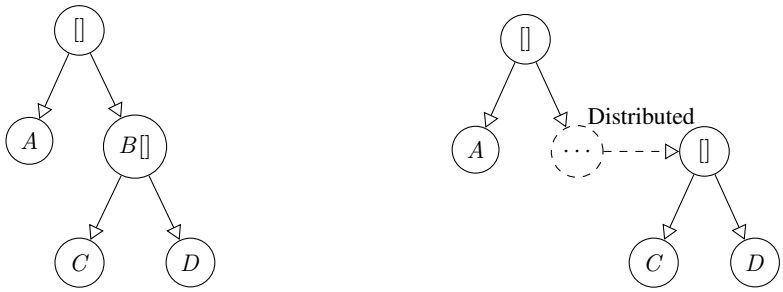
In initiating the distributed simulation the coordinator will send simulation request to all the involved collaborators, and they can then confirm or refuse the request. The distributed simulation requires acceptance from all collaborators before it can be started. If all collaborators accept the participation request, the simulation can be started by the coordinator. During the simulation the coordinator has the task of controlling the flow of execution according to the model specification, however as the possible transition are

computed in the same way as in a normal simulation, the only change is that some of the computations are performed by distributed simulators. Therefore, the distributed simulation can be seen as one model, and it is as such not distinguishable from a non-shared model that is running in a normal simulator. This means that a normal and a distributed simulation is semantically equivalent and follow the semantics of CML [252].

The distributed simulation has been enabled by extending the Symphony simulator with the addition of a network setup between simulators and some modifications to the simulator's method of inspecting and execution behaviours. The network setup is a client–server relationship where the coordinator functions as the server to which the clients can connect. The distributed simulators are connected using a direct Transmission Control Protocol (TCP) [207] connection, while the CDE itself makes use of ECF to configure the distributed simulation.

A custom simulation protocol has been defined that provides message types for handling connections and disconnections from clients. As well as messages that mirror the execution flow that occurs in the normal (non distributed) simulator by providing messages for the inspection and execution of behaviours. The protocol keeps a flow state in order to ensure that an execution message is only allowed if an inspection has previously been processed. The protocol and as such the interactions between the simulators are based on the JavaScript Object Notation (JSON) [46]. The JSON notation was chosen to ensure future interoperability such that the simulator can be connected to heterogeneous systems. This is utilised in the work presented in Subsection 3.5.2.

The modifications to the Symphony simulator have been made in two parts: a) coordinator role, and b) the client role. When the simulator has the coordinator role it will replace all the distributed processes with a skeleton that via the network setup delegates the processing to the distributed simulator on which the distributed process resides. As the protocol mimics the inspection and execution calls of the normal simulator, the coordinator use most of the existing simulator implementation just with some calls being delegated. This is illustrated in Figure 3.5, that shows: 1) the tree structure of a standard simulator and 2) the same tree structure for a distributed simulation where process B is a process located on a distributed simulator. The dashed line represents the network connection between the simulators. When functioning as a client the simulator has been altered such that the inspection and execution of behaviours are initiated from the messages received from the coordinator via the network setup. The clients will then supply the coordinator with the



(a) Standard Simulator (b) Distributed Simulator

Figure 3.5: Illustration of the internal simulator change.

results from their inspection and will rely on the coordinator to control when they can execute their behaviour.

Contribution 6. Enabled distributed simulation of CML models to alleviate the Autonomy of Constituents dimension in SoS modelling

3.4.2 Related Work and State of the Art

The use of distributed simulation as a way of performing, aiding and analysing decision-making and coordination between geographical dispersed teams has been utilised for several decades [228]. In surveying the literature for distributed simulation for modelling formalisms, there are a dominant number of examples which are related to the distributed simulation of DEVS (DEVS is described in Subsection 3.2.2).

Distributed simulation has been implemented between DEVS simulators in the DEVS/OSGi simulation framework using the OSGI module and service platform [205]. It has a strong focus on interoperability and on using a common platform. Through Eclipse, the Symphony tool also makes use of OSGI as its underlying framework.

DEVS/RMI is a distributed simulation approach that make use of Java RMI to connect simulators on the basis of standard DEVS implementation [260]. While it does require minor changes to the distributed models, it works on a large-scale and supports multiple advanced features, such as reconfiguration of simulations during run-time.

DEVS/P2P is a framework that is aimed at letting DEVS simulators run as peers in a peer-to-peer network using the JXTA protocol to establish a distributed simulation [36]. A peer-to-peer simulation protocol for large-scale simulation of DEVS models has also been proposed by Park *et al.* [231]. The focus is on performance and the protocol is to be used on top of existing DEVS simulation frameworks.

Common for all of the DEVS approaches is that they do not have a direct SoS focus and they do not combine the distributed simulation with a CDE.

A distributed simulation approach has also been proposed for the formalism Architecture Analysis and Design Language (AADL) [165]. AADL is aimed at doing analysis of systems and SoS designs by looking at both architectures and runtime characteristics. In order to do the distributed simulation the approach makes use of model transformations from AADL models to a second model formalism; SIGNAL, that is aimed at data-flow analysis [155]. The use of model transformation to establish the distributed communication differs significantly from the one presented in [P192].

3.5 Simulation with External Systems and Code

While formal modelling has been around for more than 45 years [74], the field still face many barriers [93]. Formal modelling has seen difficulties in penetrating and establishing itself as a common engineering methodology within an industrial setting, despite having been extensively researched and applied in large industrial projects [64, 253]. Formal modelling is especially challenged on two fronts: 1) the understanding of the models for all stakeholders and 2) the models' relation to actual system implementation. The former has to do with formal methods foundation in mathematics and its heavy reliance on semantics rules for describing the notations used in the model. Formal modelling notations can be difficult to grasp and understand for stakeholders with no or limited domain knowledge, particularly for stakeholders that are not accustomed to using mathematical tools [131]. The latter relates to the gap that exists between the model of the system and the actual system implementation, where the correspondence between the model and the modelled system becomes opaque [203]. This occurs as a result of the level of detail and fidelity of the model in reference to the modelled system. In general models work at a certain levels of abstraction [136] and as such does not describe all aspects of a given system implementation. When describing complex systems and structures the use of abstractions makes it possible to focus on essentials and emphasise certain properties of the system. Abstract

models can realised by the use of refinement calculus in which the model progressively is transformed into executable programs through a series of refinement steps in which the correctness of the transformation is ensured [9]. For larger systems this can however be a complex and time consuming task.

It can also be the case that it is not necessary or feasible to formalise all aspects of a system [235]. The system may have elements for which its behaviour is already well-defined and there is a high level of trust in the correctness of the functionality, or it may be a legacy system for which it is difficult and time consuming to model its exact behaviour.

These challenges are well-known in the formal modelling literature and for a long time there has been a focus on ensuring that the use of formal modelling becomes part of existing development processes and not be a hindrance to development processes [23]. To address the challenge of understanding there is a need for finding a generic way of approaching formal models that will allow for easier interpretations which will make them better suited for practical applications [218]. Better tool-support has been advocated as a means for dealing with the gaps existing between the model and the modelled system, as well for dealing with models that only capture certain aspects of a system [2]. Better tool support will enable important characteristics to be communicated better and allow for the integration with components and systems that are outside the scope of the formal modelling environment [132]. Improved tool interoperability that will enable form models to be combined with existing system and tools will provide an improved accessibility to formal modelling [65].

In this section we present two approaches that through better tool-support aim: 1) at improving stakeholders' accessibility to formal modelling, and 2) at bridging the gap between models and system implementations. Both approaches have their foundation in improved tool interoperability for enabling systems and technologies outside the modelling environment to be utilised. Subsection 3.5.1 presents an approach for coupling VDM models with externally defined Java implementations, and Subsection 3.5.2 describes an approach for including running systems in the simulation of CML models.

While the literature contains examples of coupling simulations of formal models with external systems and code, distinctions are not made between the two in the same degree as done in this text. Therefore, a combined coverage of related work is presented in Subsection 3.5.3 covering the integration between models and external elements whether it being done for visualisation, simulation or interaction with legacy code.

3.5.1 Simulation with External Code - Research Conducted

In [P190] we presented an extension for the Overture tool that allows VDM models to be linked with executable code that is external to the model. The extension had two objectives: 1) to increase the understanding of models by enabling models to interact with graphical user interfaces in order to enable visualisations of the model, 2) to enable the interaction of VDM models and external executable code, such as external libraries or external constituent systems. The reasoning behind the first objective is that by enabling visualisations and interactions it will becomes easier to convey the functionality and behaviour of a model to stakeholders with limited modelling experience, such as managers and/or domain experts [239]. The second objective is aimed at enabling models to interact with external system or libraries that either have well-defined behaviour or are less critical. Often there will not be as strong a focus on or need for modelling of these constituent systems. It also makes it possible to include legacy systems which behaviour can be difficult to model, as part of the system modelling.

The objectives were reached by making slight adjustments to the Overture tool in order to use the added functionality of the interpreter that simulates the executable VDM models. This gave the tool the functionality to perform interactions between models and external code in two ways. 1) By allowing VDM to call externally running code, essentially by delegating VDM function calls to external Java libraries placed within in the modelling project itself. 2) By letting external Java implementations take control of the VDMJ interpreter, thereby allowing the executions of VDM expressions.

External code can be called from the VDM model through the *External Call Interface* (ECI), and the VDM interpreter can be called and controller remotely by external code via the *Remote Control Interface* (RCI). The *ECI* allows the VDM model to delegate operation invocations to implementations in a Java library without any changes to the VDM syntax or any configuration of communication channels. Using the *RCI* an external Java application can control the execution of a VDM model, for instance through a graphical user interface (GUI). These two methods of integration allow VDM models to be connected to external code, and either use external libraries or use graphical prototypes for presentation or interaction with the model.

Fig. 3.6 illustrates how the *ECI* and the *RCI* can be seen in relation to Java and the VDM interpreter. The Java JVM is the foundation for the VDM interpreter that executes the VDM model. The executing model is capable of initiating the *ECI* which directly calls into Java.

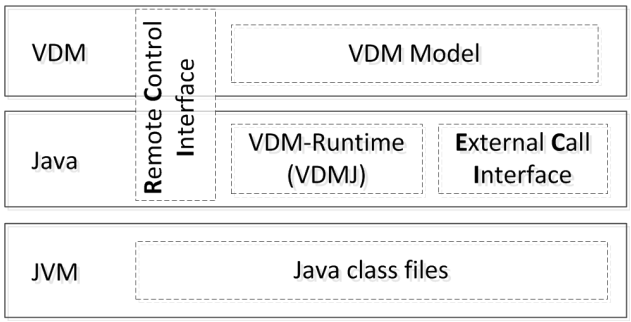


Figure 3.6: Overview of the methods in relation to the model

The case study of a simple traffic infrastructure system was used to demonstrate the techniques. In the system a group of traffic planners were attempting to analyse the optimal planning of bus routes within a city. The various infrastructure maps of the city roads and current bus routes were stored in a relational database. The system was described in a VDM model in which the existing data from the database was loaded into the model at run-time using the *RCI*. The model described the infrastructure with bus stops, connected roads and bus movements. The model simulated a constant inflow of new passengers and the purpose of the simulation was to measure the time passengers had to wait for a bus.

To give an overview of the modelled being simulated a graphical representation was created in Java Swing. Fig. 3.7 shows a screen-shot of a running animation, where a selected map is displayed with the buses indicated by squares on the roads and the waiting passengers as the circles to the right. The graphical representation is purely an overlay on top of the model; everything is continually checked and validated by the executing VDM model, from bus movements to the passenger count. The case study makes use of both the *ECI* and the *RCI*, as the VDM model can notify and update the GUI of data changes and the Java implementation can load maps from the database into the model as well as adjusting the passenger inflow during simulation.

Contribution 7. Enabled VDM models to link with and execute external code to support systems not modelled in VDM

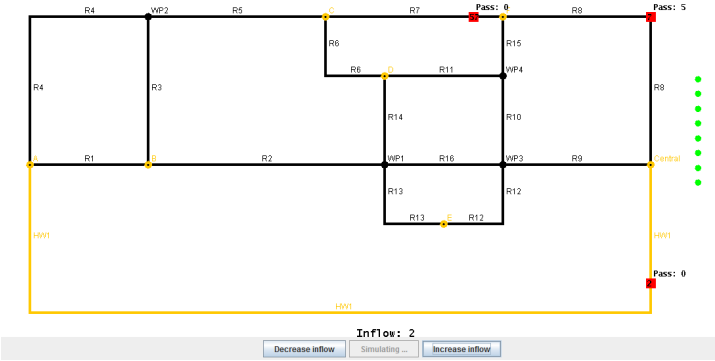


Figure 3.7: Screenshot of the models graphical representation

Contribution 8. Enabled external code to control the VDM simulator in order to perform run-time manipulation of models with the purpose of creating interactive visualisation of models

3.5.2 SoS Simulation with External Systems - Research Conducted

In [P154] we presented an approach that allows a combination of CML models and externally running systems in simulations performed in the Symphony tool. This means that a part of the constituent systems can be described in CML, while another part of the constituent systems are actual running systems. A tool extension of the Symphony tool enables the interaction between the CML simulator and the executing external system. The purpose is to simulate SoS models in which the behaviour of some constituent systems only can be obtained via the actual running constituent system.

This is needed in situations where: 1) the independent owners of the systems are not willing to share all knowledge on their system implementation, making it difficult to create models with the correct behaviour. This situation occurs when there are conflicting interests between the stakeholders of the individual constituent systems or because parts of the SoS is delivered by a supplier that has no interest in the SoS. The owners of the independent constituent systems may not be willing to share implementation details for competitive reasons, and suppliers may just have delivered a COTS product for which they have no interest in delivering documentation on its internals. In these cases it may be difficult to create sufficiently descriptive formal models.

2) Some of the constituent systems in the SoS may be legacy systems for which documentation of their realisation is no longer available and there is no precise description of their internals.

The tool has been extended in such a way that from a CML simulator perspective the external running system will act as a constituent system defined in the overall SoS model. This means that concrete system implementations can ben run and executed in parallel to the simulation of the model and the behaviour of both the model and the system can affect each other. Given that each constituent system has its own specified behaviour and specified way of communicating with the other constituent systems in the SoS model it is possible to make the external system act as if it was modelled. This was achieved by having the external systems being defined as an empty process definition in the model, that then is replace with a skeleton that delegates the interaction to the external system.

This builds on the same principles as the approach presented in Section 3.4, but it enables the simulation with external systems instead of enabling distributed simulation between homogenous simulators. The approach has many of the same goals as what was presented in Subsection 3.5.1, but it differs in that way the delegation occurs at the communication between systems via the processes in CML and by network communication being used to establish connections to an adapter connected to the external constituent system, instead of loading libraries that can be invoked.

During the simulation the Symphony will act as a *coordinator* and be in control of the simulation, while the external system will act as a client. The simulation is configured by selecting the processes that will be handled by an external system, as well as address and port information for the clients to connect to the coordinator.

In order for the external system to be part of the simulation an adaptor is needed between the system implementation and the network setup to the simulator. The core part of this involves the implementation of the simulation protocol that requires a mapping of the types of the concrete programming language to the CML types described in the simulation protocol and a mapping from the protocol messages to operations in the external system. The protocol essentially allows the simulator to send messages that can invoke and change the system state of the external system, and the external system can in return provide the simulator with a list of the possible events that the external system can perform.

The approach was examined through a Bang & Olufsen (B&O) case study involving a Audio/Video (A/V) network that can establish connection be-

tween devices (such as audio, video and legacy audio products). A key part of the system is an A/V control that manages a range of heterogeneous devices that can stream and render media. The behaviour between the devices and the overall network is defined through streaming contracts that specify the interactions between the devices. A CML model of the A/V network streaming contracts was created that describes the semantically defined transition rules for distributed state synchronisation and distributed operation calls used in the streaming.

In the study a B&O developed C++ implementation of the streaming contract was used as the externally running system. Some adapter code written in C++ was added to the implementation to handle the network setup as well as the mapping for states, types and operation logic from the C++ to CML model. Furthermore, a state machine was implemented in which the channel events of the CML model translate into operations calls in the C++ implementation. Being able include the running system as part of the model simulation enabled B&O to improve the trust in the system design, as formal analysis can be performed before costly implementations are made of the entire system.

Contribution 9. Tool-support enabling the inclusion of running system implementations in the simulation of CML models

3.5.3 Related Work and State of the Art

In the literature little distinction is made between the coupling of model simulation with external systems and the coupling of model simulation with external code. Therefore, the presented related work covers both types of coupling between models and external systems/code, whether it being done for the purpose of visualisation, simulation or interaction with legacy code.

Frohlich and Larsen describe a functionality in the VDMTools [143] tool suite for VDM that makes it possible to execute specifications consisting of both specification and externally specified C++ code [79]. The external code is loaded from Dynamic Link Libraries (DLLs) and new syntax is introduced in VDM that defines an implementation module type in which an export section can define signatures for functions defined in external libraries. Type conversion functions are used to transform the value types, used by the interpreter, to the values type of the C++ code, and vice versa. The work pre-

sented in Subsection 3.5.1 is based on this approach and has similar dynamic semantics but it does not require a modification of the VDM syntax.

The approach by Frohlich and Larsen makes it possible to interact with external code, but is does not allow for external code to interact with the model, for instance via an interactive GUI. VDMTools does however provide an interface that allows a CORBA [200] client to communicate directly with the VDMTools interpreter and pass VDM expressions that will be evaluated in the running model [234]. The use of CORBA allows for a greater heterogeneity in the range of external systems that can interact with the VDM model. The implementation of a CORBA client can however give a higher complexity in the initial implementation, than the plain Java integration offered by the approach in Subsection 3.5.1.

Another formalism that enables integration with external processes is Coloured Petri Nets (CPN) [116]. Communication between CPN models and external processes can be obtain through Comms/CPN, which is a Standard ML library for the Design/CPN tool [81]. The Comms/CPN library enables two-way communication between the CPN model and the external process using TCP/IP, by defining generic send and receive-functions which accept a byte stream of data. Encoding/decoding functions has to be implemented to marshal data for transmission. The Comms/CPN library's use of generic data streams over TCP/IP allows heterogeneous clients to interact with the simulator, but it does require the external process to implement a mapping of the received data into concrete invocations in the model.

CPN Tools, the successor of Design/CPN, provides a similar functionality that enables the integration of CPN models with external applications [247]. A Java interface enables the processing of models by communicating with the CPN simulator via a TCP/IP stream using a protocol with a custom packet format. All communication is wrapped in a high-level simulator object, which contains methods for evaluating expressions and processing models.

The formalism Event-B [3] is supported by the tool B-Motion Studio that provides a visual editor for creating visualisations of models [139]. The editor enables graphical elements to be linked to the model by using Event-B expressions as gluing code. Having an editor for creating the visualisations allows for simpler and faster creation of graphical representations without requiring specific knowledge on graphical programming. The easy of construction does however come with the price of being limited to the types of visualisations provided by the tool.

4

Conclusion

This chapter concludes this dissertation by summarising and evaluating the research contributions that has been achieved in this PhD project, as well as supplying an outlook on future work. The objectives of the dissertation are related to the convergence of SoS knowledge as a means of identifying key SoS challenges and on strengthening the field of SoS Engineering through stronger tool-support.

4.1 Introduction

This dissertation provided classifications and approaches aimed at strengthening the field of SoS Engineering. A classification has been created which consists a range of dimensions that characterise the SoS engineering field. With a basis in these dimensions, key challenges of SoS Engineering been derived and the efforts for a strengthened SoS Engineering field have been sought in three main themes: integration, modelling and collaboration. This has led to an SoS classification for design patterns and various approaches for improving the tool-support for formal modelling techniques.

The purpose of this chapter is to evaluate the outcome of the research presented in this dissertation and to provide a perspective on how the research can be taken further. Section 4.2 summarises the research contributions, followed by an evaluation of what extent the objectives of the dissertation has been reached in Section 4.3 and an assesment on how the contributions cover the SoS dimensions in Section 4.4. Finally, future work is described and presented in Section 4.5.

4.2 Research Contributions

A total of 9 research contributions have been identified in this PhD project. The research contributions can be grouped into three main areas, with one

initial-level contribution ([C1]) leading into the three groups. The contributions are shown in Figure 4.1 which illustrates how some contributions form a basis for others and how some contributions complements other contributions. The three groups are focused on integration (grey) , collaboration (red) and tool-support (blue).

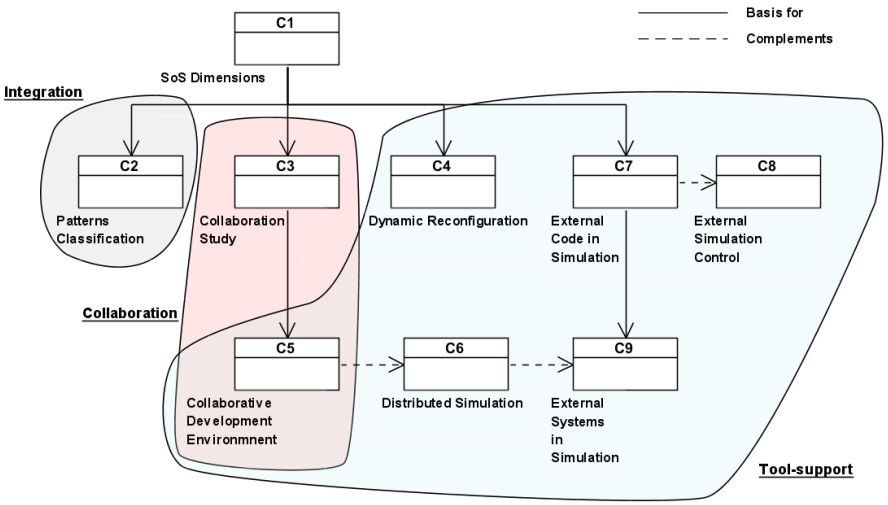


Figure 4.1: Contribution Overview

The top-level contribution and the contributions in the integration and collaboration originate from Chapter 2, while all contributions related to tool-support come from Chapter 3.

Contribution 1. Converged terms of the literature to identify dimensions that enable SoS to be positioned in the SoS Engineering field

The survey presented in Section 2.1 showed the while the SoS literature is voluminous it contains such a broad range of terms and concepts in the description of SoS that it becomes difficult to comprehend. This makes it difficult to place systems and applications within the field. Contribution [C1] encapsulates the work that was done in converging the substantial SoS vocabulary from the surveyed literature into eight dimensions, as described in Section 2.2.

The remaining contributions are presented on the basis of their grouping in the sections below.

4.2.1 Integration

When looking at the SoS literature it quickly becomes apparent that integration and interactions between constituent systems are a crucial part of SoS Engineering.

Contribution 2. Classification for identifying design patterns that aid the integration challenges in SoS Engineering

Section 2.4 presented a classification that can be used to methodically characterise patterns for SoS integration [C2]. The classification enables SoS applications to be matched to design patterns through a number of categorisations. Each categorisation describes a concrete property that relates to the concrete needs and capabilities of an SoS. The classification makes use of four categorisations for determining the SoS development context and five categorisations for determining the technical context.

4.2.2 Collaboration

Both the experience gained by working with SoS development during the PhD project and the literature showed that an SoS not only is dependent on the interactions between the constituent systems, but also on the interactions between human stakeholders. As such the SoS Engineering field not only faces technical challenges, but also socio-technical.

Contribution 3. Identified challenges with collaboration in relation to using formal modelling for SoS Engineering

Section 2.5 describes the result from a small case study focused on the challenges in doing collaborative development of formal models of SoS. The study showed that the involved formal modellers experienced challenges in exchanging model information and on agreeing on design decisions [C3]. The results from the study later resulted in the development of several of the approaches for the improved tool-support for formal modelling presented in Chapter 3 and summarised below.

4.2.3 Tool-support for Formal Modelling

It has been a key focus of the PhD project that stronger tool-support for formal modelling techniques aimed at SoS challenges can strengthen the SoS

Engineering field. This has led to the majority of the research contributions of the PhD project being achieved in approaches for improved tool-support.

The contributions made in connection to improved tool-support have relations to each other. They are all centred on a formal modelling tool to which they supply some type of advancement. One enable the modelling notation to express dynamicity [C4], while others make use of connectivity between tool instance to establish a Collaborative Development Environment (CDE) [C5] and perform distributed simulation [C6]. Finally, some open up the model to the world outside the modelling environment by enabling the formal model to invoke external code [C7], or to let the model simulation be controlled from an external source [C8], as well as for running systems to be included in the model simulation [C9]. The difference between [C7] and [C9] is that the former requires changes to the model in order to invoke the external code, while the latter requires some adapter or glue-code between the simulated model and the external system.

The relationship is illustrated in Figure 4.2 where the individual contributions are put in relation the modelling tool and its environment.

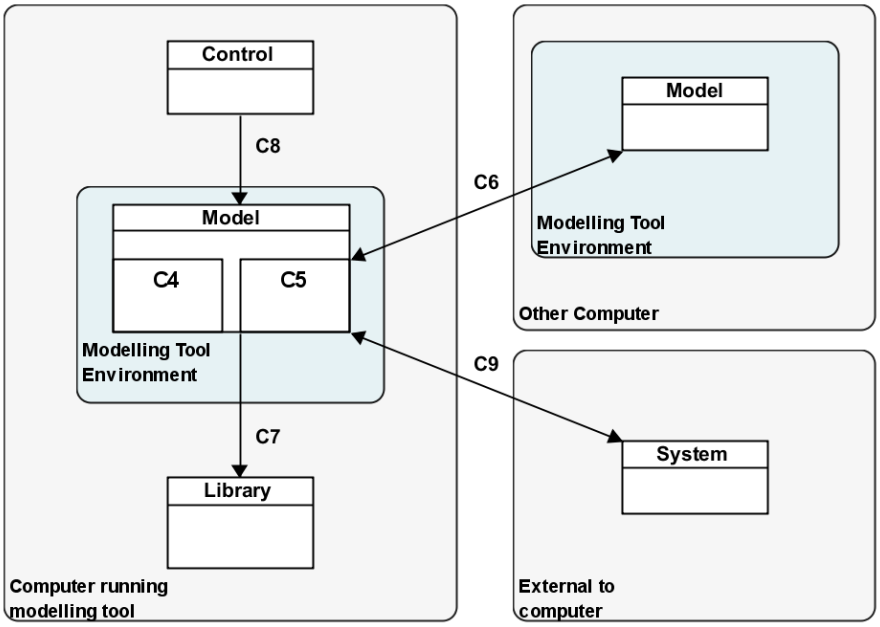


Figure 4.2: Relations of Modelling Tool Contributions

Contribution 4. Approach for expressing and simulating the SoS dimensions of Evolution and Dynamic behaviour in formal SoS models

Section 3.2 describes a language and tool extension for VDM-RT that enables the dynamic reconfiguration of the SoS topology to be expressed and simulated [C4]. With the extension it became possible to describe the overall system architecture of the SoS in VDM-RT with dynamical changes and to perform simulations that could reveal the evolving behaviour of the modelled system.

Contribution 5. Tool-support for collaborative development of formal models in SoS Engineering via a CDE

The insight gained from the collaboration study in Section 2.5 led to the development of a CDE extension of the Symphony tool that could support formal modelling in some of the technical and socio-technical challenges of SoS Engineering [C5]. The CDE presented in Section 3.3 made it possible to support the analysis of structure, behaviour and interactions of systems as well as the behaviour and interactions between stakeholders.

Contribution 6. Enabled distributed simulation of CML models to alleviate the Autonomy of Constituents dimension in SoS modelling

On the basis of the connectivity established between Symphony tool instances via the CDE, an approach for distributed simulation was presented in Section 3.4. Having distributed simulation allowed the entire SoS model to be simulation, without the individual stakeholders having to share complete models describing the internal details of their systems [C6].

Contribution 7. Enabled VDM models to link with and execute external code to support systems not modelled in VDM

Contribution 8. Enabled external code to control the VDM simulator in order to perform run-time manipulation of models with the purpose of creating interactive visualisation of models

Subsection 3.5.1 describes an extension for the Overture tool that allows VDM models to be linked with external code in two different ways: one that enable VDM models to invoke code defined in external libraries [C7] and a second that enables external control of the simulation by given input to the simulated model [C8]. The purpose of the approach were to make models

easier accessible to stakeholders by enabling visualisations and more interactive models, and to enable interactions between models and external libraries or systems that have not been formally modelled.

Contribution 9. Tool-support enabling the inclusion of running system implementations in the simulation of CML models

Subsection 3.5.2 describes an approach for combining CML models and externally running systems in simulations performed in the Symphony tool, thereby enabling parts of the constituent systems to be described in CML and other parts of the constituent systems to be actual running systems [C9].

4.3 Evaluation of Contributions

In this section, the research contributions described in Chapters 2 and 3 are evaluated with respect to the evaluation criteria listed in Section 1.7.

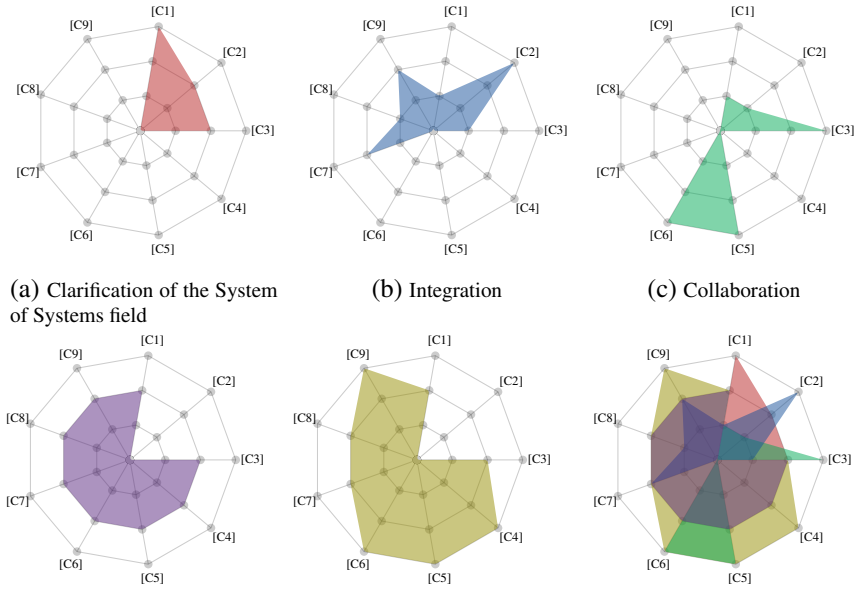
The relationships between the research contributions and the evaluation criteria are presented in Figure 4.3. The spider chart visualises an informal assessment of how the individual research contributions fulfil the respective evaluation criteria. The scale used in the figures indicates the extent to which the contributions fulfil the criteria, and as such the closer the graph is to the edge of the web, the better the criterion is covered. Figure 4.3f gives a combined view by overlaying all figures to show how the pooled research contributions cover the evaluation criteria.

Evaluations of the research contributions with respect to the evaluation criteria are given below. The evaluations relate to each of the individual subfigures of Figure 4.3, with the exception of the combined view Figure 4.3f.

4.3.1 Clarification of the System of Systems field

On the basis of the survey presented in Section 2.1 eight dimensions were established that encapsulate key characteristics of SoS [C1]. As these eight dimensions curtail the number of terms and concepts from the SoS literature they make the field more accessible and provide a way to position applications of SoS in the SoS field and identify engineering challenges.

By providing a classification for characterising patterns for SoS integration a range of categorisations were provide that related to the concrete needs



(a) Clarification of the System of Systems field

(b) Integration

(c) Collaboration

(d) Modelling

(e) Improved tool-support for SoS Engineering

(f) Combined Overview

Figure 4.3: Relation between contributions and evaluation criteria.

and capabilities of an SoS [C2]. These help to clarify certain aspects of the developmental and technical contexts of SoS.

The study on collaborative development of formal models for SoS provided an insight into the socio-technical challenges of SoS Engineering [C3]. This contributed to making the role of stakeholders clearer.

4.3.2 Integration

The classification of design patterns for SoS construction was directly focused on integration [C2], and as such provided the strongest contribution to enhancing the SoS Engineering approach to integration. The tool based approaches presented in Section 3.5 was focused on combining the model with externally define code [C7] or running systems [C9]. The approaches allow for formal modelling to be used in the study of integration challenges, for instance with legacy systems for which it provides a way of analysing and testing potential integration concerns without having to build the entire system.

4.3.3 Collaboration

With a basis in the SoS dimensions [C1], the study in collaborative development of formal models [C3] was directly aimed at investigating the interactions between human stakeholders during the engineering of a SoS using the CML modelling notation. The study supplied a central contribution in clarifying the key challenges of developing SoS models collaboratively.

The study led to the development of the CDE [C5] and the distributed simulation [C6] that both were tool developments aimed at attending to the challenge of collaboration.

4.3.4 Modelling

On the basis of the SoS dimensions [C1] the survey presented in Section 2.1 gave assessments on how formal modelling could be used in SoS Engineering and pointed to where further research is needed.

The study in collaborative development of formal models [C3] revealed some of the challenges that exists in using formal models within SoS Engineering, and directly gave suggestions that could be used to strengthen the use of modelling in SoS Engineering. These were later realised through improved tool support.

Likewise, it can be assessed that all the tool extensions presented in Chapter 3 made contributions to breaking down the barriers for the adoption of formal models in the SoS Engineering field.

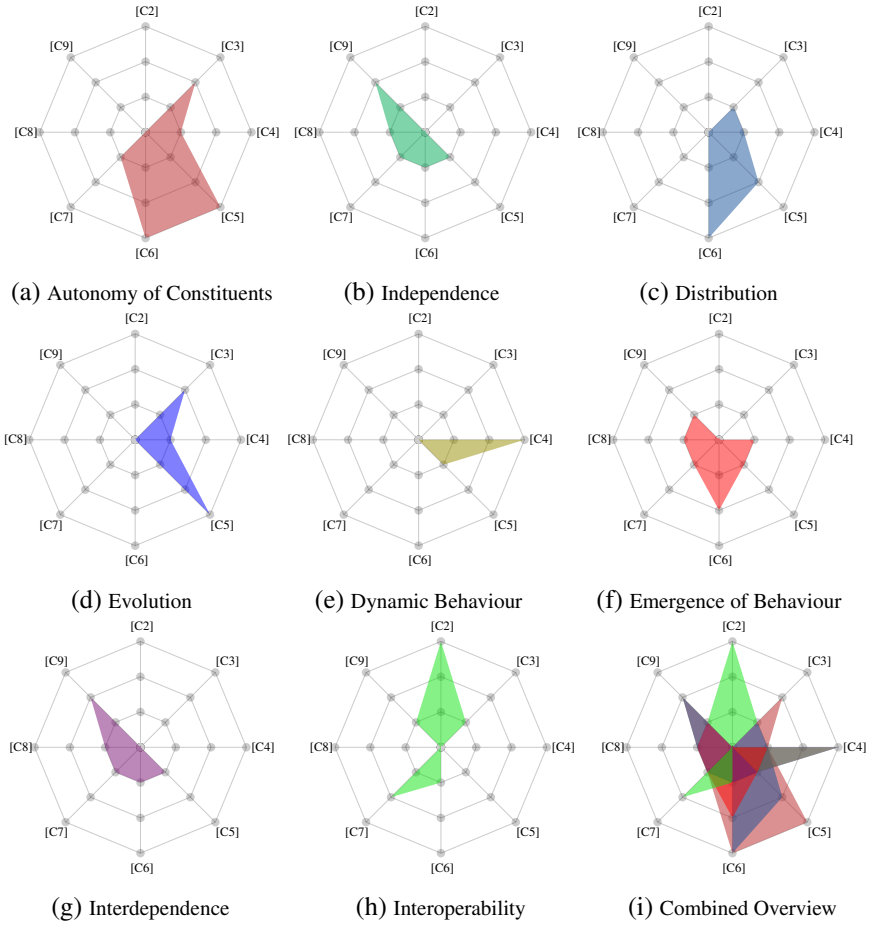
4.3.5 Improved tool-support for SoS Engineering

The SoS dimensions [C1] and the collaborative formal model development study [C3] was the basis for the approaches and extensions that were made in order to strengthen the SoS Engineering field.

The approaches and extensions that made it possible to express dynamic reconfiguration [C4], establish a CDE [C5], perform distributed simulation [C6] and allow for running systems to be included in the model simulation [C9] all had a direct focus on SoS Engineering. Enabling the formal model to invoke external code [C7] and the model simulation to be controlled from an external source [C8] in order to provide interactive visualisations of models, does strengthen the tool support for formal models, but the approaches do not have a direct SoS Engineering focus.

4.4 SoS Dimensions and Contributions

In this section, the research contributions evaluated above are related to the SoS Dimensions [C1]. Figure 4.4 visualises an informal assessment of how the individual research contributions cover and support the respective SoS dimensions. This is visualised in the same fashion as the evaluation criteria above, with the exception that contribution [C1] has been excluded, as it forms the basis for the SoS dimensions being assessed.



(a) Autonomy of Constituents (b) Independence (c) Distribution

(d) Evolution (e) Dynamic Behaviour (f) Emergence of Behaviour

(g) Interdependence (h) Interoperability (i) Combined Overview

Figure 4.4: Relation between contributions and SoS dimensions.

Autonomy of Constituents

The *Autonomy of Constituents* entails that the constituent systems have diverse stakeholders that in many cases only will be governed by their own rules and goals. This means that the human stakeholders need to collaborate on making the constituent systems interact. The collaboration may also be hindered by confidentiality concerns where the individual stakeholders may not be willing to share the internals of their respective systems.

The collaboration study examined this aspect in connection with using formal modelling for SoS engineering [C3] and provided insight into the challenges. The CDE [C5] and the distributed simulation [C6] was developed to support diverse stakeholders in developing their SoS models collaboratively.

Independence

Independence implies that a given constituent is individually developed and can operate while being detached from the rest of the SoS. This means that it may be difficult to develop formal models of all the constituent systems in the SoS as the required knowledge and insight may not be available. It may also entail a high degree of heterogeneity between the systems which require the adaptation of interfaces, protocols and standards to establish interaction between them.

The approaches made for improved tool-support all address some of the challenges of this dimension, as they either aid stakeholders in communicating and agreeing on interfaces, or they enable some degree of interaction with external systems or code implementations that are not modelled. For instance, the CDE [C5], the distributed simulation [C6] and the visualisation of models [C8] support the collaboration between independence stakeholder, while the integration between model and external systems [C9] and code [C7] allow for inclusion and analysis of systems that have not been formally modelled.

Distribution

The *Distribution* dimension means that both systems and human stakeholders in an SoS may be globally dispersed, which raise challenges in ensuring precise and efficient communication to support the design process.

Working with distributed teams on formal modelling was examined in the collaborative study [C3], while the CDE [C5] and the distributed simulation [C6] was developed to support the development teams being distributed. The distribution between constituent systems entail challenges with connectivity,

concurrency and resilience to failure, which to a certain degree can be analysed with the extension made to VDM-RT [C4], as it allows models to express connections being created and lost.

Evolution

An SoS will be under constant *evolution* throughout its lifetime in which there will be changes to its topology and to the capabilities of the constituent systems.

The collaborative study [C3] showed that the collaborators did have challenges in deal with evolution, especially when it was manifested via changes made by other collaborators. The CDE includes [C5] functionality that aid developers in identifying changes as they occur and it aids in incorporating these changes into the overall SoS.

Dynamic Behaviour

The *Dynamic Behaviour* of SoS refers to the changes to topology and composition that occur as a result of dynamic modification of SoS architecture and constituent systems' interfaces.

The dynamic reconfiguration extension for VDM-RT [C4] made it possible to express and simulate the addition of new systems as well as changes to the SoS topology.

Emergence of Behaviour

Emergence of Behaviour refers to the behaviours that arise as a result of the synergistic collaboration of constituents. This is one of the most challenging parts of SoS development as it is an aspect that cannot directly be engineered.

Simulation of models can be used to reveal the emergent behaviour in an SoS, as this occurs as a result of the behaviours of the constituent systems and the interactions between them. All of the approaches aimed at strengthening the tool support for SoS modelling make a minor contribution to handling the *Emergence of Behaviour* dimension.

The CDE [C5] and the distributed simulation [C6] enables the simulation of the dispersed constituent system models. Being able to the integrate a model with external systems [C9] and code [C7], as well as express dynamic reconfiguration [C4] and do visualisations of models [C8] all add to strengthening the simulation capabilities.

Interdependence

The *Interdependence* dimension refers to the mutual dependencies that will occur between the independent constituent systems as they have to rely on each other to reach the common goal of the SoS.

Interestingly enough, the contributions mapping to *Interdependence* is identical to that of *Independence*. This has to do with the two dimensions being related, as the *Interdependence* comes from the trade-offs between the degree of independence of the constituent systems in relation to the interdependence required to reach the common goal of the SoS. As such *Interdependence* can only be studied once it has become apparent what the role and impact *Independence* has on the SoS.

Interoperability

The dimension of *Interoperability* is concerned with the coupling and interactions between heterogeneous constituent systems in the SoS. The integration between the constituents requires the adaptation of interfaces, data types and protocols in order to establish bridges between both legacy and newly designed systems.

The classification of design patterns to be used in SoS Engineering had a direct focus on the integration between systems [C2], while some of the tool based approaches was focused on combining the model with externally define code [C7] or running systems [C9]. As such they all contribute to enhancing the way SoS Engineering handles the *Interoperability* dimension.

4.5 Future Work

This section gives some directions for future work. These are grouped into three main themes of this dissertation: integration, modelling and collaboration.

The research and development challenges of SoS Engineering are widespread and the research described in this dissertation only focuses on small parts of the field. The performed research has been focused on strengthening the field of SoS Engineering by working within the themes of integration, modelling and collaboration. Within these themes approaches for classification and improved tool-support have been developed, but they do not offer complete solutions to these challenges. This suggests an array of research directions that can be pursued in order to further develop the approaches.

4.5.1 Integration

The purpose of the classification of design patterns was to aid engineers in dealing with SoS integration challenges by identifying concrete design patterns on the basis of a range of categorisations. In [P124] we identified 15 patterns related to integration and placed them in the classification. In order to strengthen this approach there is a need for building up a bigger repository of patterns that engineers can use as a reference. Having a stronger repository would also make it more feasible to create a study of the classification in concrete SoS development on a larger scale.

In addition to this it would be relevant to investigate how the classification can be used to address other challenges in SoS Engineering besides integration.

4.5.2 Modelling

In the survey [P189] a range of further research directions were given for the next steps towards a strengthened foundation for Model-Based SoS engineering [C2]. These were based on the limitations of current formal modelling notations when seen in relation to the SoS dimensions and the engineering challenges they entail.

The ability to include the *Dynamicity of Behaviour* dimension by being able to express the alternation of system topologies is an area in which formal modelling techniques could be strengthened. The current version of CML is for instance not well suited for expressing dynamic changes in the system structure. In addition to the modelling notation being able to express dynamic behaviour, simulators should also have mechanisms for keeping track of changes, for instance by visually indicating the current topology. As such, a future research direction is to investigate the combination of the functionality in contribution [C4] and [C8] further.

Many formal modelling notations only consider the current systems and does not have a very strong support for the dimension of *Evolution*. An SoS will evolve and some of the changes may require new facets of the system that needs to be expressed. One possible way of dealing with this challenge is for the SoS modelling language to have features that enable well-founded extensions of the language to be made during the development of the SoS. If the semantics are to be preserved this is by no means trivial, so reaching the scientific basis for this is a long-term goal.

While we have made suggestions for including the aspects of the *Interoperability* dimension, one part that is not well-addressed is joint simulations

with heterogeneous models that use different notations. Having this ability could aid the construction of an SoS by allowing the integration of various heterogeneous models defined for different constituent systems. One could claim that the approach provided by [C9] could be used, but this would still not be an easy task as it would require some type of adapter that could make the semantic mapping between the various heterogeneous models.

This PhD project has attempted to develop a set of approaches that could strengthen the tool-support for SoS Modelling. Several of the approaches do however require a fair degree of manual labour and legwork to get good results. A very interesting future research direction would be the development of a higher degree of tool automation. For instance by auto-generating larger parts of the adaptor code needed between the simulator and the external system in relation to contribution [C9]. Additionally, it would be interesting to utilise the possibilities of creating visualisations easier, such as is seen in B-Motion Studio [139] in relation to [C8].

4.5.3 Collaboration

The study on formal SoS models being developed collaboratively only outlined the elementary aspect of the socio-technical challenges in SoS Engineering. Further studies are needed to create a more substantial basis on which the research in collaborative formal modelling can be developed.

As the author and the involved research environment had a little experience in performing this type of studies, an important step in a future study would be to include other researchers that can deliver a stronger ethnographical perspective and provide a strong set of qualitative social science methods.

With industrial acceptance and adoption being an important factor in formal methods, an important next step is performing studies on how the collaborative challenges manifest themselves in the development of an SoS in an industrial setting. This would optimally be performed with an SoS application that involves constituent systems originating from diverse parts of engineering, as this would make it possible to conduct a study with multiple groups of engineers with different backgrounds.

Part II

Publications

5

Model-based Engineering of Systems of Systems

The paper presented in this chapter has been submitted as a journal paper to ACM Computing Surveys.

[P189] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock and Jan Peleska. *Model-based Engineering of Systems of Systems*. Submitted to ACM Computing Surveys, September 2013.

6

Understanding Patterns for System of Systems Integration

The paper presented in this chapter has been accepted as a peer-reviewed conference paper.

[P125] Rick Kazman, Klaus Schmid, Claus Ballegaard Nielsen and John Klein. *Understanding Patterns for System of Systems Integration*. 8th International Conference on System of Systems Engineering (SoSE), IEEE SoSE 2013, 141-146, June 2013.

7

Challenges in Collaborative Formal Modelling of System of Systems

The paper presented in this chapter is a draft paper to be submitted to the International Journal of System of Systems Engineering.

[P186] Claus Ballegaard Nielsen, Claire Ingram, André Didier, Uwe Schulze, Stefan Hallerstede, Andrew Galloway and Peter Gorm Larsen. *Challenges in Collaborative Formal Modelling of System of Systems*. Draft paper to be submitted to the International Journal of System of Systems Engineering, 2014.

8

Extending VDM-RT to Enable the Formal Modelling of System of Systems

The paper presented in this chapter has been accepted as a peer-reviewed conference paper.

[P187] Claus Ballegaard Nielsen and Peter Gorm Larsen. *Extending VDM-RT to Enable the Formal Modelling of System of Systems*. Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012, July 2012. 978-1-4673-2974-3

9

Collaborative Formal Modeling of System of Systems

The paper presented in this chapter has been accepted as a peer-reviewed conference paper.

[P188] Claus Ballegaard Nielsen and Peter Gorm Larsen. *Collaborative Formal Modeling of System of Systems*. IEEE SysCon 2014, March 2014. IEEE Best Student Paper Award

10

Distributed Simulation in System of Systems Modelling

The paper presented in this chapter has been accepted as a peer-reviewed conference paper.

[P192] Claus Ballegaard Nielsen, Kenneth Lausdahl and Peter Gorm Larsen. *Distributed Simulation of Formal Models in System of Systems Engineering*. 4th IEEE track on Collaborative Modelling and Simulation in IEEE WETICE 2014, June 2014.

The content of this chapter has been excluded due to copyright restrictions, but can be obtained though the respective publisher.

11

Combining VDM with Executable Code

The paper presented in this chapter has been accepted as a peer-reviewed conference paper.

[P190] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. *Combining VDM with Executable Code*. In Abstract State Machines, Alloy, B, VDM, and Z, volume 7316 of Lecture Notes in Computer Science, pages 266– 279, 2012. Springer-Verlag. ISBN 978-3-642- 30884-0.

12

System of Systems Modelling with External Systems

The paper presented in this chapter has been submitted to peer-reviewed conference.

[P154] Kenneth Lausdahl, Claus Ballegaard Nielsen and Klaus Kristensen. *Including Running System Implementations in the Simulation of System of Systems Models*. Submitted to Software Engineering and Formal Methods (SEFM) 2014, March 2014.

Bibliography

[1] Russ Abbott. Open at the Top; Open at the Bottom; and Continually (but slowly) Evolving. In *System of Systems Engineering, 2006 IEEE/SMC International Conference on*. IEEE, April 2006.

[2] Jean-Raymond Abrial. Formal methods: Theory becoming practice. *Journal of Universal Computer Science*, 13(5):619–628, May 2007.

[3] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[4] Russell L. Ackoff. Towards A System of Systems Concept. *Management Science*, 17(11):661–671, July 1971.

[5] Russell L. Ackoff. "The systems revolution. *Long Range Planning*, 7(6):2–20, July 1974.

[6] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic. A Survey of Social Software Engineering. In *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, pages 1–12, Sept 2008.

[7] Kevin Ashton. That 'Internet of Things' Thing, in the real world things matter more than ideas. *RFID Journal*, (22):65–78, 2009.

[8] Robert Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, Princeton, NJ, 1997.

[9] Ralph-Johan Back. *On the Correctness of Refinement Steps in Program Development*. PhD thesis, Åbo Akademi, Department of Computer Science, Helsinki, Finland, 1978. Report A–1978–4.

[10] W. Clifton Baldwin, Wilson N. Felder, and Brian J. Sauser. Taxonomy of increasingly complex systems. *International Journal of Industrial and Systems Engineering*, 9(3):298–316, Jan 2011.

[11] Yaneer Bar-Yam, Mary Ann Allison, Ron Batdorf, Hao Chen, Hoa Generazio, Harcharanjit Singh, and Steve Tucker. The Characteristics and Emerging Behaviors of System of Systems. *NECSI: Complex Physical, Biological and Social Systems Project*, pages 1–16, 2004.

[12] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 3rd ed.* Addison-Wesley, 2013.

[13] Nick Battle. VDMJ User Guide. Technical report, Fujitsu Services Ltd., UK, 2009.

[14] Brian J.L. Berry. Cites as Systems within System of Cites. *Papers and Proceedings of the Regional Science Association*, 13(1):149–163, January 1964.

[15] D. Bjørner and C.B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978.

[16] Dines Bjørner. Pinnacles of software engineering: 25 years of formal methods. *Annals of Software Engineering*, 10:11–66, 2000.

[17] John Boardman and Brian Sauser. System of Systems – the meaning of "of". In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 118–123, Los Angeles, CA, April 2006. IEEE.

[18] Barry Boehm. Some Future Trends and Implications for Systems and Software Engineering Processes. *Systems Engineering*, 9(1):4–13, 2006.

[19] Barry Boehm and Alexander Egyed. Software requirements negotiation: some lessons learned. In *Proceedings of the 20th international conference on Software engineering*, ICSE '98, pages 503–506, Washington, DC, USA, 1998. IEEE Computer Society.

[20] Barry W. Boehm. Software engineering. *IEEE Transactions on Computers*, (12), December 1976.

[21] Richard C. Booton and Simon Ramo. The Development of Systems Engineering. *Aerospace and Electronic Systems, IEEE Transactions on*, AES-20(4):306–310, July 1984.

[22] Kenneth E. Boulding. General Systems Theory–The Skeleton of Science. *Management Science*, 2(3):197–208, April 1956.

[23] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(3):34–41, July 1995.

[24] J. W. Bryans, J. S. Fitzgerald, C. B. Jones, and I. Mozolevsky. Dimensions of Dynamic Coalitions. Technical Report CS-TR-963, Newcastle University, School of Computing Science, May 2006.

[25] J. W. Bryans, J. S. Fitzgerald, C. B. Jones, and I. Mozolevsky. Formal Modelling of Dynamic Coalitions, with an Application in Chemical Engineering. In T. Margaria, A. Philippou, and B. Steffen, editors, *IEEE-ISoLA 2006: Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 90–97, November 2006.

[26] Dennis M Buede. *Engineering Design of Systems - Models and Methods*. Wiley, 2011.

[27] Donald L. Burkhard and Per V. Jenster. Applications of computer-aided software engineering tools: Survey of current and prospective users. *SIGMIS Database*, 20(3):28–37, June 1989.

[28] M.L. Butterfield, J.S. Pearlman, and S.C. Vickroy. A System-of-Systems Engineering GEOSS: Architectural Approach. *Systems Journal, IEEE*, 2(3):321 –332, sept. 2008.

[29] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, March 2009.

[30] J.B. Caffall, D.S.; Michael. Formal methods in a system-of-systems development. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, pages 1856–1863. IEEE, October 2005.

[31] Martin Campbell-Kelly. Development and structure of the international software industry: 1950-1990. *Business and Economic History*, pages 73–110, Summer 1995.

[32] Pascal Cantot and Dominique Luzeaux. *Simulation and Modeling of Systems of Systems*. Wiley, 2011.

[33] Paul G. Carlock and Robert E. Fenton. System of Systems (SoS) Enterprise Systems Engineering for Information-Intensive Organizations. *Systems Engineering*, 4(4):242–261, 2001.

[34] Peter H. Carstensen and Kjeld Schmidt. Computer Supported Cooperative Work: New Challenges to Systems Design. In *In K. Itoh (Ed.), Handbook of Human Factors*, pages 619–636, 1999.

[35] Pin Chen and Jennie Clothier. Advancing Systems Engineering for systems-of-systems Challenges. *Systems Engineering*, 6(3):170–183, May 2003.

[36] S. Cheon, C. Seo, S. Park, and B. P. Zeigler. Design and implementation of distributed DEVS simulation in a peer to peer network system. In *Advanced Simulation Technologies Conference 2004*, April 2004.

[37] James R. Chiles. *Inviting Disaster: Lessons From the Edge of Technology*. Harper-Business, 2002.

[38] Edmund M. Clarke and Jeannette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[39] Dan Cocks. How Should We Use the Term "System of Systems" and Why Should We Care? In *Proceedings of the 16th INCOSE International Symposium 2006*. INCOSE, July 2006.

[40] Andrew L. Cohen, Debra Cash, and Michael J. Muller. Designing to support adversarial collaboration. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, page 9, New York, NY, USA, 2000. ACM.

[41] Joey W. Coleman, Anders Kaels Malmos, Peter Gorm Larsen, Jan Peleska, Ralph Hains, Zoe Andrews, Richard Payne, Simon Foster, Alvaro Miyazawa, Cristiano Bertolini, and André Didier. COMPASS Tool Vision for a System of Systems Collaborative Development Environment. In *Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012*, pages 451–456, July 2012.

[42] Joey W. Coleman, Anders Kaels Malmos, Claus Ballegaard Nielsen, and Peter Gorm Larsen. Evolution of the Overture Tool Platform. In *Proceedings of the 10th Overture Workshop 2012*, School of Computing Science, Newcastle University, 2012.

[43] Popular Mechanics Company. *Popular Mechanics*, volume 91, chapter The navy explores the air waves, pages 89–93. H.H. Windsor Jr., March 1949.

[44] S. Cook, E. Lawson, and J. Allison. Towards a Unified Systems Methodology for Australian Defence Systems-of-Systems. *Proceedings of the 9th International Symposium of the International Council on Systems Engineering*, June 1999.

[45] S. C. Cook. On the Acquisition of Systems of Systems. *Proceedings of the 2001 INCOSE International Symposium*, July 2001.

[46] D. Crockford. The application/json media type for javascript object notation (json). RFC 4627, Internet Engineering Task Force, July 2006.

[47] W.A. Crossley. System of Systems: An Introduction of Purdue University Schools of Engineering's Signature Area. In *Engineering Systems Symposium,*. MIT Engineering Systems Division, 2004.

[48] Judith Dahmann and Kristen Baldwin. Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering. In *IEEE Systems Conference*. IEEE, April 2008.

[49] Wayne J Davis. Distributed simulation and control: the foundations. In *Winter Simulation Conference 2001. Proceedings of the* , pages 187–198 vol.1, Dec 2001.

[50] G. de Vreede and R.O. Briggs. Collaboration Engineering: Designing Repeatable Processes for High-Value Collaborative Tasks. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, Jan 2005.

[51] Sidney Dekker. *Drift into Failure*. Ashgate, 2011.

[52] Daniel DeLaurentis. Role of humans in complexity of a system-of-systems. In *Digital Human Modeling*, volume 4561 of *Lecture Notes in Computer Science*, pages 363–371. Springer Berlin Heidelberg, 2007.

[53] Daniel A. DeLaurentis. A taxonomy-based Perspective for Systems of Systems Design Methods. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 86–91. IEEE, October 2005.

[54] Daniel A. DeLaurentis and W.A. Crossley. A taxonomy-based Perspective for Systems of Systems Design Methods. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 86–91. IEEE, October 2005.

[55] C.E. Dickerson and D. Mavris. A Brief History of Models and Model Based Systems Engineering and the Case for Relational Orientation. *Systems Journal, IEEE*, 7(4):581–592, 2013.

[56] M.J. DiMario. System of systems interoperability types and characteristics in joint command and control. In *System of Systems Engineering, 2006 IEEE/SMC International Conference on*. IEEE, April 2006.

[57] F. Dittmann. The first computer communication network between east and west. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, Sept. 2008.

[58] D. Drusinksy and M.-T. Shing. Creation and Evaluation of Formal Specifications for System-of-Systems Development. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1864 – 1869 Vol. 2, oct. 2005.

[59] *Eclipse Documentation*, 2008.

[60] R.Johnson E.Gamma, R.Helm and J.Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, 1995.

[61] H. Eisner, J. Marciniak, and R. McMillan. Computer-Aided System of Systems (S2) Engineering. In *Systems, Man, and Cybernetics, 1991. 'Decision Aiding for Complex Systems, Conference Proceedings., 1991 IEEE International Conference on*, pages 531 –537 vol.1, oct 1991.

[62] J. Ferguson. Crouching dragon, hidden software: software in DoD weapon systems. *Software, IEEE*, 18(4):105–107, 2001.

[63] David A. Fisher. An Emergent Perspective on Interoperation in Systems of Systems. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, March 2006. CMU/SEI-2006-TR-003.

[64] J. S. Fitzgerald and P. G. Larsen. Balancing Insight and Effort: the Industrial Uptake of Formal Methods. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems, Essays in Honour of Dines Bjørner and Chaochen Zhou on the Occasion of Their 70th Birthdays*, pages 237–254, Volume 4700, September 2007. Springer, Lecture Notes in Computer Science. ISBN 978-3-540-75220-2.

[65] J. S. Fitzgerald and P. G. Larsen. Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods. In T. Margaria, A. Philippou, and B. Steffen, editors, *Proc. 2nd Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2007)*, 2007. Also Technical Report CS-TR-999, School of Computing Science, Newcastle University.

[66] J. S. Fitzgerald, P. G. Larsen, S. Tjell, and M. Verhoef. Validation Support for Real-Time Embedded Systems in VDM++. Technical Report CS-TR-1017, School of Computing Science, Newcastle University, April 2007. Revised version in Proc. 10th IEEE High Assurance Systems Engineering Symposium, November, 2007, Dallas, Texas, IEEE.

[67] J. S. Fitzgerald, P. G. Larsen, and M. Verhoef. Vienna Development Method. *Wiley Encyclopedia of Computer Science and Engineering*, 2008. edited by Benjamin Wah, John Wiley & Sons, Inc.

[68] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edition, 2009. ISBN 0-521-62348-0.

[69] John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[70] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *ACM Sigplan Notices*, 43(2):3–11, February 2008.

[71] John Fitzgerald, Peter Gorm Larsen, and Jim Woodcock. Modelling and Analysis Technology for Systems of Systems Engineering: Research Challenges. In *INCOSE*, Rome, Italy, July 2012.

[72] John Fitzgerald, Peter Gorm Larsen, and Jim Woodcock. Foundations for Model-based Engineering of Systems of Systems. In M. Aiguier et al., editor, *Complex Systems Design and Management*, pages 1–19. Springer, January 2014.

[73] John S. Fitzgerald, Jeremy Bryans, and Richard Payne. A Formal Model-Based Approach to Engineering Systems-of-Systems. In Luis M. Camarinha-Matos, Lai Xu, and Hamideh Afsarmanesh, editors, *Collaborative Networks in the Internet of Services - 13th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2012, Bournemouth, UK, October 1-3, 2012. Proceedings*, volume 380 of *IFIP Advances in Information and Communication Technology*, pages 53–62. Springer, 2012.

[74] R.W. Floyd. Assigning Meanings to Programs. In *the Symposium on Applied Mathematics*, volume 19, pages 19–32. American Mathematical Society, 1967.

[75] Jay W. Forrester. System dynamics: the foundation under systems thinking. Sloan School of Management, Massachusetts Institute of Technology, Dec 2010.

[76] Charles. François. Systemics and cybernetics in a historical perspective. *Systems Research and Behavioral Science*, 16(3):203–219, Summer 1999.

[77] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering, 2007. FOSE '07*, pages 37–54, May 2007.

[78] Moti Frank. Engineering systems thinking and systems thinking. *Systems Engineering*, 3(3):163–168, 2000.

[79] Brigitte Fröhlich and Peter Gorm Larsen. Combining VDM-SL Specifications with C++ Code. In Marie-Claude Gaudel and Jim Woodcock, editors, *FME'96: Industrial Benefit and Advances in Formal Methods*, pages 179–194. Springer-Verlag, March 1996.

[80] R.M. Fujimoto. A Distributed simulation systems. In *Winter Simulation Conference 2003. Proceedings of the* , pages 124–134, Dec 2003.

[81] Guy Gallasch and Lars M. Kristensen. Comms/CPN: A communication infrastructure for external communication with design/CPN. In *3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01)*, pages 75–90. DAIMI PB-554, Aarhus University, aug 2001.

[82] M.T. Gamble and R.F. Gamble. Reasoning about Hybrid System of Systems Designs. In *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, pages 154 –163, feb 2008.

[83] E Gamma, R Helm, R Johnson, and J Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP' 93 - Object-Oriented Programming*, volume 707, pages 406–431, 1993.

[84] Ervan G. Garrison. *History of Engineering and Technology: Artful Methods*. CRC Press. Addison-Wesley Publishing Company, 2. edition, Jul 1998.

[85] N.D. Geddes, D.M. Smith, and C.S. Lizza. Fostering Collaboration in Systems of Systems. In *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, pages 950 – 954 vol.1. IEEE, oct 1998.

[86] Tayfun Gezgin, Christoph Etzien, Stefan Henkler, and Achim Rettberg. Towards a rigorous modeling formalism for systems of systems. In *15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 204–211. IEEE, April 2012.

[87] Harry H. Goode and Robert Engel Machol. *Systems Engineering: An Introduction to the Design of large-Scale Systems*. McGraw-Hill, New York, 1957.

[88] A. Gorod, B. Sauser, and J. Boardman. System-of-Systems Engineering Management: A Review of Modern History and a Path Forward. *Systems Journal, IEEE*, 2(4):484–499, December 2008.

[89] Jeff Gray, Ted Bapty, Eep Neema, and James Tuck. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, 44:87–93, 2001.

[90] C4ISR Interoperability Working Group. System of systems interoperability (sosi): Final report cmu/sei-2004-tr-004. Technical report, Department of Defense, 1998.

[91] J. Grudin. Computer-supported cooperative work: history and focus. *Computer*, 27(5):19–26, May 1994.

[92] J.O. Gutierrez-Garcia, F.F. Ramos-Corchado, and J.-L. Koning. Obligations as Constrainers, Descriptors, and Linkers of Open System of Systems. In *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pages 1 –6, june 2009.

[93] Anthony Hall. Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19, September 1990.

[94] Arthur David Hall. *A methodology for systems engineering*. Van Nostrand, 1962.

[95] D. Harel. Biting the silver bullet: toward a brighter future for system development. *Computer*, 25(1):8–20, Jan 1992.

[96] D. Harel. Systems engineering - A retrospective view. *Systems Engineering*, 1(4):258–266, 1998.

[97] I. T. Hawryszkiewycz. Providing Agent Support for Collaborative Systems Using a Domain Oriented Design Method. *Int. J. Agent-Oriented Softw. Eng.*, 1(2):175–192, July 2007.

[98] Igor T. Hawryszkiewycz. A Design Framework for Collaboration in Systems of Systems. In *Advancing Democracy, Government and Governance*, volume 7452 of *Lecture Notes in Computer Science*, pages 67–78. Springer Berlin Heidelberg, 2012.

[99] Naim A. heir. *Systems Modeling and Computer Simulation, Second Edition*. Marcel Dekker, 1995.

[100] J.D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering, 2007. FOSE '07*, pages 188–198, May 2007.

[101] J.D. Herbsleb, A. Mockus, T.A. Finholt, and R.E. Grinter. An empirical study of global software development: distance and speed. In *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 81–9, 2001.

[102] Martin Hirsch, Stefan Henkler, and Holger Giese. Modeling Collaborations with Dynamic Structural Adaptation in Mechatronic UML. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, pages 33–40. ACM, 2008.

[103] Derek K Hitchins. Systems methodology. In *Conference on Systems Engineering Research*, March 2005.

[104] H. Holmstrom, E.O. Conchuir, P.J. Agerfalk, and B. Fitzgerald. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. In *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 3–11, 2006.

[105] Eric C. Honour. Incose: History of the international council on systems engineering. *Systems Engineering*, 1(1):4–13, 1999.

[106] J. Hooman and M. Verhoef. Formal semantics of a VDM extension for distributed embedded systems. In D. Dams, U. Hannemann, and M. Steffen, editors, *Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever*, volume 5930 of *Lecture Notes in Computer Science*, pages 142–161. Springer-Verlag, 2010.

[107] Georges Ifrah. *The Universal History of Computing: From the Abacus to the Quantum Computer*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.

[108] INCOSE. Systems Engineering Handbook. A Guide for System Life Cycle Processes and Activities. Technical report, International Council on Systems Engineering, 7670 Opportunity Rd., Suite 220 San Diego, CA, January 2010.

[109] Claire Ingram, Richard Payne, Simon Perry, Jon Holt, Finn Overgaard Hansen, and Luís Diogo Couto. Modelling Patterns for Systems of Systems Architectures. In *IEEE SysCon 2014*, 2014.

[110] José Antonio Esparza Isasa, Peter W.V. Jørgensen, and Claus Ballegaard. Modelling Energy Consumption in Embedded Systems with VDM-RT. In *Proceedings of State Machines, Alloy, B, VDM, and Z 2014 (ABZ 2014)*, July 2014.

[111] Michael C. Jackson. *Systems Methodology for the Management Sciences*. Springer, 1991.

[112] Francçois Jacob. *The Logic of Living Systems: a History of Heredity*. Allen Lane, London, 1974.

[113] M. Jamshidi. System-of-Systems Engineering – a Definition. In *International Conference on Systems, Man, and Cybernetics*. IEEE, October 2005.

[114] M. Jamshidi. System of Systems Engineering – New Challenges for the 21st Century. *Aerospace and Electronic Systems Magazine, IEEE*, 23(5):4 –19, may 2008.

[115] G.M. Jenkins. The systems approach. *Journal of Systems Engienering*, I:3–49, 1996.

[116] Kurt Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. Technical Report Daimi PB 338, Aahus University, Ny Munkegade, Building 540, DK-8000 AArhus C, Denmark, November 1990.

[117] E. B. Johnsen, O. Owe, J. Bjørk, and M. Kyas. An Object-Oriented Component Model for Heterogeneous Nets. In *6th International Symposium on Formal Methods for Components and Objects FMCO'2007*, volume 5382 of *LNCS*, pages 257–279, Springer, Berlin Heidelberg, October 2008. Springer-Verlag.

[118] Einar Broch Johnsen and Olaf Owe. An Asynchronous Communication Model for Distributed Concurrent Objects. *Software and Systems Modeling*, 6(1):39–58, March 2007.

[119] L Johnson. A view from the 1960s: how the software industry began. *Annals of the History of Computing, IEEE*, 20(1):36–42, Jan 1998.

[120] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall International, Englewood Cliffs, New Jersey, second edition, 1990. ISBN 0-13-880733-7.

[121] Cliff B. Jones. Scientific Decisions which Characterize VDM. In J.M. Wing, J.C.P. Woodcock, and J. Davies, editors, *FM'99 - Formal Methods*, pages 28–47. Springer-Verlag, 1999. Lecture Notes in Computer Science 1708.

[122] N. Karcanias and A.G. Hessami. Complexity and the notion of system of systems: Part (I): defining the notion of system of systems. In *World Automation Congress (WAC), 2010*, pages 1–7, September 2010.

[123] N. Karcanias and A.G. Hessami. Complexity and the Notion of System of Systems: Part (II): Defining the Notion of System of Systems. In *World Automation Congress (WAC), 2010*, pages 1–7, September 2010.

[P124] Rick Kazman, Claus Ballegaard Nielsen, and Klaus Schmid. Understanding Patterns for System-of-Systems Integration. Technical Report CMU/SEI-2013-TR-017, Software Engineering Institute, Carnegie Mellon University, 2013.

[P125] Rick Kazman, Klaus Schmid, Claus Ballegaard Nielsen, and John Klein. Understanding patterns for system of systems integration. In *8th International Conference on System of Systems Engineering (SoSE), IEEE SoSE 2013*, pages 141–146, June 2013.

[126] C.B. Keating. Research Foundations for System of Systems Engineering. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, pages 2720 – 2725. IEEE, Oct 2005.

[127] Charles Keating, Ralph Rogers, Resit Unal, David Dryer, Andres Sousa-Poza, Robert Safford, William Peterson, and Ghaith Rabadi. System of System Engineering. *Engineering Management Journal*, 15(3):36–45, September 2003.

[128] Duncan Kemp, Rhianne Evans, Jon Elphick, and David Camm. Steampunk System of Systems Engineering : A case study of successful system of systems engineering in 19th century Britain. In *In Proceedings 23rd Annual International Symposium of the International Council on Systems Engineering*, 2013.

[129] N. Kilicay-Ergin and C. Dagli. Executable Modeling for System of Systems Architecting: An Artificial Life Framework. In *Systems Conference, 2008 2nd Annual IEEE*, pages 1 –5, april 2008.

[130] J.C. Knight, K.S. Hanks, and S.R. Travis. Tool support for production use of formal techniques. In *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pages 242–251, 2001.

[131] John Knight. Challenges in the utilization of formal methods. In Hans Rischel Anders P. Ravn, editor, *Formal Techniques in Real-time and Fault-tolerant Systems*, pages 1–17, LNCS 1486, 1998. Springer.

[132] John C. Knight, Colleen L. Dejong, Matthew S. Gibble, and Lus G. Nakano. Why are formal methods not used more widely. In *Fourth NASA Formal Methods Workshop*, pages 1–12, 1997.

[133] Vadim. Kotov. Systems-of-Systems as Communicating Structures. Technical Report HPL-97-124, Hewlett Packard Computer Systems Laboratory Paper, 1997.

[134] S. W. J. Kozlowski and B. S. Bell. *Handbook of psychology: Industrial and Organizational Psychology*, volume 12, chapter Work groups and teams in organizations, pages 333–375). Wiley-Blackwell., 2003.

[135] Steve W.J. Kozlowski and Daniel R. Ilgen. Enhancing the Effectiveness of Work Groups and Teams. *Psychological Science in the Public Interest*, 7(3):77–124, Dec. 2006.

[136] Jeff Kramer. Is Abstraction the Key to Computing? *Communications of the ACM*, 50(4):37–42, 2007.

[137] Jeff Kramer and Orit Hazzan, editors. *Procedings from 1st workshop on The Role of Abstraction in Software Engineering: Organizational, Managerial and Cognitive Perspectives*, May 2006.

[138] Anette J. Krygiel. *Behind the Wizard's Curtain, an Integration Environment for a System of Systems*. CCRP publication series, July 1999.

[139] Lukas Ladenberger, Jens Bendisposto, and Michael Leuschel. Visualising Event-B Models with B-Motion Studio. In *Proceedings of the 14th International Workshop on Formal Methods for Industrial Critical Systems*, pages 202–204. Springer-Verlag, November 2009.

[140] Caroline Twomey Lamb and Donna H. Rhodes. Systems thinking as an emergent team property: Ongoing research into enablers and barriers to team-level systems thinking. In *IEEE International Systems Conference*, 2008.

[141] Jo Ann Lane and Ricardo Valerdi. Synthesizing SoS Concepts for use in Cost Modeling. *Systems Engineering*, 10(4):297–308, December 2007.

[142] P. G. Larsen, B. S. Hansen, et al. Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language, December 1996. International Standard ISO/IEC 13817-1.

[143] Peter Gorm Larsen. Ten Years of Historical Development: "Bootstrapping" VDM-Tools. *Journal of Universal Computer Science*, 7(8):692–709, 2001.

[144] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6, January 2010.

[145] Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. In Min Zhang and Volker Stolz, editors, *Harnessing Theories for Tool Support in Software*, pages 9–19, November 2010.

[146] Peter Gorm Larsen and Poul Bøgh Lassen. An Executable Subset of Meta-IV with Loose Specification. In *VDM '91: Formal Software Development Methods*. VDM Europe, Springer-Verlag, March 1991.

[147] Peter Gorm Larsen and Kenneth Lausdahl. Overture/VDM Tools Status. Handout at SEFM20 Tool Workshop, September 2010. Second edition.

[148] Peter Gorm Larsen, Kenneth Lausdahl, and Nick Battle. The VDM-10 Language Manual. Technical Report TR-2010-06, The Overture Open Source Initiative, April 2010.

[149] Peter Gorm Larsen and Wiesław Pawłowski. The Formal Semantics of ISO VDM-SL. *Computer Standards and Interfaces*, 17(5–6):585–602, September 1995.

[150] Alexander Laszlo and Stanley Krippner. Systems theories: Their origins, foundations, and development. In J.S. Jordan, editor, *Systems Theories and A Priori Aspects of Perception*, chapter 3, pages 47–74. Elsevier Science, Amsterdam, The Netherlands, second edition, 1998.

[151] Kenneth Lausdahl. *Enhancing Formal Modelling Tool Support with Increased Automation*. PhD thesis, Aarhus University, June 2013.

[152] Kenneth Lausdahl. Translating VDM to Alloy. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods*, volume 7940 of *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin Heidelberg, 2013. 10th International Conference, IFM 2013.

[153] Kenneth Lausdahl, Hans Kristian Agerlund Lintrup, and Peter Gorm Larsen. Connecting UML and VDM++ with Open Tool Support. In Ana Cavalcanti and Dennis R. Dams, editors, *Proceedings of the 2nd World Congress on Formal Methods*, volume 5850 of *Lecture Notes in Computer Science*, pages 563–578, Berlin, Heidelberg, November 2009. Springer-Verlag. ISBN 978-3-642-05088-6.

[P154] Kenneth Lausdahl, Claus Ballegaard Nielsen, and Klaus Kristensen. Including Running System Implementations in the Simulation of System of Systems Models. In *Submitted to Software Engineering and Formal Methods (SEFM) 2014*, 2014.

[155] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. Programming Real-Time Applications with Signal. *Proceedings of the IEEE*, 79(9):1321–1336, Sept. 1991.

[156] Michael Lees, Brian Logan, and Georgios Theodoropoulos. Distributed Simulation of Agent-based Systems with HLA. *ACM Trans. Model. Comput. Simul.*, 17, July 2007.

[157] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, October 2009.

[158] G. Lewis, E. Morris, P. Place, S. Simanta, D. Smith, and L. Wrage. Engineering systems of systems. In *Systems Conference, 2008 2nd Annual IEEE*, pages 1–6. IEEE, April 2008.

[159] Shiyong Liu. Employing System of Systems Engineering in China's Emergency Management. *Systems Conference, 2010 4th Annual IEEE*, 5(2):541 –546, april 2011.

[160] R. Lock and I. Sommerville. Modelling and Analysis of Socio-Technical System of Systems. In *Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on*, pages 224 –232, march 2010.

[161] Carsten Lucke, Sascha Krell, and Ulrike Lechner. Critical Issues in Enterprise Architecting – A Literature Review. In *AMCIS 2010 Proceedings*, Aug 2010.

[162] Jochen Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2(1):5–14, 2003.

[163] Marie Ludwig, Nicolas Farcet, Jean-Philippe Babau, and Joël Champeau. Integrating Design and Runtime Variability Support into a System ADL. In *Proceedings of the 7th European conference on Modelling foundations and applications*, pages 270–281, Berlin, Heidelberg, 2011. Springer-Verlag.

[164] Stephen J. Lukasik. Systems, Systems of Systems, and the Education of Engineers. In *AI EDAM*, volume 12, pages 55–60. Cambridge University, 1998.

[165] Yue Ma, Jean-Pierre Talpin, Sandeep Kumar Shukla, and Thierry Gautier. Distributed Simulation of AADL Specifications in a Polychronous Model of Computation. In *Embedded Software and Systems, 2009. ICESS '09, International Conference on*, pages 604–614, May 2007.

[166] V. Mahulkar, S. McKay, D.E. Adams, and A.R. Chaturvedi. System-of-Systems Modeling and Simulation of a Ship Environment With Wireless and Intelligent Maintenance Technologies. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(6):1255 –1270, Nov 2009.

[167] Mark W. Maier. Architecting Principles for Systems-of-Systems. In *Sixth International Symposium of the International Council on Systems Engineering*. INCOSE, 1996.

[168] Mark W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1998.

[169] Mark W. Maier. Architecting principles for systems-of-systems. The Information Architects Cooperative (TIAC) whitepaper, www.infoed.com/Open/PAPERS/systems.htm, 1998.

[170] Mark W. Maier and Eberhardt Rechtin. *The Art of Systems Architecting*. CRC Press LLC, December 1997.

[171] M. Mane and D.A. DeLaurentis. Impact of programmatic system interdependencies on system-of-systems development. In *System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on*, pages 1–6. IEEE, june 2009.

[172] William H. J. Manthorpe. The Emerging Joint System of Systems: A Systems Engineering Challenge and Opportunity for APL. *John Hopkins APL Technical Digest*, 17(3):305–310, 1996.

[173] John Mcleod. Ten years of computer simulation. *Electronic Computers, IRE Transactions on*, EC-11(1):2–6, Feb 1962.

[174] W.K. McQuay. Distributed collaborative environments for systems engineering. In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, number 2, Oct. 2004.

[175] W.K. McQuay. Systems engineering of computer based systems: status and future perspectives. In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, volume 2, pages 9.D.3–91–10 Vol.2, Oct 2004.

[176] Donella H. Meadows. *Thinking in Systems: A Primer*. Chelsea Green Publishing, ISBN: 978-1603580557, Dec. 2008.

[177] A. Meilich. Human systems integration - a system of systems engineenng challenge. In *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, pages 1–6. IEEE, April 2007.

[178] Abe Meilich. System of Systems (SoS) Engineering & Architecture Challenges in a Net Centric Environment. In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Los Angeles, CA, April 2006. IEEE.

[179] Jayadev Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1):39–65, March 1986.

[180] David Holst Møller and Christian Rane Paysen Thillermann. Using Eclipse for Exploring an Integration Architecture for VDM. Master's thesis, Aarhus University/Engineering College of Aarhus, June 2009.

[181] Edwin Morris, Linda Levine, Craig Meyers, Pat Place, and Dan Plakosh. System of systems interoperability (sosi): Final report cmu/sei-2004-tr-004. Technical report, CMU SEI, April 2004.

[182] P. Naur and (Ed.) B. Randell. Software Engineering: Report on a Conference sponsored by the NATO Science Committee. Garmisch, Germany, 7th to 11th October 1968, Brussels, Scientific Affairs Division, NATO, January 1969.

[183] Claus Ballegaard Nielsen. Towards Dynamic Reconfiguration of Distributed Systems in VDM-RT. In *Semantic Issues in VDM: a BCS-FACS and Overture Workshop*, September 2010.

[184] Claus Ballegaard Nielsen. Modelling Dynamic Topologies via Extensions of VDM-RT : A Case Study of an Evolving System. Technical Report ECE-TR-9, Aarhus University, School of Engineering, July 2012.

[185] Claus Ballegaard Nielsen. Utilizing VDM Models in Process Management Tool Development: an Industrial Case. In *Proceedings of the 9th Overture Workshop 2011*, number ECE-TR-2 in ECE Technical Report, Electrical and Computer Engineering, Aarhus University. Aarhus University, 2012.

[P186] Claus Ballegaard Nielsen, Claire Ingram, André Didier, Uwe Schulze, Stefan Hallerstede, Andrew Galloway, and Peter Gorm Larsen. Challenges in Collaborative Formal Modelling of System of Systems. In *Draft paper for submission to the International Journal of System of Systems Engineering*, 2014.

[P187] Claus Ballegaard Nielsen and Peter Gorm Larsen. Extending VDM-RT to Enable the Formal Modelling of System of Systems. In *Proceedings of the 7th International Conference on System of System Engineering, IEEE SoSE 2012*, IEEE Systems Journal. IEEE, July 2012.

[P188] Claus Ballegaard Nielsen and Peter Gorm Larsen. Collaborative Formal Modeling of System of Systems. In *IEEE SysCon 2014*, 2014.

[P189] Claus Ballegaard Nielsen, Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, and Jan Peleska. Model-based Engineering of Systems of Systems. *Submitted to ACM Computing Surveys*, June 2013.

[P190] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. Combining VDM with Executable Code. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z*, volume 7316 of *Lecture Notes in Computer Science*, pages 266–279, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-30884-0.

[P191] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. Using the Overture Tool as a More General Platform. In Franco Mazzanti, editor, *iFM 2012 &*

ABZ 2012 - Proceedings of the Posters & Tool demos Session, pages 1–34. CNR-ISTI, June 2012.

[P192] Claus Ballegaard Nielsen, Kenneth Lausdahl, and Peter Gorm Larsen. Distributed Simulation of Formal Models in System of Systems Engineering. In *4th IEEE track on Collaborative Modelling and Simulation in IEEE WETICE 2014*, June 2014.

[193] Jacob Porsborg Nielsen and Jens Kielsgaard Hansen. Development of an Overture/VDM++ Tool Set for Eclipse. Master's thesis, Technical University of Denmark, Informatics and Mathematical Modelling, August 2005. IMM-THESIS-2005-58.

[194] A.R. Nielson and C. Koepping. Integrating information systems into the net-centric environment. In *Autotestcon, 2006 IEEE*, pages 403–409. IEEE, Sept 2006.

[195] E.G. Nilsson, E.K. Nordhagen, and G. Oftedal. Aspects of systems integration. In *Systems Integration, 1990. Systems Integration '90., Proceedings of the First International Conference on*. IEEE, 1990.

[196] Eli M. Noam. Beyond Liberalization : From the Network of Networks to the System of Systems. *Telecommunications Policy*, 18(4):286–294, May 1994.

[197] William D Nordhaus. The Progress of Computing. *Cowles Foundation*, (Paper No. 1324), September 2001.

[198] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. A UTP semantics for Circus. *Formal Aspectss of Computing*, 21:3–32, February 2009.

[199] M.W. Oliver. Systems engineering and software engineering, contrasts and synergism. In *Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop o*, pages 125–132, 1995.

[200] *The Common Object Request Broker: Architecture and Specification*. OMG, July 1996.

[201] Harold Ossher, William Harrison, and Peri Tarr. Software engineering tools and environments: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 261–277, 2000.

[202] OUSD(AT&L), DoD. Systems and Software Engineering. Systems Engineering Guide for Systems of Systems. Technical Report Version 1.0., Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Department of Defense, Washington DC, August 2008.

[203] D. L. Parnas. Really rethinking formal methods. *Software*, 43(1):28–34, January 2010.

[204] R. S. Pei. Systems of Systems Integration (SoSI)-A Smart Way of Acquiring Army C4I2WS Systems. In *Summer Computer Simulation Conference*, 2000.

[205] Martin Petzold, Oliver Ullrich, and Ewald Speckenmeyer. Dynamic Distributed Simulation of DEVS Models on the OSGi Service Platform. In *Proceedings of ASIM 2011 - 21st Symposium Simulation Technique*. Hrsg. Pabst Science Publ, 2011.

[206] Nico Plat and Peter Gorm Larsen. An Overview of the ISO/VDM-SL Standard. *Sigplan Notices*, 27(8):76–82, August 1992.

[207] J. Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[208] Guo qing Ye, Peng Wang, and Meng jun Li. Research on distributed collaborative design environment supporting complex system-of-systems evolution. In *Industrial Engineering and Engineering Management (IE EM), 2011 IEEE 18Th International Conference on*, pages 453–457. IEEE, Sept 2011.

[209] C.V. Ramamoorthy, C. Chandra, H.G. Kim, Y. C Shim, and V. Vij. Systems integration: problems and approaches. In *Systems Integration, 1992. ICSI '92., Proceedings of the Second International Conference on*. IEEE, 1992.

[210] W. Reinhard, J. Schweitzer, G. Volksen, and M. Weber. CSCW tools: concepts and architectures. *IEEE Computer*, 27(5):28–36, 1994.

[211] J. Ring and A.M. Madni. Key challenges and opportunities in 'system of systems' engineering. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, pages 973–978, oct. 2005.

[212] C. L. Roe. A systems engineering process for systems of systems. In *Proceedings of the 9th Annual International Symposium of the INCOSE*, pages 27–33. INCOSE, June 1999.

[213] A. W. Roscoe. *Understanding Concurrent Systems*. Springer, London Dordrecht Heidelberg New York, 2010.

[214] Saul Rosen. Electronic computers: a historical survey. *ACM Computing Surveys*, 1, August 1996.

[215] Andrew P. Sage and Christopher D. Cuppan. On the Systems Engineering and Management of Systems of Systems and Federations of Systems. *Inf. Knowl. Syst. Manag.*, 2(4):325–345, December 2001.

[216] A.P. Sage. Systems engineering of computer based systems: status and future perspectives. In *Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop of*, pages 5–15, 1995.

[217] F. Sahin, M. Jamshidi, and P. Sridhar. A Discrete Event XML based Simulation Framework for System of Systems Architectures. In *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, April 2007.

[218] David Till Sara Jones and Ann Wrightson. Formal methods and requirements engineering: Challenges and synergies. *Journal Systems Software*, 40:263–273, 1998.

[219] P.M. Sargent. Software models and engineering practice. *Computer-Aided Engineering Journal*, 5(6):237–240, Dec 1988.

[220] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[221] Thomas Saunders, Charles Croom, Wanda Austin, John Brock, Natalie Crawford, Mica Endsley, Ed Glasgow, Dan Hastings, Alex Levis, and Richard Murray. System-of-systems engineering for air force capability development: Executive summary and annotated brief. Report SAB-TR-05-04, Scientific advisory board, US Air Force, Jul 2005.

[222] Daniel Schneider and Mario Trapp. Runtime Safety Models in Open Systems of Systems. *Dependable, Autonomic and Secure Computing, IEEE International Symposium on*, pages 455–460, 2009.

[223] C.B. Seaman. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4):557–572, Jul 1999.

[224] Scott A. Selberg and Mark A. Austin. Toward an Evolutionary System of Systems Architecture. Technical report, Institute for Systems Research., INCOSE, 2008.

[225] Abeer Sharawi, Serge N. Sala-Diakanda, Sergio Quijada, Nabeel Yousef, Luis Rabelo, and Jose Sepulveda. A Distributed Simulation Approach for Modeling and Analyzing System of Systems. In *Proceedings of the 2006 Winter Simulation Conference*, 2006.

[226] Aaron J. Shenhar. A New Systems Engineering Taxonomy. In *Proceedings of the 4th Annual International Symposium of The National Council on Systems Engineering*, volume 2, pages 261–276, August 1994.

[227] Aaron J. Shenhar and Zeev Bonen. The New Taxonomy of Systems: Toward an Adaptive Systems Engineering Framework. *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, 27(2):137–145, March 1997.

[228] A.A. Song and D.L. Kleinman. A distributed simulation system for team decisionmaking. In *AI, Simulation, and Planning in High Autonomy Systems, 1994. Distributed Interactive Simulation Environments., Proceedings of the Fifth Annual Conference on*, pages 129–135, Dec 1994.

[229] Rasmus Ask Sørensen and Jasper Moltke Nygaard. Evaluating Distributed Architectures using VDM++ Real-Time Modelling with a Proof of Concept Implementation. Master's thesis, Enginering College of Aarhus, December 2007.

[230] Richard Stevens, Peter Brook, Ken Jackson, and Stuart Arnold. *Systems Engineering: Coping with Complexity*. Prentice Hall, 1998.

[231] S.H.; Dongsun Park Sunwoo Park; Hunt, C.A.; Kim. DEVS Peer-to-Peer Protocol for Distributed and Parallel Simulation of Hierarchical and Decomposable DEVS Models. In *Information Technology Convergence, 2007. ISITC 2007. International Symposium on*. IEEE, 2007.

[232] OMG Systems Modeling Language (OMG SysML$^{\mathrm{TM}}$). Technical Report Version 1.2, SysML Modelling team, June 2010. http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

[233] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[234] The VDM Tool Group. VDM Toolbox API. Technical report, CSK Systems, January 2008.

[235] Jan Tretmans, Klaas Wijbrans, and Michel Chaudron. Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods. *Form. Methods Syst. Des.*, 19(2):195–215, 2001.

[236] United States, Congress, Senate, Committee on Armed Services. *Restructuring of the Strategic Defense Initiative (SDI) Program: Joint hearing before the Committee on Armed Services, United States Senate and the Committe*. University of Michigan Library, October 6 1988.

[237] Ricardo Valerdi, Elliot Axelband, Barry Boehm Thomas Baehren, Dave Dorenbos, Scott Jackson, Azad Madni, Gerald Nadler, Paul Robitaille, and Stan Settles. A Research Agenda for Systems of Systems Architecting. *Int. J. System of Systems Engineering*, 1(1/2):171–188, 2008.

[238] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific Languages: An Annotated Bibliography. *SIGPLAN Not*, 35(6):26–36, June 2000.

[239] J.J. van Wijk. The value of visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 79–86, Oct. 2005.

[240] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[241] Marcel Verhoef. On the Use of VDM++ for Specifying Real-Time Systems. *Proc. First Overture workshop*, November 2005.

[242] Marcel Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. PhD thesis, Radboud University Nijmegen, 2009. The right one is [243].

[243] Marcel Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. PhD thesis, Radboud University Nijmegen, 2009.

[244] Marcel Verhoef, Peter Gorm Larsen, and Jozef Hooman. Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, Lecture Notes in Computer Science 4085, pages 147–162. Springer-Verlag, 2006.

[245] Alan Wassyng and Mark Lawford. Lessons learned from a successful implementation of formal methods in an industrial project. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153. Springer Berlin Heidelberg, 2003.

[246] Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, 265(3):10, September 1991.

[247] Michael Westergaard and Lars Kristensen. The access/cpn framework: A tool for interacting with the cpn tools simulator. In Giuliana Franceschinis and Karsten Wolf, editors, *Proceedings of the 30th International Conference on Applications and Theory of Petri Nets*, pages 313–322. Springer Berlin / Heidelberg, 2009.

[248] Jim Whitehead. Collaboration in Software Engineering: A Roadmap. In *2007 Future of Software Engineering*, FOSE '07, 2007.

[249] Jeannette M. Wing. Computational Thinking. *Commun. ACM*, 49(3):33–35, March 2006.

[250] Larry D. Wittie. Computer networks and distributed systems. *Computer*, 24(9), September 1991.

[251] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. Features of CML: a Formal Modelling Language for Systems of Systems. In *Proceedings of the 7th International Conference on System of System Engineering*. IEEE, July 2012.

[252] Jim Woodcock. Engineering UToPiA - Formal Semantics for CML. In *FM2014*, 2014.

[253] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal Methods: Practice and Experience. *ACM Computing Surveys*, 41(4):1–36, October 2009.

[254] A. Wayne Wymore. *A mathematical theory of systems engineering*. John Wiley & Sons Inc, 1967.

[255] A. Wayne Wymore. *Systems Engineering Methodology for Interdisciplinary Teams*. John Wiley & Sons Inc, 1976.

[256] A. Wayne Wymore. *Model-Based Systems Engineering*. CRC Press, 1993.

[257] P. Zave. Feature Interactions and Formal Specifications in Telecommunications. *Computer*, 26(8):20–28, 1993.

[258] Bernard P. Zeigler. *Theory of Modelling and Simulation*. Krieger Publishing Co., Inc., Melbourne, FL, USA, 1984.

[259] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2000.

[260] Ming Zhang, Bernard P. Zeigler, and Phillip Hammonds. DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies. *ITEA Journal*, 2005.