A Modelling Framework for Systems-of-Systems with Real-Time and Reliability Requirements

DISSERTATION

zur Erlangung des Grades eines Doktors

der Naturwissenschaften/Ingenieurwissenschaften/Pädagogik

vorgelegt von

Ing.-Imad Sanduka

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät

der Universität Siegen

Siegen 2015

Stand: Juli 2015

UNIVERSITÄT SIEGEN
Institut für Embedded Systems

PhD Thesis

A Modelling Framework for Systems-of-Systems with Real-Time and Reliability Requirements

Submitted by:

Imad Sanduka

Supervised by:

Prof. Dr. Roman Obermaisser

Examination committee: Prof. Dr. Roland Wismüller
Prof. Dr. Marcin Grzegorzek
Prof. Dr. Madjid Fathi
Prof. Dr. Roman Obermaisser

Examination Date : 01.10.2015 Date of delivery: 03.07.2015

This Thesis is dedicated to my parents

and my fiancé

Acknowledgement

I would like to thank my supervisor Prof. Obermaisser, for his guidance and support throughout this study. Also, I would like to thank Airbus Group Innovation for supporting my thesis and studies. Special thanks to my technical supervisor at Airbus Innovations Mr.Tim Lochow and the Systems Engineering and Platforms team.

I would like to thank my parents and my family for their endless love and support through my life, and for my lovely fiancé for her unconditional love, support and understanding.

To all my friends thank you for your motivation and encouragement.

Table of Contents

Table of Contents V
Table of Figures VII
List of Tables IX
Abstract 1
Kurzfassung 3
1 Introduction 5
1.1 Overview 5
1.2 Thesis Contents 5
1.3 SoS Examples 6
1.3.1 SoS in Defence Applications 6
1.3.2 SoS in Emergency Response 7
1.3.3 SoS in Transportation Applications 8
1.3.4 SoS in Space 8
1.3.5 SoS in Health 9
1.4 Industrial Needs 10
1.5 Scientific Contribution and Research Problem 10
2 Concepts and Terms 13
2.1 Monolithic Systems 13
2.1.1 Definition 13
2.1.2 Characteristics 14
2.2 System of Systems 15
2.2.1 Definition of SoS 15
2.2.2 Characteristics 16
2.2.3 Classification 18
2.2.4 Classification of Constituent Systems 19
2.3 Systems of Systems vs. Monolithic Systems 19
2.4 Real-Time Systems 21
2.4.1 Definition 22
2.4.2 Classification 23
2.4.3 Temporal Parameters 24
2.4.4 Requirements of Real-Time Systems 25
2.4.5 Examples of Real-Time Systems 25
2.5 Dependability 28
2.5.1 Threats 28
2.5.2 Reliability Definition 31
2.5.3 Reliability Engineering for SoS 33
2.5.4 SoS Dependability 35
2.6 Model-Based System Engineering 35
2.6.1 Models of Systems 36
2.6.2 Principles of modelling 36
2.6.3 Modelling Viewpoints 37
3 State of the Art: System of Systems Engineering Practices 39
3.1 SoS Engineering 39
3.1.1 System Engineering 39
3.1.2 SoS Engineering Definition 39

3.2 Activities in SoS Engineering....40
3.2.1 System of Systems Engineering vs. System Engineering....42
3.3 Modelling Frameworks....44
3.3.1 DoDAF....45
3.3.2 MODAF....54
3.3.3 NAF....55
3.3.4 TOGAF....56
3.3.5 Other Architecture Frameworks....57
3.3.6 Summary....59
3.4 Modelling Languages....61
3.4.1 UML....61
3.4.2 SysML....63
3.4.3 SoaML....65
3.4.4 UPDM....66
3.4.5 Other Modelling languages....68
3.4.6 Summary....68
3.5 Research Gap in the State of the Art....70
4 Model-based System of System Engineering....73
4.1 Introduction....73
4.2 Architecture Patterns....73
4.2.1 Definition....73
4.2.2 Architecture Patterns in SoS....74
4.2.3 Mining Architecture Patterns....75
4.2.4 Using Architecture Patterns in SoS....76
4.3 Architecture Optimisation....78
4.3.1 Using Architecture Patterns in Architecture Optimisation....80
4.3.2 Modelling Process....81
4.4 Modelling Methodology....85
5 Timing Analysis and Optimisation....95
5.1 Introduction....95
5.2 SoS Timing Analysis and Model Definition....95
5.3 Integrating Timing Analysis in the Architecture Methodology....97
5.4 Optimisation....100
6 Reliability Analysis and Optimisation....101
6.1 Introduction....101
6.2 Reliability Block Diagram in DoDAF Architecture Framework....101
6.3 Reliability Formulation in Architecture Models....102
6.4 Reliability Optimisation and Architecture Patterns....103
6.5 Integrating Reliability Analysis in the Architecture Methodology....104
7 Examples and Results....107
7.1.1 Antenna Allocation of Emergency Response Communication System....107
7.1.2 Real-Time Analysis....115
7.1.3 Reliability Example....121
8 Discussion and Conclusion....129
9 References....131
10 List of Publications....137
11 Declaration / Eidesstattliche Erkärung....139

Table of Figures

Figure 1-1 Emergency response SoS operational concept [7] 7
Figure 1-2 Search and rescue high level operational scenario [9] 9
Figure 2-1 System definition 13
Figure 2-2 System structure 14
Figure 2-3 System of Systems (SoS) 15
Figure 2-4 System with inputs and outputs 21
Figure 2-5 Real-time system example 22
Figure 2-6 Temporal parameters 24
Figure 2-7 Fluid control system 26
Figure 2-8 Measurements of aircraft navigation system 27
Figure 2-9 Aircraft navigation systems 27
Figure 2-10 Dependability tree 28
Figure 2-11 Fault, error and failure concept 28
Figure 2-12 Fault classification 29
Figure 2-13 Failure classifications 30
Figure 2-14 Procedure for defining SoS reliability diagram 34
Figure 2-15 System model example 36
Figure 3-1 SoS engineering core elements [23] 42
Figure 3-2 SoS engineering activities and constituent-system engineering 42
Figure 3-3 Architecture frameworke concept 45
Figure 3-4 DoDAF viewpoints [59] 46
Figure 3-5 DoDAF operational view of emergency response SoS 48
Figure 3-6 Operational activity model of a house fire scenario 49
Figure 3-7 System view (SV-1) for the emergency response SoS 52
Figure 3-8 System functionality view of the emergency response SoS 53
Figure 3-9 MODAF architecture viewpoints 54
Figure 3-10 NAF architecture viewpoints 55
Figure 3-11 TOGAF architecture development method [61] 57
Figure 3-12 Zachman architecture framework 58
Figure 3-13 Core of the architecture description according to IEEE 1471 [57] 59
Figure 3-14 SysML profile construction 64
Figure 3-15 UPDM compliance levels 66
Figure 3-16 UPDM level 0 and level 1 [4] 67
Figure 3-17 UPDM viewpoints 68
Figure 3-18 Industrial example of current architecture frameworks in SoS engineering 71
Figure 4-1 Architecture pattern mining process [80] 75
Figure 4-2 Architecture patterns at different stages of the SoS development process 76
Figure 4-3 Using architecture patterns in SoS architecture development [81] 77
Figure 4-4 Dynamic interaction of architecture patterns over the SoS life cycle [81] 78
Figure 4-5 DSE methodology process flow 80
Figure 4-6 Using architecture patterns in the architecture optimisation process flow 80
Figure 4-7 Comparing and organizing architecture patterns using architecture optimisation . 81
Figure 4-8 Architecture optimisation process 82
Figure 4-9 Concise plugin 83
Figure 4-10 Constituent-system catalogue database generated using concise plugin 85
Figure 4-11 General flow of architecture modelling methodology 86

Figure 4-12 Architecture modelling methodology in DoDAF views 87
Figure 4-13 Operational activity diagram for police operational scenario 88
Figure 4-14 Operational activity to functions mapping 89
Figure 4-15 Mapping of functions to constituent-system classes 89
Figure 4-16 Using concise modelling packages in DoDAF 90
Figure 4-17 Package for constituent systems and functions in UPDM 90
Figure 4-18 Modelling optimisation objectives and constraints 92
Figure 4-19 Adding locations of constituent systems to the optimisation database 92
Figure 4-20 Adding the FMU URI to the constituent-system block 93
Figure 4-21 Generic architecture methodology flow 94
Figure 5-1 Timing properties attached to the system function 95
Figure 5-2 Functional graphs 96
Figure 5-3 UPDM class for constituent systems 97
Figure 5-4 UPDM class for SoS 97
Figure 5-5 Integrating timing analysis in the architecture methodology 98
Figure 5-6 Deriving timing requirements from operational scenarios 99
Figure 5-7 Functional graph identification 99
Figure 6-1 SoS reliability diagram procedure mapped to DoDAF views 102
Figure 6-2 Parallel-series reliability structure 102
Figure 6-3 Processing steps for reliability optimisation and architecture patterns 104
Figure 6-4 Integrating reliability analysis in the architecture methodology 105
Figure 7-1 Communication coverage optimisation use case 107
Figure 7-2 Operational activity flow diagram for antenna allocation example 108
Figure 7-3 Functions to operational activities mapping for antenna allocation example 109
Figure 7-4 Functional flow diagram for antenna allocation example 109
Figure 7-5 Architecture pattern for antenna allocation example 110
Figure 7-6 Mapping of geographical locations to constituent-system classes 111
Figure 7-7 Map of Berlin divided into grid points 111
Figure 7-8 Adding the optimisation goals to the model for the antenna coverage example 114
Figure 7-9 SoS antenna coverage solution 114
Figure 7-10 Matalb results for LTE coverage solution 115
Figure 7-11 Matlab results for Tetra coverage solution 115
Figure 7-12 Timing analysis example model 116
Figure 7-13 Simplified functional flow 116
Figure 7-14 Operational activity flow diagram for the timing example 117
Figure 7-15 Operational activity to function mapping for timing example 117
Figure 7-16 Timing example functional flow diagram 118
Figure 7-17 Functional graphs of timing example 119
Figure 7-18 Back-annotated model for timing example 120
Figure 7-19 Matlab results for timing example 121
Figure 7-20 Architecture model of the reliability example 121
Figure 7-21 Operational activities flow diagram of the reliability example 122
Figure 7-22 Operational activity to functional mapping of the reliability example 123
Figure 7-23 Functional flow diagram of the reliability example 124
Figure 7-24 Reliability block diagram of the reliability example 125
Figure 7-25 Matlab results for reliability with logarithmic values 126
Figure 7-26 Matlab results for reliability with linear scaling 127

List of Tables

Table 1 Fundamental system concepts in monolithic systems and SoS [24] [25] 20
Table 2 Differentiation between monolithic systems and SoS 21
Table 3 Differences of enterprises and products in TSE and SoS 43
Table 4 Analysis of main drivers in system engineering and SoS engineering [56] 44
Table 5 UML diagrams 63
Table 6 SysML diagrams 65
Table 7 SysML and concise stereotypes used in architecture optimisation 84
Table 8 UPDM stereotypes used for architecture optimisation 91
Table 9 Adding reliability values to the constituent-system catalogue 106
Table 10 Adding area grid points to the model 112
Table 11 Example of adding the antenna coverage data to the model 113
Table 12 Redundancy values in reliability optimisation 125

Abstract

The design and development of large and complex systems, such as System of Systems (SoS) with autonomous constituent systems is gaining increasing importance in many application domains (e.g., medical, aerospace, military and transportation). The wide scale and huge number of interactions involved in SoS makes it difficult to model and analyse the SoS architecture through state-of-the-art techniques for modelling, architecture development and analysis of monolithic systems. SoS are large-scale concurrent and distributed systems that are comprised of autonomous constituent systems with operational and managerial independence. The characteristics of SoS result in various challenges during the system design. It is necessary to consider emergent behaviour, an evolutionary development process, and an increased state space. In particular, these challenges are unsolved when moving form directed SoS, where the system has central control, towards virtual SoS with no central management and no common purpose ascribed to the constituent systems.

In addition to meeting functional requirements, a major challenge is the design space exploration and optimisation of non-functional properties among heterogeneous constituent systems such as reliability, timeliness, safety and security. Many SoS are real-time systems where timing requirements are central to the development process. The failure in meeting these requirements in the late stages of the SoS development can be avoided by introducing them already in the architecture development. Likewise, many SoS represent critical infrastructures that provide safety-relevant services to the users and the environment. Therefore, the specified reliability of the SoS must be guaranteed despite failures of individual constituent systems using fault-tolerance mechanisms.

The state-of-the-art modelling approaches describe the SoS using architecture frameworks such as the US Department of Defence Architecture Framework (DoDAF), British Ministry of Defence Architecture Framework (MODAF) and NATO Architecture Framework (NAF). However, these frameworks do not offer a precise connection between the different views of the SoS, which is needed to enable design exploration technologies, reusability and design automation using integrated tool chains. In addition, significant non-functional system properties (e.g., real time, reliability) are not addressed in the state-of-the-art modelling frameworks.

This thesis addresses the research gap by establishing a modelling framework that extends the current SoS frameworks to link the different views, and to satisfy real-time and reliability requirements. We provide a methodology that uses design exploration techniques and tools for SoS architecture optimization. The use of architecture patterns is introduced to enhance the model re-usability and facilitate the model evolution.

Scientific contributions beyond the state-of-the-art include a Model Based System Engineering (MBSE) methodology for SoS architecting based on the Unified Profile for DoDAF and MODAF (UPDM) with extensions to support significant SoS properties such as timing requirements and reliability. The methodology supports an architecture optimisation process based on Mixed Integer Linear Programming (MILP) connected to architecture patterns that enhance the model reusability. The developed modelling language and design methods are evaluated using simulations and example scenarios.

Kurzfassung

Systems-of-Systems (SoS) stellen große, komplexe Systeme dar, die in zahlreichen Anwendungsgebieten zunehmend an Bedeutung gewinnen (z.B. Medizin, Raumfahrt, Militär, Transportsysteme). Der große Umfang sowie die hohe Anzahl an Interaktionen in SoS erschweren deren Modellierung und die Analyse mit Verfahren des Stands der Technik aus monolithischen Systemen. SoS sind großangelegte, parallele verteilte Systeme, deren autonome Teilsysteme operativ und administrativ unabhängig sind. Diese Besonderheit führt zu vielfältigen Herausforderungen beim Entwurf dieser Systeme. Emergentes Verhalten, ein evolutionärer Entwicklungsprozess sowie ein vergrößerter Zustandsraum müssen berücksichtigt werden. Diese Herausforderungen sind insbesondere in virtuellen SoS ungelöst, bei denen für die Teilsysteme kein zentrales Management und kein gemeinsames Ziel vorgesehen ist.

Zusätzlich zur Erfüllung funktionaler Anforderungen besteht eine große Herausforderung bei der Optimierung des Entwurfsraums bezüglich der nichtfunktionalen Eigenschaften der heterogenen Teilsysteme (z.B. Zuverlässigkeit, Echtzeitfähigkeit und Sicherheit). Viele SoS sind Echtzeitsysteme, bei denen die zeitlichen Anforderungen ein zentraler Bestandteil des Entwicklungsprozesses sind. Die kostspielige Nichterfüllung dieser Anforderungen in späten Phasen der Entwicklung kann durch die frühzeitige Einbeziehung der Echtzeitanforderungen in die Architekturentwicklung vermieden werden. Häufig sind SoS auch kritische Infrastrukturen für sicherheitsrelevante Dienste. Daher muss die spezifizierte Zuverlässigkeit des SoS unabhängig von der Zuverlässigkeit der einzelnen Teilsysteme durch Fehlertoleranzmechanismen gewährleistet werden.

Bei den Modellierungsansätzen im Stand der Technik werden SoS durch Architekturframeworks wie das US Department of Defence Architecture Framework (DoDAF), das British Ministry of Defence Architecture Framework (MODAF) und das NATO Architecture Framework (NAF) beschrieben. Diese Frameworks bieten jedoch keine präzise Verbindungen zwischen verschiedenen Ansichten eines SoS, welche notwendig wären um Techniken für Design-Exploration, Wiederverwendbarkeit und Designautomatisierung durch integrierte Werkzeugketten zu ermöglichen. Des Weiteren werden wesentliche nichtfunktionale Systemeigenschaften wie Echtzeitfähigkeit und Zuverlässigkeit nicht adressiert.

Diese Doktorarbeit adressiert jene Forschungslücke durch die Entwicklung eines Modellierungsframeworks, welches die derzeitigen SoS-Frameworks erweitert. Das Framework stellt eine Verbindung zwischen verschiedenen Ansichten her und unterstützt Echtzeit- und Zuverlässigkeitsanforderungen. Zudem werden Techniken für Design-Exploration und Werkzeuge für die Optimierung von SoS-Architekturen eingeführt. Architekturmuster erleichtern die Wiederverwendbarkeit der Modelle und verbessern die Evolution der Modelle.

Die wissenschaftlichen Beiträge über den Stand der Technik hinaus beinhalten modellbasierte Systems-Engineering Methoden für die Erstellung von SoS-Architekturen basierend auf dem Unified Profile für DoDAF und MODAF (UPDM) mit Erweiterungen zur Unterstützung von Echtzeitfähigkeit und Zuverlässigkeit. Die Methodik unterstützt einen Architekturoptimierungsprozess basierend auf Mixed Integer Linear Programming (MILP) in Verbindung mit Architekturmustern. Die entwickelte Modellierungssprache und die Designmethoden wurden mithilfe von Simulationen und Beispielsszenarien evaluiert.

1 Introduction

1.1 Overview

The system development process becomes more complex at the level of large-scale systems such as System of Systems (SoS). A SoS is a large-scale concurrent and distributed systems that is comprised of autonomous constituent systems with operational and managerial independence. The characteristics of SoS result in challenges during the system design where emergent behaviour, an evolutionary development process, and an increased state space need to be considered. In particular, these challenges are unsolved when moving form directed SoS, where the system has central control, towards virtual SoS with no central management and no common purpose of the constituent systems.

Due to the complexity of SoS, it is infeasible to model all system aspects in a single model. Past research has focused on managing the design complexity by expressing the SoS within modelling frameworks. Such frameworks reduce SoS modelling complexity by using multiple layers called architecture views that depict the system from different perspectives (e.g. operational, functional, system). US Department of Defence Architecture Framework (DoDAF) [1], British Ministry of Defence Architecture Framework (MODAF) [2] and NATO Architecture Framework (NAF) [3] are the mainly used architecture frameworks for modelling SoS. To support these frameworks the Object Management Group (OMG) [4] developed a new modelling language, namely the Unified Profile for DoDAF and MODAF (UPDM) [4]. This profile is based on UML and SysML with profile extensions containing specialised stereotypes that support architecture-framework elements and parts. However, these frameworks and the respective profiles do not offer a precise connection between the different views of the SoS, which would be required for design exploration technologies and integrated tool chains. In addition, significant system properties (e.g., real time, reliability) are not addressed in the state-of-the-art modelling frameworks.

Many SoS are real-time systems and temporal correctness needs to be considered in the development process. Introducing a temporal specification and timing analysis in early development phases of the SoS architecting process avoids costly redesign in later phases when the SoS fails to meet its timing requirements. The same applies to SoS reliability where the resulting system architecture fails to meet the system's reliability requirements.

1.2 Thesis Contents

Chapter 1: is an introduction to the thesis and its subject. It starts with an overview of SoS and the contents of the thesis. It also contains multiple examples of different applications of SoS in different disciplines. The last section of the chapter presents the research gap and the research contribution.

Chapter 2: covers the terms and concepts that are used in the thesis. It defines systems and SoS concepts and their characteristics, and presents a comparison between both domains. It investigates the real-time properties of the system and provides a definition of dependability with its attributes. At the end of the chapter, the model-based system engineering approach is discussed.

Chapter 3: discusses the state of the art in SoS engineering and modelling. It gives a detailed presentation of the SoS engineering process and activities. The chapter lists the main modelling frameworks which are used in SoS engineering, compares their capabilities, and presents different modelling languages that are used for system and SoS modelling. Furthermore, the chapter discusses the design space exploration and its application to the SoS architecture domain. The rest of the chapter illustrates the research gaps in the state of the art.

Chapter 4: this chapter presents our development methodology for SoS. The first part of the chapter describes the re-usability of the models using architecture patterns. The chapter continues with architecture optimisation and the combination between architecture patterns and architecture optimisation in SoS architecture modelling. At the end of the chapter the modelling methodology is described in details.

Chapter 5: the chapter describes the application of our methodology for SoS with real-time requirements. At the beginning the timing analysis of SoS is presented. Next, the chapter discusses how to integrate the timing analysis in the architecture development methodology and ends with describing the optimisation process.

Chapter 6: in this chapter the reliability analysis of SoS is presented. The chapter defines the SoS reliability block diagram and shows the connection between the diagram and the architecture development methodology.

Chapter 7: introduces three examples for applying the methodology. The first example is an architecture optimisation problem with geographical allocation of the constituent systems and communication coverage optimisation. The second example shows how to include the real-time requirements in the architecture methodology. The last example is about satisfying reliability requirements in the architecture methodology as optimisation objectives or constraints.

Chapter 8: the chapter presents the conclusion of the thesis and a future outlook for applying the methodology.

1.3 SoS Examples

SoS applications can be found in different domains including military, aerospace, space, health, manufacturing, transportation, environmental systems, and many others. In the following subsections different examples of SoS are discussed and applications for different domains are presented.

1.3.1 SoS in Defence Applications

Many military applications are examples of SoS where the integration of capabilities from various constituent systems is required for accomplishing the military missions. The highly precise weapons that are recently used in military are based on information systems that analyse information received from other systems (e.g. surveillance and radars) to define the target and plan the mission. In order to manage and integrate the capabilities of the involved constituent systems, and to achieve the best use of technological development, some governments (e.g. UK and USA) moved towards SoS concepts in understanding the defence systems, especially in planning and designing mission capabilities [5].

An example of a SoS in the defence area is Integrated Air Defence [6]. An integrated air defence system consists of multiple autonomous constituent systems that include surveillance radars, passive surveillance systems, missile launch batteries, missile tracking and control sites, airborne surveillance and tracking radars, fighter aircraft, and anti-aircraft artillery. The systems exchange data and information using a communication network with lo-

cal, regional, and national command and control centres. The systems collaborate with each other to realise an emergent behaviour to achieve specific missions, e.g. targeting a tank of an enemy at a battle requires the collaboration between surveillance systems that determine the tank's location and the fighter aircraft that fires the missiles. In this example there are two levels of services: the SoS global services, where the integrated air defence performs the military mission, and the constituent systems' local services, where the surveillance systems offer their services to other constituent systems. Each of the involved constituent systems has its commander who is controlling its processes, but they also offer their services to each other and work together to achieve the appointed mission according to the defined plan. Each of the constituent systems is independent in its existence and can perform limited operations. The fighter aircraft can perform the mission of destroying the target without using the services offered from the other constituent systems but at a low level of accuracy and precision and it will be under the threat of attack by enemy weapons. An accurate mission requires the emerging capabilities of the SoS that provides the fighter aircraft with the required data to hit the target and avoid the threats.

1.3.2 SoS in Emergency Response

Another example of an SoS application is an Emergency Response SoS (ER SoS) [7]. Figure 1-1 depicts the overall operational concept and mission of the ER SoS. The ER SoS consists of a number of individual constituent systems, all having their own life cycle, partially with managerial independence distributed over multiple geographical places. The constituent systems are connected to each other through a communication network. They exchange information and distribute their resources based on the required mission.

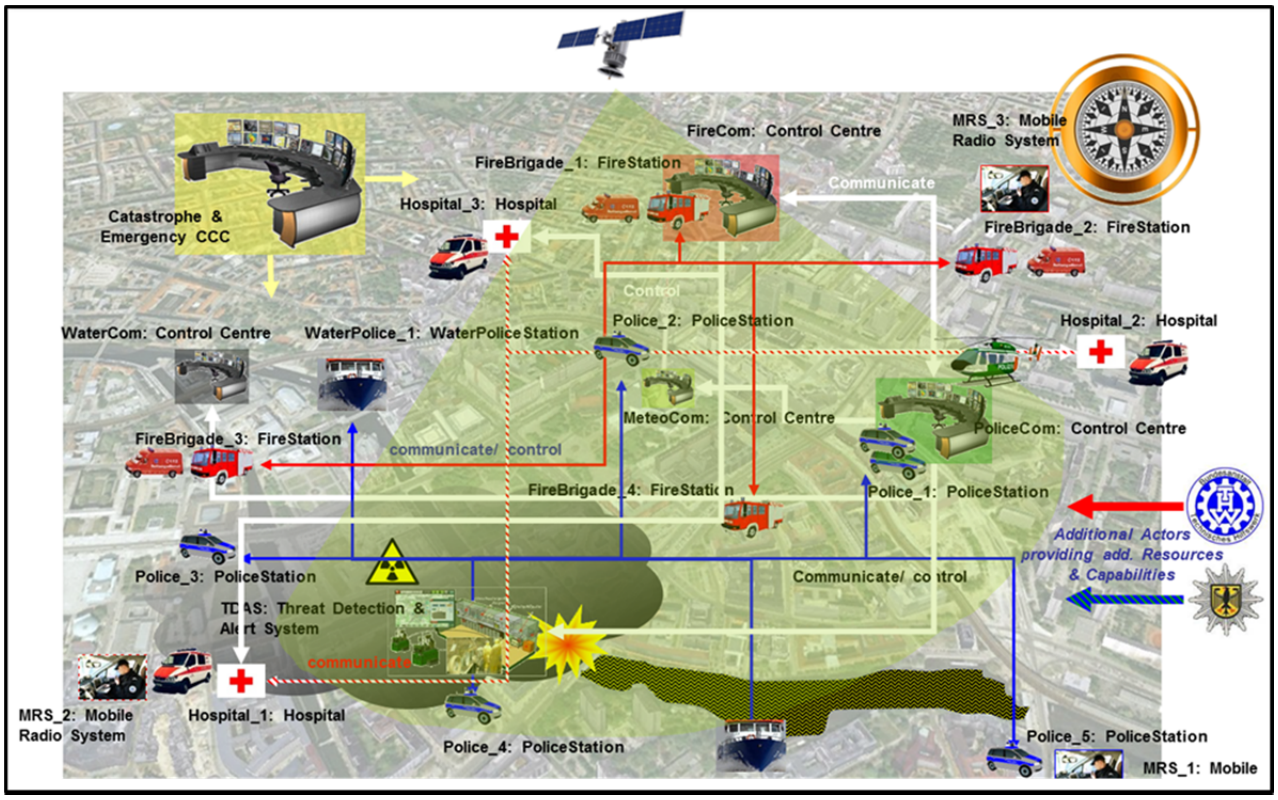

Figure 1-1 Emergency response SoS operational concept [7]

The fire brigades are distributed across the city and work jointly with the police stations. Each of them may be controlled by individual Command and Control Centres (CCC) in order to mitigate the emergency or incident. A

lot of other ER Resources are also involved. For example, the hospitals provide ambulances that try to evacuate and rescue injured people, the Water Command and Control controls the local water resources, the surveillance systems send information about the emergency area and overall situation to the CCCs, mobile nodes such as a Threat Detection and Alert System detect further information or possible danger close to the emergency scene and finally different types of communication systems are involved to facilitate the communication between different operational nodes in the ER SoS.

The emergency response mission cannot be achieved with one system alone; it obviously requires joint capabilities of the mentioned constituent systems. Depending on the assigned mission, the rescue operations are assigned. The fire brigades require the support of the police stations at different stages, the surveillance systems provide the information about the site to the involved partners, and they all exchange data through the communication systems. The same applies to the other constituent systems such as water command and control, medical systems, and external organisations.

1.3.3 SoS in Transportation Applications

Recently, the transportation system development and management has been investigated in the SoS context for better understanding of transportation systems and to provide enhanced solutions. In [8], the author introduces the transportation system as a SoS with heterogeneous systems serving in different domains (e.g. aircraft, trains). These systems are geographically distributed at a national and an international scope.

Most of the transportation systems are operationally independent but coordinate themselves with each other. For example, commercial aircraft operate independently but are coordinated with other aircraft while they are flying. As in most SoS, a communications network is required for exchanging the information between these systems.

Transportation systems keep evolving over the time. Dynamic developments and investigations are required to adapt to the population growth, to new technologies, and new systems in the transportation domain. Transportation systems are characterised by their dynamicity where new constituent systems are introduced and old ones leave.

The constituent systems in many transportation systems are affected by each other. The traffic depends on the coordination and collaboration between the involved constituent systems. In some cases a command and control centre keeps monitoring and organising the movements of different transportations systems. However, the emergent behaviour is the most important characteristic for this kind of systems where the constituent systems exhibit a high degree of autonomy. The emergent behaviour becomes problematic when it results in undesired emergence that leads to accidents and delays in the traffic. For example, in managing and organising the traffic with several trains that are using the same track, many constituent systems are concerned in monitoring the track, providing the location information of the trains, controlling the traffic on the track, and facilitating the data exchange between the involved constituent systems. If the desired emergent behaviour for organising the traffic is not achieved, trains will be exposed to accidents.

1.3.4 SoS in Space

Gianni et al. presented in their work a SoS space methodology [9] with two examples of SoS space applications: Galileo-COSPAS/SARSAT and the Global Monitoring for Environment and Security (GMES).

Galileo is a European Global Navigation Satellite System that supports global timing and navigation capabilities [10]. One of its applications is an international satellite system for search and rescue called COSPAS/SARSAT [11]. Figure 1-2 illustrates a typical search and rescue scenario. The SoS consists of diverse constituent systems in different disciplines: surveillance systems that provide information about the investigated target, satellite systems that offer the communication services, operational systems like helicopters and other rescue tools.

The constituent systems that are part of the SoS, can also be part of other SoS. The satellite system is a part of the rescue SoS but it is also offering other services for civil or military communications. The wireless communication between the constituent systems enables them to be geographically distributed over a wide area. The constituent systems often have different life cycles. The satellites are usually developed to serve for a long period to be cost effective, while local controllers that are used for communication like the hand-sets can have shorter life cycles to integrate new technologies and provide opportunities for having new capabilities.

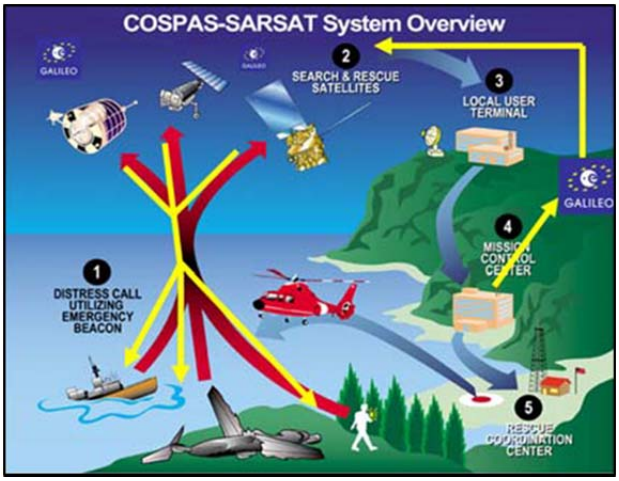

Figure 1-2 Search and rescue high level operational scenario [9]

Another example of a SoS in space is the Global Monitoring for Environment and Security Program (GEMS) [12]. It is a European program that aims to establish a European Earth Observation capability. The GEMOS SoS collects data from different assets (e.g. satellites, air and ground stations) and offers services for constituent systems that are responsible for monitoring and forecasting the environment resources and changes. GEMS is intrinsically a SoS configuration [9], which comprises multiple constituent systems and assets that provide information and support to deliver earth observation services, such as oil spill tracking, farming and air quality.

1.3.5 SoS in Health

SoS are also used in different health and medical applications. The National Healthcare Information Network (NHIN) [13] is an example of a complex SoS where a large community of health systems is involved. The design and development of the NHIN was launched in 2006 by the US Department of Health and Human Services and aims to create a technical framework that allows sharing of medical data between care providers and to create an electronic health record for US citizens [14].

The NHIN integrates heterogeneous care data from care providers (e.g. hospitals, physician, and patient care) and presents it to its users (e.g. hospitals). It is based on a health network that communicates with a large number of medical information systems used in hospitals and physician care offices.

1.4 Industrial Needs

In industry two types of SoS can be distinguished: new SoS are developed from scratch according to the customer requirements and needs, whereas legacy SoS are extended by adding new functionality and new constituent systems. The development process can also be differentiated according to the geographical distribution of the SoS, where policies, protocols and standards are considered based on the scope of the deployment (i.e. global, national, urban, regional) and the operational domain where the SoS provides services.

The SoS development process is challenged by the needs of the customers who are always concerned with cost-effective optimized solutions in order to increase the SoS capabilities and performance. The design of an SoS requires a large degree of trade-offs between different SoS architectures and ensuing properties, while showing the level of confidence in such trade-offs.

In industry, the customer needs for the SoS are first described at an operational level, where operational scenarios are created to depict these needs. The operational level is then reflected on the established SoS architecture. In order to keep the development process consistent between different development stages, it is important to maintain the communication between development teams and establish a common picture of the developed SoS. The absence of a common picture between the developments teams leads to an inconsistent understanding of the required solution and different interpretations of the customer requirements.

The state of the art of SoS modelling lacks executable models that are required for the design, verification and analysis (e.g. simulation). Existing models also miss the mapping from the SoS requirements to the constituent system requirements. The combination of constituent systems from different disciplines leads to a large numbers of interfaces and data flows that are prone to problems and errors at later stages of the SoS development. It is required to develop methodologies that adapt these models to the dynamic evolution of the SoS and the constituent systems as well as new technologies and environmental changes.

SoS models must facilitate the information exchange and interaction between customers and developers during the requirements analysis. Due to the complexity of the SoS, customers are typically not involved in later stages of the development process. Developing SoS models is a costly and time consuming process. Therefore the modelling process should be simplified and the established models should be re-usable for new similar projects.

In the state-of-the-art of SoS frameworks, there are no common methodologies and guidelines between developers for modelling SoS and establishing the SoS architecture. The methodologies must ensure interoperability when integrating constituent systems in pre-existing SoS and maximize the SoS capabilities. The SoS capabilities result from the combination and integration of different constituent systems. The SoS development process must optimize the architecture to connect the constituent systems with regards to functional and non-functional requirements to deliver efficient SoS capabilities that enable the SoS to provide its services.

1.5 Scientific Contribution and Research Problem

In the previous section, the industrial needs for the SoS development process were discussed. There is a significant research gap between the state-of-the-art techniques for SoS development and the industrial needs. The developed SoS models are not connected to executable models, thus preventing testing and simulation of the SoS architecture prior to the actual implementation. There are no SoS development methodologies that would con-

nect different stages of the SoS development and map those to the constituent systems' executable models that are used in simulation, testing and model checking. Furthermore, the available architecture frameworks do not offer optimization techniques that consider optimizing the SoS architecture for functional and non-functional requirements, because of the poor mapping between the SoS models and the constituent-system properties.

In addition, the re-usability of the models is a critical factor in reducing cost and time for developing new SoS that are similar to previously developed ones. In the state-of-the-art, the SoS models are not suited for reuse and are not consistent.

The scientific contributions of this thesis beyond the state of the art are as follows:

1. ***A Model Based System Engineering (MBSE) methodology for SoS architecting based on the UPDM profile*:** We introduce a SoS architecture modelling methodology that organises the SoS engineering activities for building the SoS architecture based on the operational analysis of the SoS requirements. The methodology is based on existing SoS modelling frameworks and extends the UPDM modelling language. It connects the different SoS architecture development steps and enables architecture exploration and architecture generation with the required functional and non-functional properties.
2. ***Extension of SoS modelling languages and processes for reliability and real time***: the state of the art of the architecture frameworks and modelling languages lacks the support for real-time and reliability requirements. We extend the UPDM profile and the architecture frameworks by introducing stereotypes, diagrams and tool support for real-time and reliability analysis of the SoS architecture.
3. ***Architecture optimisation process for SoS architecture generation***: the introduced methodology supports the automatic generation of an optimized SoS architecture with regards to functional and non-functional requirements (e.g., real time, reliability, cost) based on Mixed Integer Linear Programming (MILP) connected to architecture patterns that enhance the model reusability.

These contributions enhance the SoS development by delivering a consistent methodology with a step-by-step process. The produced SoS models enable developers to analyse the functional and non-functional requirements, thereby reducing the risk of SoS architecture design faults at early development stages before the actual implementation. The impact is a significant reduction of time and cost in SoS development.

2 Concepts and Terms

This chapter offers definitions of the main concepts and terms used in this thesis. It starts by defining fundamental concepts of monolithic systems in section 2.1. In section 2.2, the SoS definition is illustrated and characteristics of SoS are discussed. In section 2.3, a comparison between SoS and monolithic systems is given based on different perspectives.

The second part of the chapter defines the functional and non-functional requirements that are considered in this thesis. In section 2.4 real-time systems, their classifications and properties are presented, while dependability and its attributes as well as reliability concepts are defined in section 2.5.

The rest of the chapter provides an overview of the model-based systems engineering approach and the principles of modelling. This is presented in section 2.6.

2.1 Monolithic Systems

2.1.1 Definition

The standard definition of a system is a combination of interacting elements organised to achieve one or more stated purposes [15]. A system consists of inter-related elements called components. They are connected to each other within the system's structure and are organised to achieve a function or a set of functions. The components of a system form a unified whole that is delineated by the system's boundaries. These boundaries identify the system and separate it from its surroundings. The relation between a system and its surroundings is governed by interfaces that organise the system's interactions with its environment through inputs and outputs as shown in Figure 2-1.

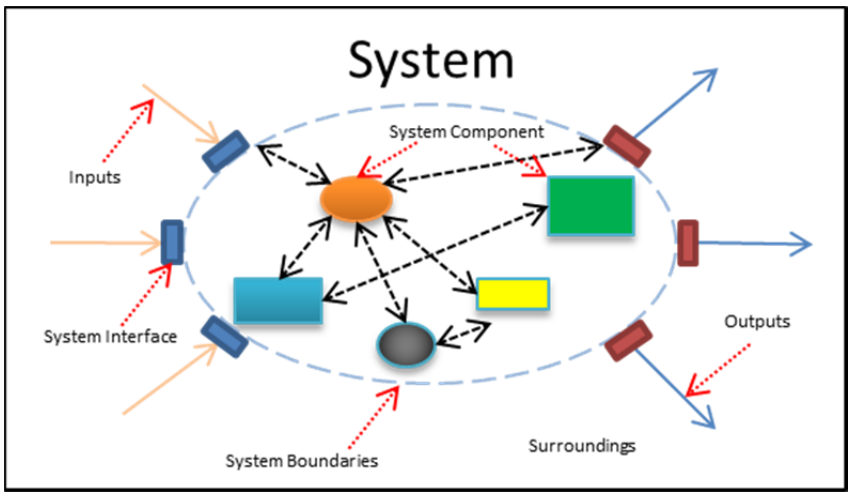

Figure 2-1 System definition

In the ISO/IEC 15288 standard a system is considered as 'man-made, created and utilised to provide products and/or services in defined environments for the benefit of users and other stakeholders. These systems may be configured with one or more of the following system elements: hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials and naturally occurring entities. In practice, they are thought of as products or services' [15].

2.1.2 Characteristics

The aim of characterising monolithic systems is to facilitate the development of these systems and to share a common understanding of what represents a monolithic system. The following characteristics of systems are the most common ones:

2.1.2.1 Inter-related Components

The components of a system are discrete parts of the system that are composed to perform the system's tasks. They are elements that can be classified into hardware, software and humans. They are designed according to the system's requirements and they may be independent or dependant on each other for accomplishing the required task.

2.1.2.2 Structure

The structure of a system includes a set of components and their relations to fulfil the system's requirements (see Figure 2-2). These components are related directly or indirectly to each other and their role is defined in the system's structure. The structure of the system may be represented in a hierarchy or any other representation depending on the system's size and complexity. The definition of the components within the system's structure depends on the abstraction level. Some systems are decomposed into subsystems that consist of further components.

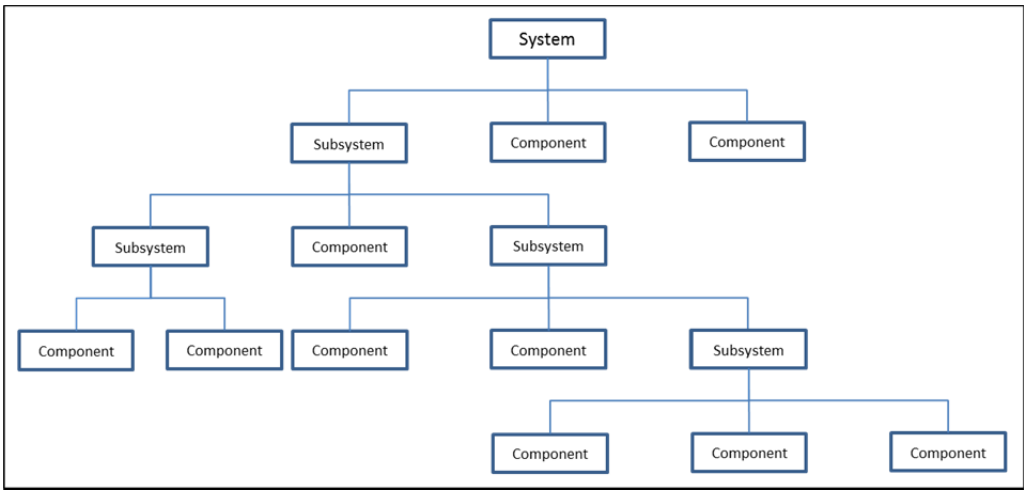

Figure 2-2 System structure

2.1.2.3 Boundaries

The boundaries of the system encapsulate its components and separate them from the external entities (as shown in Figure 2-2). These boundaries can be physical or virtual according to the type of the system. For example, hardware systems have clear physical boundaries, while systems with software components have virtual boundaries. It is important to define the boundaries of the systems to determine the scope of the system and the system's interaction with the surroundings and other systems and to realise the inputs and outputs of the system. The system interacts with other systems and its environment through interfaces that control the system's interactions with the external world outside the system's boundaries.

2.1.2.4 Specifications

Systems are designed to fulfil specific user requirements by delivering services to their environment. The specifications of the system define the service [16]. The system fails when its behaviour violates its specifications. Thus, every system must have specifications that determine its behaviour and shape its interactions with the surroundings. In a correct system, components are working together to achieve its expected behaviour for which it was designed for.

2.2 System of Systems

New technological developments and growing human needs impose challenging requirements to the system developers. These requirements increase the system complexity and direct the developers' interest towards hierarchical systems whose components are complex systems themselves. Such a large complex system consisting of heterogeneous, large scaled and geographically distributed systems is a System of Systems (SoS).

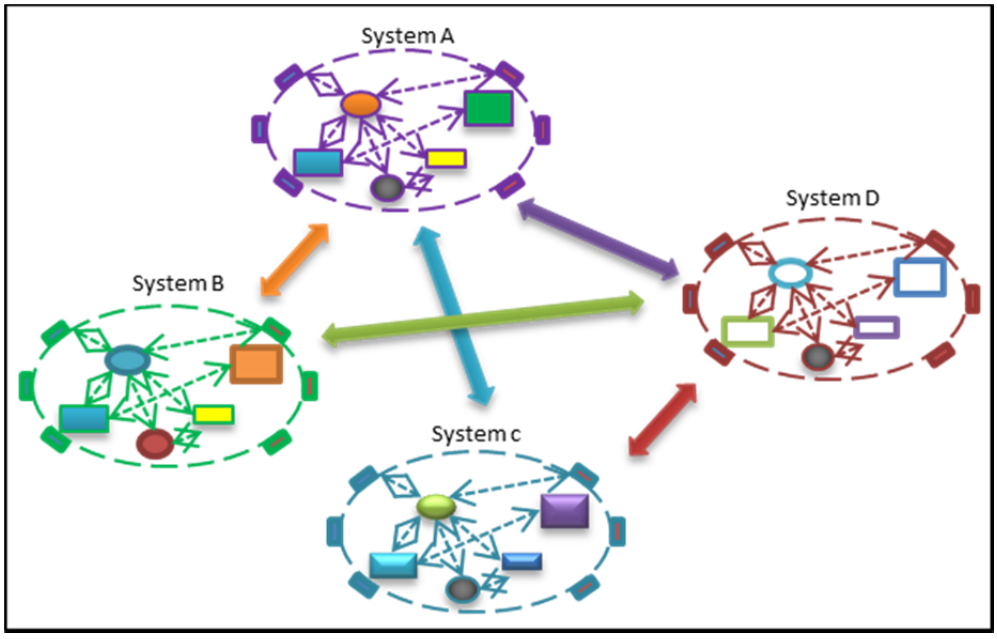

Figure 2-3 System of Systems (SoS)

The SoS perspective was presented in 1991 [17] in order to reduce complexity and manage complex systems that were integrated to achieve emergent purposes. The main idea behind the SoS is to obtain higher capabilities and emergent services that would not be achievable by a monolithic system [18]. The SoS perspective is a way of structuring and conceptualizing complex systems by decomposing them through different viewpoints, abstraction levels and hierarchies to bounded and controllable models that help in design, implementation and analysis.

2.2.1 Definition of SoS

In the literature and previous studies on SoS there is no universal definition in the system engineering community [18]. All definitions that were presented consider different aspects and properties of SoS according to the purposes and the applications that the SoS are used for.

In [18] several definitions of SoS are listed, each of them concerns different perspectives. The SoS is defined as an enterprise that joins the activities of monolithic systems with enterprise activities. Another definition is based

on viewing the SoS as an integration and interoperability of constituent systems that enhances the overall performance. SoS can be viewed as synergism of diverse constituent systems.

Maier [6] defined the SoS by distinguishing it from very large and complex but monolithic systems. He presented five principles for characterizing the SoS:

1. Operational Independence of Elements: The SoS consists of constituent systems that are able to operate independently when they are disassembled and detached from the SoS. The constituent systems are existing systems that deliver useful services even if they work independently of the SoS.
2. Managerial Independence of Elements: The constituent systems keep operating independently even after joining the SoS. They perform their managerial operations on their own after joining the SoS.
3. Evolutionary Development: The SoS always changes over time according to the added, removed and modified constituent systems. There are two inter-related dimensions of evolution: (1) the SoS evolution with new functions and purposes and (2) the constituent systems evolution with new technological developments.
4. Emergent Behaviour: Emergent behaviour best distinguishes SoS from monolithic systems [19]. The combinations of the constituent systems result in emerging services that do not reside in any constituent system.
5. Geographic Distribution: The constituent systems of an SoS are geographically distributed. They are connected to each other by communication networks that allow them to exchange information.

Sahin defines SoS as 'large-scale concurrent and distributed systems that are comprised of complex systems' [18].

A combination of the above definitions can best describe the SoS and its constituent systems: Systems of Systems are large-scale concurrent emergent and evolutionary systems that are comprised of operationally and managerially independent, geographically distributed complex systems that are networked together to achieve a common goal.

2.2.2 Characteristics

[20] lists five main characteristics of SoS derived from different SoS definitions: autonomy, belonging, connectivity, diversity and emergence.

2.2.2.1 Autonomy

The autonomy combines two of Maier's principles, namely operational independence and managerial independence. In the SoS context autonomy denotes 'the ability to complete one's own goal within limits and without the control of another entity' [21]. In other words each entity can exercise independent actions or decision making [19].

Autonomy is related to the constituent systems of the SoS which are utilised to fulfil the purpose of the SoS [22]. Each of the constituent systems is self-controlled and can take its own decision. However this autonomy must

not affect the linkage between the constituent systems of the SoS that is needed in order to achieve the overall goal.

2.2.2.2 Belonging

In monolithic systems, a component is chosen to be an integral part of the whole system. If the component is separated from the system, it will not be active or useful anymore. However, in SoS each constituent system is an autonomous system and can chose to belong to the SoS or not according to its goals and cost/benefit bases [22] to cooperate and connect to other constituent systems of the SoS to achieve common goals.

In order to belong to a SoS, the constituent system must be able to extend its goals to the holistic goals of the whole SoS or of a network of constituents systems [17]. This implies that new connections and communication channels will be established.

2.2.2.3 Connectivity

The connectivity property is required to achieve interoperability between constituent systems. Connectivity can be defined according to [21] as the 'capability to form connections as needed to benefit the entity'. Thus, the connectivity enhances the capabilities of the SoS [22] and provides a prerequisite for establishing the emergence of SoS services especially in the case of geographically distributed systems. Connectivity refers to the system ability to efficiently exchange information or materials between constituent systems to achieve a specific function.

2.2.2.4 Diversity

The SoS joins the capabilities of different constituent systems to accomplish the overall SoS goals. As described in [22], SoS diversity increases the overall capabilities that can be achieved by autonomy, committed belonging, and open connectivity. SoS are designed to fulfil the requirements that cannot be reached by one monolithic system in isolation. [20] indicates the importance of SoS diversity by stating that 'a SoS should, out of necessity, be incredibly diverse in its capability as a system compared to rather limited functionality of a constituent system, limited by design'.

2.2.2.5 Emergence

The purpose of constructing SoS is to provide emergent services that could not be achieved by one constituent system alone. The emergent services result from the cross interaction between the constituent systems that belong to the SoS and offer their services to each other. For example, in a military mission the air force system, surveillance systems, and communications systems work together in order to destroy the enemy's tanks, thereby constructing a SoS with clearly defined goals. Another example of emergence [5] is the symphony produced by an orchestra. The symphony is produced due to the interaction between different instruments and music players, where none of the music players can produce the symphony in isolation.

While integrating the capabilities of heterogeneous systems brings new desired services, it may also lead to undesired emergent behaviour because of the autonomy of constituent systems and their unexpected interactions. According to [17] the emergence of SoS cannot be foreseen through analysis because it comes from collaboration and autonomy of constituent systems'. In [5] the author presents the preferred definition of emergence as

'something unexpected in the collective behaviour of an entity within its environment, not attributed to any subset of its parts, that is present (and observed) in a given view and not present (and observed) in any other view'. As emergent behaviour is unexpected, frameworks facilitating the emergence of desired behaviour and mechanisms for the early detection of undesired emergent behaviour are required [22].

Emergent behaviour can be classified into two categories according to its effect on the behaviour of the SoS: desired and undesired emergent behaviour. A desired emergent behaviour represents an opportunity to the system and it is the purpose of building the SoS in the first place, where a monolithic system cannot fulfil the requirements. Uncontrolled undesired emergent behaviour forms a risk for the SoS and should be avoided.

Since the number of potential interactions increases significantly with the number of constituent systems, emergent behaviour cannot be predicted in the current state-of-the-art. At present, the choice of the constituent systems for constructing the SoS typically depends on their capabilities, communication characteristics and their individual constraints, without considering the behaviour of these systems when they are working together. The problem with undesired emergent behaviour is its consequence on the environment, the SoS, and the constituent systems. For example, undesired behaviour of water controllers in a national water management SoS can be caused by unexpected high climate temperature. The result is improper pressure of the pumped water that may cause a damage of water pumps and disturbances of water supply for the inhabitants. Unexpected behaviour can prevent the SoS from offering its services, which can even imply the loss of lives in safety-critical systems such as military or emergency cases.

2.2.3 Classification

Based on the construction purpose and managerial aspects, SoS can be classified according to four categories: directed, acknowledged, collaborative and virtual SoS.

- Directed: a directed SoS has specific purposes and is designed and built to achieve particular goals. In this case, the SoS is managed and owned by an announced owner (i.e. a controlling authority or a primary stakeholder) who identifies the purposes that he targets by forming the SoS. The operations of the SoS are centrally managed to fulfil the SoS objectives without affecting the individual properties of the constituent systems. An example of a directed SoS is a military system which is built to achieve a specific mission. Each of the involved constituent systems provides its functions and services to the SoS in order to achieve the planned mission.
- Acknowledged: it is a type of SoS where the constituent systems have their independent ownership and their own goals and budgets while at the same time belonging to a SoS with a designated manager and declared objectives [23].
- Collaborative: the constituent systems voluntarily collaborate to achieve the agreed purposes without a coercive power from a central management organisation. All the involved systems benefit from joining the SoS which keeps improving and developing over time. An example of a collaborative SoS is the Internet [6], where all the systems involved join the network and exchange information without central management.

- Virtual: a virtual SoS has no agreed common purpose and lack of central management. The constituent systems realise a benefit by developing a SoS and interact with each other within a complex network. An example of such a system is the national economy [6].

2.2.4 Classification of Constituent Systems

SoS can be constructed from different types of constituent systems, depending on the role that each constituent system plays and its contribution to the overall goals of the SoS. The constituent systems can be classified as human systems, technical systems and third party systems:

- Human systems: they include all human activities and efforts used to manage, organise, control, and operate the operational activities of the SoS.
- Technical systems: these are constituent systems that are fully owned and controlled by the SoS owners.
- Third Party systems: constituent systems that offer services to the SoS according to the specifications. The third party systems are owned and controlled by different owners.

As examples of constituent systems according to this classification, consider an emergency Command and Control Center (CCC) SoS. It contains technical constituent systems that are owned by the CCC, e.g. the internal communication network that is fully controlled by the CCC information department. The CCC uses external constituent systems to support the communication with its employees, e.g. Tetra and LTE. These constituent systems are third party systems that offer communication services which are controlled by the local owner of the network. The employees of the CCC are an example of the human constituent systems.

This kind of classification is required for modelling SoS, where different diagrams are used to describe the behaviour of the constituent systems according to their types. It is also important to understand the different types of interfaces (e.g., communication buses) of these constituent systems to represent the interactions between them.

2.3 Systems of Systems vs. Monolithic Systems

While SoS consists of autonomous constituent systems, a traditional monolithic system is composed of interdependent components. The literature compares monolithic systems and SoS from different perspectives. Table 1 summaries the differences between SoS and monolithic systems based on fundamental system concepts such as boundaries, components, structure, stakeholders, management and purpose.

It is difficult to define SoS boundaries due to the huge numbers of interaction points between the constituent systems and their environment. The environment of the SoS is the union of the environments of the constituent systems. The SoS consists of integrated constituent systems that are diverse and autonomous. The structure of the SoS includes a network of connections. In contrast, monolithic systems consist of interrelated components that do not provide meaningful services in isolation. An SoS typically has a large community of stakeholders due to the large number of involved constituent systems and in most cases the SoS has no designated manager. In monolithic systems there is one manager of the system and its stakeholders are committed directly to the system.

Further characteristics to distinguish SoS from monolithic systems include autonomy, connectivity, belonging, diversity, and emergence as explained in section 2.2.2.

Table 2 summarizes this comparison [20].

Another differentiation between monolithic systems and SoS can be done by considering the system engineering view as discussed in section 3.2.1.

Table 1 Fundamental system concepts in monolithic systems and SoS [24] [25]

Fundamental System Concept	Monolithic Systems	System of Systems
Boundary	• Delineated by clear external boundaries • Can be open, closed, and dynamic.	• Difficult to define the system boundaries • Diverse constituent systems interact with the environment
Parts and Relationships	Interdependent components comprise a unified whole.	Autonomous and diverse constituent systems contribute to the overall objectives.
Structure, Function, Process	• Well-defined structure with clear component interactions • Designated functions and operations	• Product of multiple constituent systems • Network-centric • Overlapping levels of interactions and connections. • Shared operations in addition to designated functions
Stakeholder Involvement	Stakeholders committed to one system.	• Stakeholders more diverse. • Cross-stakeholders who are involved in more than one constituent system. • Dynamic involvement due to constituent systems dynamicity.
Governance	One project manager and funding.	Depends on the type of the SoS whether it has a designated manager or not.
Mission Environment	A predefined and stable mission.	Multiple missions.

Table 2 Differentiation between monolithic systems and SoS

Element	Monolithic System	System of Systems
Autonomy	The components are fully controlled and owned by the system.	Constituent systems are operationally and managerially independent.
Belonging	The components belong to the system in their nature.	Constituent systems chose to belong to a SoS according to their cost, benefits, and purposes.
Connectivity	High connectivity between components and low connectivity among major subsystems.	Important to enhance SoS capability with many communication possibilities between constituent systems.
Diversity	Components are chosen according to their functionality and their contribution to the system objectives.	Heterogeneous constituent systems are geographically distributed with distinct capabilities.
Emergence	Foreseen and designed as appropriate.	Can be facilitated but not always foreseen.

2.4 Real-Time Systems

In section 2.1 we defined a system as a set of interacting components that represent a whole. In this section we consider another perspective for viewing systems. Here we view the system as it receives inputs, processes these inputs and converts them to output [26] as illustrated in Figure 2-4.

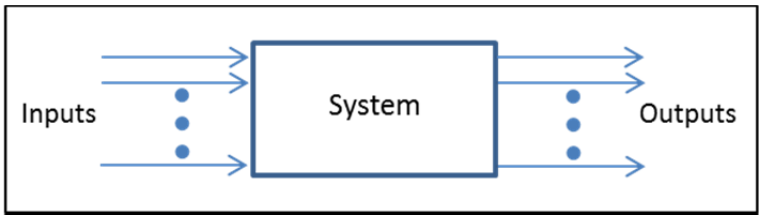

Figure 2-4 System with inputs and outputs

Different timing models and control paradigms can be distinguished for inputs and outputs such as periodic, sporadic and aperiodic messages.

Time-triggered activities are periodically initiated based on the progression of time at specific points in time with respect to a global time base. Event-triggered activities are initiated upon the occurrence of a significant event. Event-triggered activities can be sporadic in case a minimum inter-arrival time between subsequent events is known. Otherwise, event-triggered activities are aperiodic.

For example, in a traffic management system, the command and control system receives regular information from traffic surveillance systems at a specific period. However, in case of accidents it is required that the command and control system establishes a rapid event-triggered reaction to an alarm.

In real-time systems, the correctness of outputs depends on time. For example, consider the dynamic readings of radar systems at the airports, where the position of the aircraft depends on the time of the readings. The timely reaction of other systems, which depend on the position of aircraft, must also consider the changes of the position during the processing of the readings.

Nowadays, real time systems are invading our everyday life in numerous applications such as manufacturing, transportation, emergency systems, traffic systems, medical systems and many other applications where the system's services must meet temporal specifications and deadlines.

2.4.1 Definition

The German industry standard DIN 44 300 defines real-time operation as 'the operating mode of computer systems in which the programs for the processing of data arriving from the outside are permanently ready, so that their results will be available within predetermined periods of time; the arrival times of the data can be randomly distributed or be already a priori determined depending on the different applications' [27].

In real-time systems correctness depends not only on the results of the logical computation of the processing operation, but also on the time at which these results are produced. Real-time systems satisfy timing constraints and requirements in processing their operations.

A real-time system can be decomposed into a computational cluster and an environment (i.e. operator and controlled object) as shown in Figure 2-5.

The computational cluster reacts to stimuli that are initiated from its environment and received as inputs through the cluster interfaces within a specific time interval. It processes the inputs and produces the results within a response time constraint called deadline. The deadline is called firm when the output has no utility after passing the deadline, while it is called soft if the results have utility even if the system fails to meet the deadline.

In distributed real-time systems the computational cluster consists of a set of nodes interacting with each other through a real-time communication network.

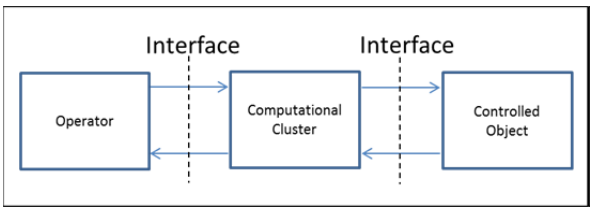

Figure 2-5 Real-time system example

In real-time systems there are three fundamental concepts [28]:

- Time: from an engineering perspective, time is an independent variable that describes the continuum of real-time. In Newtonian based models, time is modelled as a directed timeline divided into an infinite set of instants. Time in the system is realised by physical clock, which partitions the time into micro granules of

the clock. The micro granules are the duration between two consecutive micro-ticks. They can be recorded by an observer that indicates the timestamp of an occurrence.

- State: the state of the system is a data structure that contains the accumulation of the history of the system at a given instant, which is relevant to the future processing.
- Event: an event is an occurrence that changes the system state at an instant.

2.4.2 Classification

2.4.2.1 Hard Real-Time System versus Soft Real-Time System

Based on the characteristics of applications, real-time systems are classified into hard and soft-real-time systems. In hard real-time systems the failure in meeting the deadline can cause a catastrophe, while in soft real-time systems the occasional missing of deadlines is tolerable. This type of classification significantly affects the system design decisions including the provisioning of resources and the fault-tolerance mechanisms. If the system is a hard real-time system, the system must deliver correct results at the specified times even under all considered load and fault conditions, otherwise a catastrophe can happen.

2.4.2.2 Fail-Safe versus Fail-Operational Systems

A fail-safe system can reach a previously identified safe state in the case of a fault. For example, failure detection can be provided with an external device called a watch-dog. A watch-dog is an independent device that monitors the system operations to ensure high error-detection coverage. Once the watchdog detects a component failure, it will force the system to a safe state to avoid a catastrophe as a result of the failure.

Systems which cannot identify a safe state are fail-operational systems. In the case of faults, the fail-operational system must be able to provide a minimal level of service to avoid a critical system failure. There are two levels of failures, component failures and system failures. A fault causes a component failure and the component failure may subsequently cause a system failure, if it is not detected and masked. The component failure can be masked by fault tolerance including redundancy of components.

2.4.2.3 Event-Triggered versus Time-Triggered Systems

Systems initiate a process in response to external or internal stimuli. Depending on the control signal that initiates an activity in the system (also called a trigger [29]) one can distinguish event-triggered and time-triggered systems.

In event-triggered systems the control signal is created by an event. It can be an internal event resulting from the system operation (e.g. termination of a computation), or an external event coming from the system environment (e.g. an input or a service request from a sensor).

However, in time-triggered systems the trigger event is always a change in the state of the global time. The trigger is generated at a particular point in time of a synchronized global time base [28]. Thus the main difference between time-triggered and event-triggered systems is the type of control. The event-triggered systems respond to actions occurring in the system environment as they occur, whereas time-triggered systems preserve their autonomous control and react to the environment according of their internal plans.

2.4.2.4 Asynchronous and Synchronous Systems

The design and implementation process of asynchronous systems is more complex than in synchronous systems. In [30] the author listed five attributes of the system that must be bounded and known in fully synchronous systems, namely the processing speed, the message delivery delay, the local clock drift rate, the load patterns and the difference of local clocks.

In fully asynchronous systems, the mentioned properties are unbounded or unknown. Furthermore it is impossible for distributed asynchronous systems to tolerate even a single crash failure when solving agreement problems [31]. This result is known as the FLP impossibility result.

However, many existing applications are based on intermediate levels of synchrony where the synchrony conditions are partially satisfied.

2.4.3 Temporal Parameters

Real-time systems execute a set of tasks to provide the required services in response to external or internal stimuli. These tasks are characterised by a set of temporal parameters such as:

- Release time: It is the instant of time at which the task becomes eligible for execution [32].
- Execution time: The execution time depends on the inputs, the interference with other tasks, and the properties of the available system resources (e.g. processor speed). The execution time is typically defined as the amount of time required to complete the execution of a task when it executes alone and has all the resources it requires [32].
- Timing model of tasks: Based on the timing model, tasks are classified into periodic, sporadic, and aperiodic tasks. In the periodic task model tasks are executed at regular time intervals. A periodic task is a task that has a constant time interval between successive task requests [33]. In sporadic tasks the arrival time of the tasks is unknown but a minimum inter-arrival time between two successive tasks is known. The aperiodic task has an unknown arrival time with an undefined inter-arrival time between tasks. Most of the hard real-time applications are characterised by a periodic task model where the system has a deterministic work load, in particular in digital control and real-time monitoring applications.
- Period: The period is the exact time between two successive release times of a periodic task.

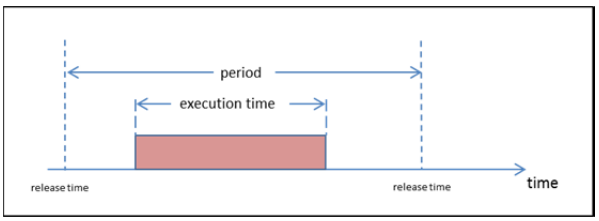

Figure 2-6 Temporal parameters

- Worst-case Execution Time (WCET) and Best Case Execution Time (BCET): The term WCET is defined as the maximum latency required by the task between receiving a service request and delivering the required output. The BCET is the minimum latency required by the task between receiving a service

request and delivering the required output. The WCET and BCET depend on the tasks themselves, their inputs, the properties of the underling computational platform and the interference between tasks [34].

2.4.4 Requirements of Real-Time Systems

The design of real-time systems must satisfy specific requirements including functional requirements, temporal requirements and dependability requirements [33]. In this section we discuss the functional requirements and temporal requirements. The dependability requirements are discussed later in section 2.5

2.4.4.1 Functional Requirements

The functional requirements characterise the behaviour of the real-time system and the activities that the system performs. One can distinguish three main categories:

- Data collection: In real-time systems, the state variables that are used to describe the current state of the system and are significant to a given purpose are called real-time entities. The real-time system must be able to observe the real-time entities with their changes and to collect these observations. It must be able to monitor these entities and detect abnormal behaviours.
- Direct digital control: Many real-time systems are used in control applications where the system directly provides outputs to the controlled object. In this case the system must be able calculate the set values and to deliver the required output within specified deadlines to ensure the stability of control.
- Man-machine interaction: Man-machine interfaces are used for applications where the system requires inputs from operators and provides output to operators. The real-time system must be able to display information about its state to its operators. Meanwhile it must be able to receive instructions from its operators that may include human systems.

2.4.4.2 Temporal Requirements

There are two important temporal requirements coming from the nature of real-time systems: minimum latency jitter and latency boundaries. Depending on the real-time application and the rate at which the system environment changes, the stringency of these requirements is determined.

It is always required that the delay jitter is a small fraction of the delay to avoid errors in the value of the observed real-time entities. However, it is very important to the real-times system to be able to detect and/or mask errors within a minimal latency and with high probability especially in hard real-time systems.

2.4.5 Examples of Real-Time Systems

In this section we present two examples of real-time systems. The first example represents a typical real-time application consisting of a controlled object and a computer. The second example is a navigation system that is connected to other systems within a SoS.

2.4.5.1 Monolithic System: Process Control

In Figure 2-7, a fluid control system is shown. The purpose of the system is to control the flow of a fluid inside a pipe. The flow is controlled through the valve angel that is manipulated by a computer system. At the end of the

pipe a flow meter is mounted. The flow meter measures the actual flow rate and sends the flow readings to the computer system.

The system is required to maintain the flow rate in the pipe at an optimum rate according to the user needs. The system variables are changing over time, e.g. the fluid flow rate.

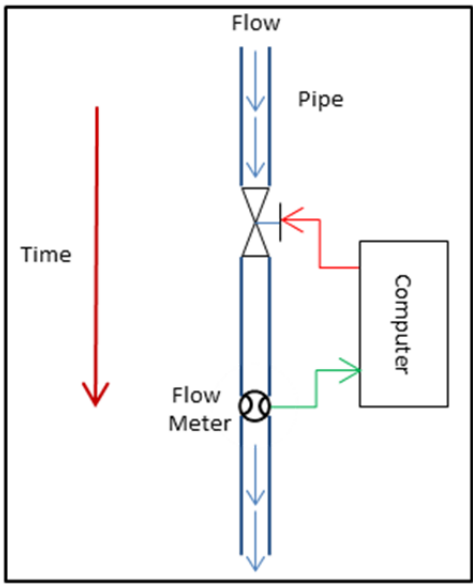

Figure 2-7 Fluid control system

There are two types of information needed by the computer system to control the process:

1. The angel of the valve is measured by an angel sensor mounted in the valve.
2. The fluid flow rate is measured by the flow meter.

The system is in a dynamic change; therefore the computer system must consider the following time delays within the computational process:

a) Computer system computation time of the required valve angle

b) The time required to change the angle of the valve.

c) Sensor delays in reading and delivering the required information

d) Actuator delay

2.4.5.2 SoS: Aircraft Navigation System

An example of a complex and safety-critical SoS is an Air Traffic Management (ATM) system. The ATM system consists of diverse constituent systems distributed over different geographical places (e.g. airports, airplanes, radars). The aircraft navigation system is one of the constituent systems, which is a real-time system that provides data about the aircraft such as velocity and position. The information is distributed over the ATM using communication networks based on Aeronautical Radio, Incorporated (ARINC) standards [35].

In order to measure the velocity of the aircraft, the computational part of the system collects information from different sensors distributed over the aircraft:

1. x, y and z coordinates provided by accelerometers
2. Roll, pitch and yaw angels measured by gyros as shown in Figure 2-8.
3. Environment conditions that may affect the sensor readings, e.g. temperature

Each of the sensors delivers its information at different sampling rates according to the sensitivity and the physical properties of the sensors.

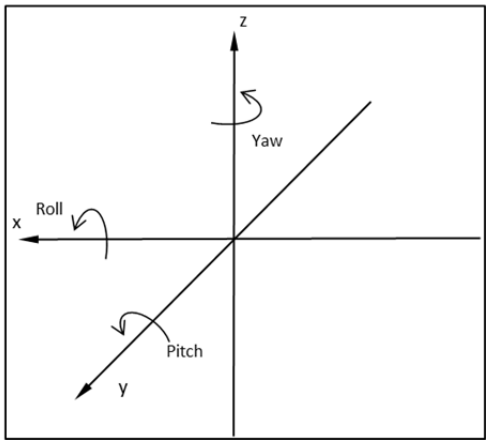

Figure 2-8 Measurements of aircraft navigation system

The navigation system interacts with other controlling systems in the aircraft that require the velocity and position information and meanwhile it displays the readings for the pilot. In parallel, the system interacts with other constituent systems at the ground in the traffic management control center to send them information about the position and status of the aircraft. Figure 2-9 illustrates the concept of the air-traffic navigation system.

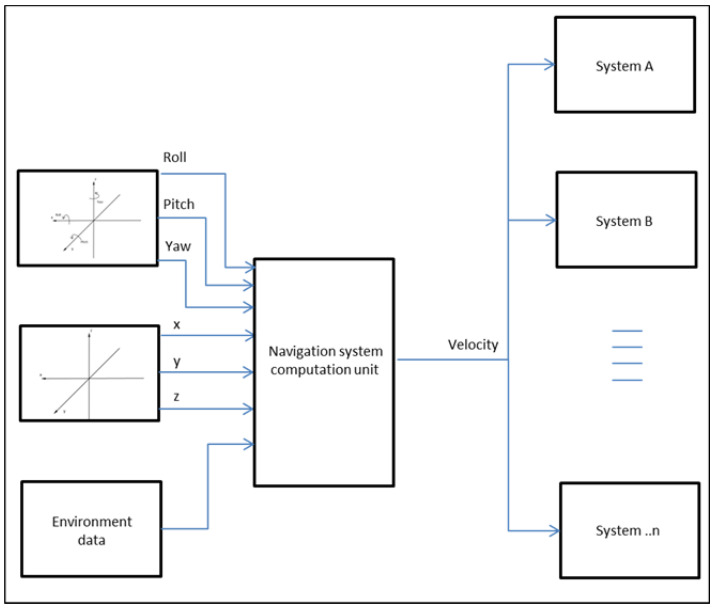

Figure 2-9 Aircraft navigation systems

2.5 Dependability

Dependability is a measure of the ability of a system to operate properly during a specified mission given the starting conditions of the mission. The dependability concept includes three parts according to the dependability tree [36] in Figure 2-10.

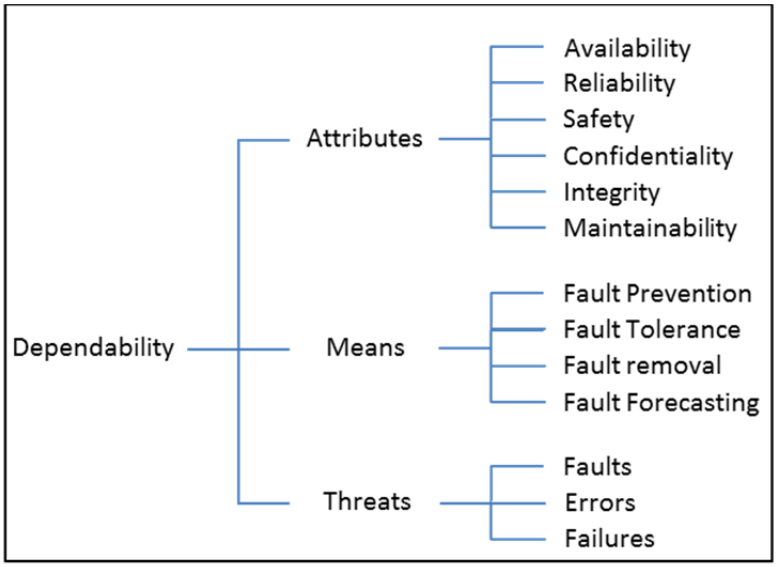

Figure 2-10 Dependability tree

2.5.1 Threats

The dependability threats are failures, errors and faults. A failure occurs when the system stops performing its intended function [37] resulting in the deviation of the behaviour from the specified services. An error is an unintended state [33] which can lead to a system failure. A fault is the adjudged or hypothesised cause of an error [36]. Figure 2-11 illustrates the concept of the fault, error, and failure.

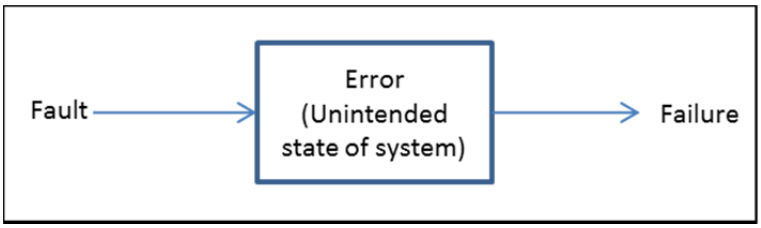

Figure 2-11 Fault, error and failure concept

According to [38] faults can be classified based on the phase of creation into development and operational faults. The origin of the occurrence allows to distinguish between faults that are inside or outside the system boundaries. The cause of the fault is either natural or human-made and in the latter case deliberate and non-deliberate faults can be distinguished. The dimension of the fault can be a hardware or software fault and the persistence of the fault can be permanent or transient. Figure 2-12 shows this fault classification and the taxonomy.

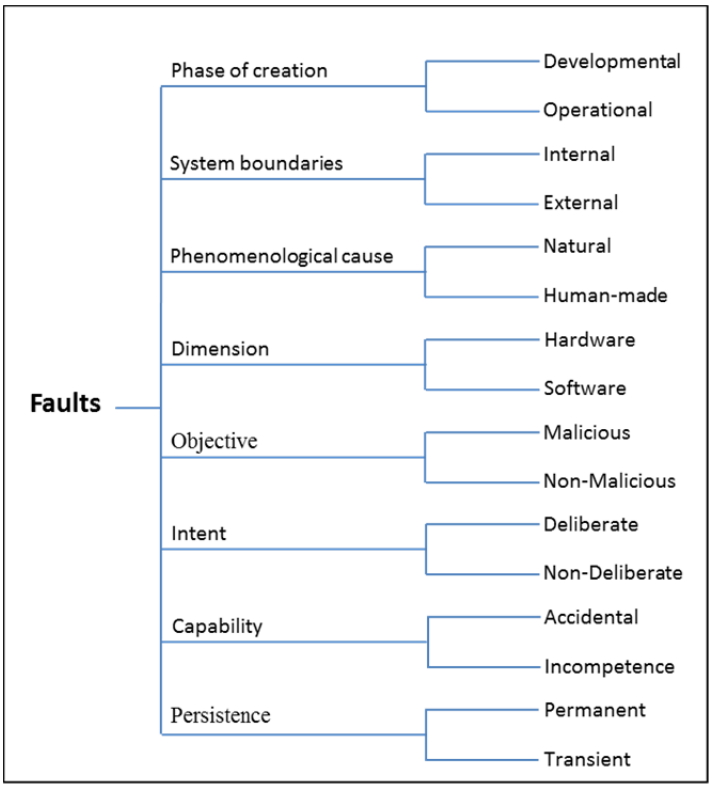

Figure 2-12 Fault classification

Failures are classified according to their domain, consequences, frequency and consistency [33] as show in Figure 2-13. From the point of view of the domain the failure represents incorrect values (i.e., value domain) or the results are delivered outside the intended time interval (i.e., temporal domain).

Based on the effects caused by the failure occurrence, the failure is classified as benign or malign. Benign faults occur when the consequences caused by the system failure are of the same cost or magnitude as the benefits delivered by the correct system. Malign faults occur when the system failure may lead to a catastrophe and the harmful consequences are of higher cost and magnitude than the normal utility of the system.

According to the frequency of occurrence within a given time interval, the failure is transient if it occurs only once and the system continues to operate after the failure. In case of a permanent failure the system stops delivering its service until the cause of the failure is removed.

The last classification of the system failure is according to how the system users view the failure. In case of consistency all system users perceive the same identical failing behaviour. An inconsistent failure occurs when the system users perceive different failing behaviours.

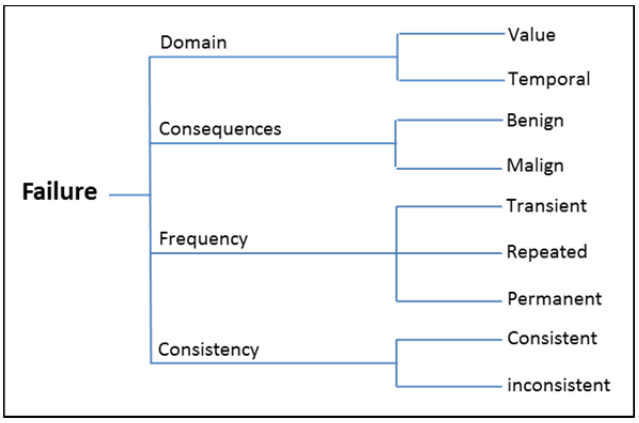

Figure 2-13 Failure classifications

Attributes:

Dependability is described by the attributes of availability, reliability, safety, security, integrity, and maintainability. According to a given application the importance of these attributes varies.

Availability is a measure of the readiness of the system to provide its specified service given certain conditions at a stated time instant. It is measured as the fraction of the time the system is ready to perform its specified service based on the over-all runtime of the system under operation.

Reliability is related to the continuity of service and it is defined as the probability that the system functions properly and continuously in a defined time interval assuming that it was operational at time 0 (see section 2.5.2).

Safety is the ability of the system to avoid the occurrence of a catastrophic failure. Safety is reliability with respect to critical failure modes.

Security is a measure of the system confidentiality and the ability of preventing unauthorized access to the information.

Integrity concerns the ability of the system to prevent improper modifications of the system services.

Maintainability is related to the repairing time that the system needs after the occurrence of a system failure. It characterises the ability of the system to undergo repairs, modifications and evolution.

Means

Four means of dependability are distinguished: fault prevention, fault tolerance, fault removal and fault forecasting.

Fault prevention is a part of the engineering process during the system design and development phase. It deals with the methods and techniques that are implemented to prevent the introduction of faults. It is also a part of the quality control process that is followed during manufacturing.

The purpose of fault-tolerance techniques is to keep the system's ability to provide its services within its specifications even in the presence of faults. It is achieved by

- Error detection methods that indicate the existence of an error
- Recovery techniques that recover the system's ability to deliver its services

Fault removal is an iterative method. It starts with observation and verification activities where the system behaviour is checked against the system specifications. The next step is to launch a diagnosis process in case any violations of the specification are detected. The goal of the diagnosis process is to identify the fault that was introduced during the system implementation process. Then the process is followed by corrective actions to remove faults. Fault removal also takes place after the deployment of a system during maintenance activities.

Fault forecasting encompasses system evaluation techniques regarding the fault occurrence and activation. The evaluation process is either qualitative by identifying and classifying the failures, or quantitative and based on the probabilistic evaluation of the dependability attributes.

2.5.2 Reliability Definition

System reliability is the probability that the system will provide its specified services within a defined time interval given that the system was operational at time t_0. It is the ability of the system to remain functional [38]. The reliability of the system is defined as a function of time and denoted as $R(t)$. Reliability at time t with a constant failure rate λ is given by:

$$R(t) = e^{-\lambda(t-t_0)} \tag{1}$$

Where t is the time and λ is the failure rate.

The failure rate is measured in Failures in Time (FIT) [33], where the inverse of the failure rate is called the Mean Time To Failure ($MTTF$) :

$$\frac{1}{\lambda} = MTTF \tag{2}$$

1 FIT is one failure in 10^9 hours.

The fault hypothesis is a statement about the assumptions made concerning the types and numbers of faults [39], the rate at which the components fail and how components may fail [40]. The fault hypothesis distinguishes between faults that will be tolerated and considered during the system design (covered faults) and those which are not tolerated and will lead to a system failure (uncovered faults) [41]. A precise hypothesis must be defined for the following reasons:

1. Design of fault-tolerance algorithms: the fault hypothesis states which fault-classes will be addressed during the system design.
2. Assumption coverage: It is the probability that the assumptions made in the fault hypotheses with regards to the fault-tolerant system hold in reality [42]. A precise hypothesis is required to predict the probability for the occurrence of uncovered failures.
3. Validation: To validate a fault-tolerant implementation it is required to precisely specify which faults will be considered and tolerated by the concerned system and which faults are out of the scope of the fault tolerance.
4. Certification: The certification of the fault-tolerant implementation needs a precise fault hypothesis for the safety arguments.

Fault Containment Regions

A Fault Containment Region (FCR) [39] is the set of subsystem that share one or more common resources and that may thus be affected by a single fault. Thus, it defines the boundaries of the immediate impact of a fault. The occurrence of correlated failures in multiple FCRs depends on the containment coverage. However, the independence of the FCRs may be affected by the sharing of physical resources (e.g. power supply) or by a common exposure to the same external effects such as electro-magnetic interference.

Failure Modes

Failure modes describe the types of failures in the FCR. Depending on the criticality and the type of the system there are different classifications of the failure modes. In SoS, information about the actual cause of a failure is not always available due to the large number of constituent systems involved, and the autonomy property of theses constituent systems. For this reason, we adopt in this thesis a classification that is independent from the actual cause or rate of failures [42]:

- Fail-stop Failures: the FCR stops to provide the required service or output until it restarts again. It is assumed that the other correct FCR detect or are informed about the occurrence of the fail-stop failure.
- Crash Failures: In case of a crash failure the effected FCR does not provide any services or outputs. Thus the failure may remain undetected by other subsystems.
- Omission Failures: is a communication failure where the sender subsystem fails to send a message or the receiver fails to receive a message, which causes the receiver not to respond to an input. The omission failure may remain undetected by correct subsystems.
- Timing Failure: This failure is related to real-time specifications where the system fails to meet the real time specifications of the output.
- Byzantine (Arbitrary) Failure: This is an unrestricted failure mode with no limitation of the behaviour, which can be inconsistently perceived by different correct receivers.
- Babbling Idiot: the FCR fails to meet its temporal specification by sending untimely messages.
- Slightly-off-Specification: This is a special type of byzantine failure. It can be a value failure where the subsystem provides undefined output values or a temporal type where the subsystem's output timing is

out of the expected time interval. The temporal type can be caused by the inability to perfectly synchronize the clocks between senders and receivers, thereby leading to an inconsistency in accepting a received message. Some nodes classify the messages as timely, whereas another nodes detect a timing failure [42].

- Masquerading: It is a failure of the identity of the sent or received message with an unauthorized identity change.

Failure Rate Assumption

Failure rate assumptions are one of the important parts that must be specified in the fault hypothesis. They are used for the validation of the fault-tolerance mechanisms and in reliability calculations. The assumptions are concerned with the failure rate of the FCR. They differ with respect to different failure modes and the failure persistence.

Maximum Number of Failures

This part of the fault hypothesis characterises the system's ability to tolerate failures, and it is defined as the maximum number of FCR failures that must be tolerated by the system. It depends on the failure rate and the recovery interval of the FCR.

Recovery Interval of an FCR

The recovery interval is the maximum time interval needed by an FCR to recover and provide its intended services after the occurrence of a failure. Its time calculations depend on the failure mode.

2.5.3 Reliability Engineering for SoS

Reliability is a Key Performance Factor (KPF) in building an SoS architecture and in the acquisition of its constituent systems [43]. SoS are widely used in critical applications such as military, emergency response, airports and water management, where a system failure can cause a substantial damage and affect human lives.

Modelling, analyzing and optimising system reliability properties in early design phases of the SoS architecting process avoids costly change efforts in later phases where the SoS architecture fails to meet its reliability requirements. System reliability can be improved by fault avoidance including the enhancing of the quality of constituent systems, reducing the system complexity and practicing a planned maintenance and repair schedule [45]. In addition, system reliability can be increased by fault tolerance such as active redundancy or fault recovery by the reassignment of failed services [45].

Reliability engineering is the discipline where methods and tools are developed to support the system design process in building reliable and maintainable systems by providing techniques for predicting, evaluating and demonstrating system reliability and maintainability [37]. Reliability requirements are considered as extra-functional requirements that assure the system's ability to perform the correct services with a high probability. The target value of system reliability is determined according to the system requirements. During the design process and architecture definition these reliability values are included as optimisation goals and system development constraints.

Reliability analysis becomes more complex at the level of SoS where a malfunction in any of its constituent systems may lead to a SoS failure. We regard each constituent system of the SoS as a FCR, which is independent from other constituent systems with respect to the immediate impact of a fault. Thus, at design time, it is important to analyse the constituent-system reliability and the effect of these systems on the overall SoS reliability to build a Reliability Block Diagram (RBD) [37]. An RBD is an event diagram that provides developers with the information about reliability dependencies within the system. It can provide insight about constituent systems that affect the functionality of the SoS and it provides information about the effects of a constituent-system failure on the overall SoS.

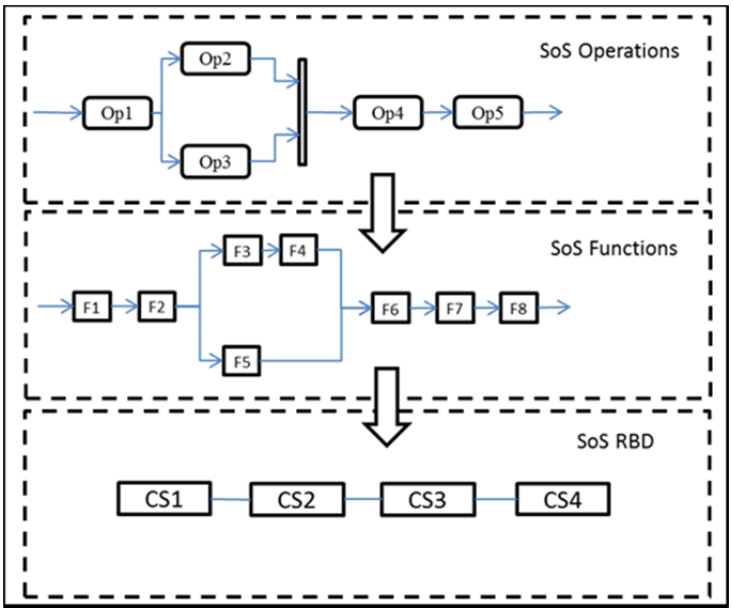

Figure 2-14 Procedure for defining SoS reliability diagram

As shown in Figure 2-14, the operational analysis is the starting point for defining the RBD of the SoS. A SoS operational activity is a group of functions that work together for achieving a certain task of the SoS. It represents a high level of abstraction that could be conducted and understood by SoS customers for building the SoS operational scenarios that define the purpose of the SoS. The procedure for defining a SoS reliability diagram includes the following steps:

- Definition of operational activities: The required SoS operational activities and their flow are defined as illustrated in section 4.4.
- Definition of SoS functions: From the operational level we move to lower levels of abstraction by describing the required functions for the SoS for implementing each of the listed operational activities. These functions are mapped to the operational activities in order to establish the functional flow diagram.
- Definition of SoS RBD: Each function is associated with a constituent system that implements the function. By mapping functions to the constituent systems we refine the SoS reliability calculations from the SoS operations to the SoS constituent systems. Thereby, the SoS reliability diagram is created, which describes the constituent systems and their reliability dependencies according to the functional flow diagram and the functional dependencies.

2.5.4 *SoS Dependability*

To perform the dependability analysis of the SoS, a certain level of abstraction must be defined to avoid overwhelming by the complexity. The SoS is constructed out of constituent systems with different owners and suppliers as well as multiple levels of control based on the SoS type.

The fault hypothesis is defined at the SoS level that includes the constituent systems, where the relations and connections between the constituent systems define the fault containment regions. It is important to define the relation between the constituent systems and the SoS to define the fault-tolerance strategies. For example if the SoS owns the constituent systems then the SoS management has the authority to change the constituent system's components to improve their quality, while in the case where the SoS only uses the services offered by the constituent systems like a service from a surveillance systems, the fault-tolerance strategy of the SoS must consider redundancy at the level of constituent systems.

2.6 Model-Based System Engineering

Model Based System Engineering (MBSE) facilitates the system development by using a set of models that document, formalise and organise the system development process. Models are used at different levels of the system development starting from the requirements phase down to the system verification and implementation. It is an effective and systematic method for sharing the system design, enhancing communication during development and establishing traceability of system changes. Models are also used in the system analysis process where the design space is analysed and verified against the system requirements.

The International Council on Systems Engineering (INCOSE) [46] definition of MBSE is 'the formalised application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing through development and later life cycle phases'.

Traditionally, a document-based engineering approach [47] was used in the design and development process for large projects and systems. Although the document-based approach can be rigorous, it has some limitations on consistency and completeness because the design information is spread over several documents, which makes it difficult to control changes in requirements and to perform engineering analysis.

In MBSE the system engineering activities are shifted from the documentation to the modelling environment. The system specifications and the design information are included in the models of the system which are created using a modelling tool. MBSE is considered as a modelling method which contains a set of activities and techniques to construct a system using a model-based engineering process. Applying MBSE results in the following advantages [47]:

- Enhancing communication: A common understanding of the system is shared across the system stakeholders, which enhances the interaction between system developers and the customers.
- Reduced development risks and improved quality: The connection of requirements to design elements facilitates efficient traceability, verification and validation. The development risks are reduced by verifying the design of the system before the actual implementation.

- Increased productivity: The reusability of the system models reduces the time required to design new systems. MBSE reduces design errors and the time required for integration through rigorous connections between system requirements and design elements and by providing a testing environment with executable models.
- Enhanced knowledge transfer: The system models provide detailed design specifications, development information, and permit the automatic generation of design documentation.

2.6.1 Models of Systems

Modelling is a fundamental technique to understand and simplify reality through abstraction [48]. A model is a representation of one or more concepts that may be realised in the physical world [47]. It can be represented in various forms that include mathematical, graphical, and logical representations. The required documents can be generated from the models of the system whenever needed. MBSE links and integrates different system modelling activities. For example, models of operational activities can be linked to the system's requirements models and the system's functional models can be linked to the models of the system's operational activities (see Figure 2-15).

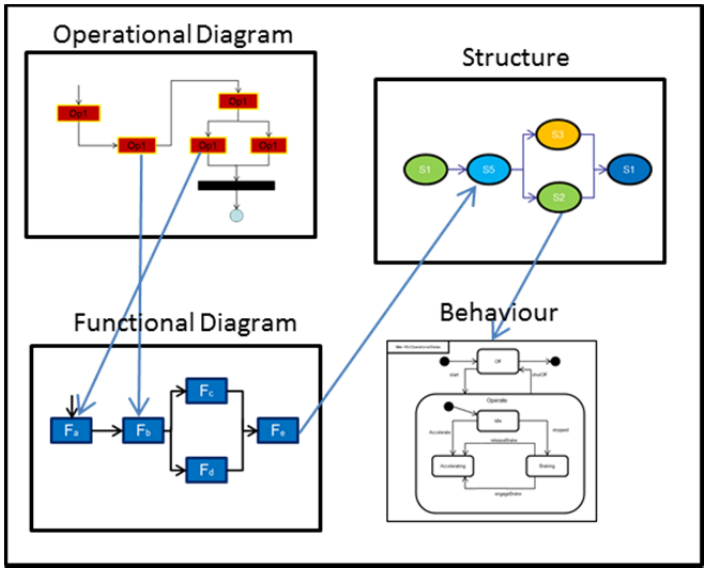

Figure 2-15 System model example

2.6.2 Principles of modelling

In order to build successful models, the following principles must be considered [49]:

- Choice of model diagrams: what diagrams are chosen has a great influence on solving the problem. A wrong diagram choice will be misleading and disperse the focus of the relevant issues that the developers need to address.
- Level of abstraction: The decision of the level of abstraction is based on the purpose of the model. For example, if the purpose of the model is to create an executable specification that supports simulation then the model must be created with a low level of abstraction such as behaviour diagrams. If the model

is used to describe the system's operation to discuss it with customers, then the model is constructed at a high level of abstraction without technical details.

- Connection to reality: The behaviour described in models must be connected to the real world. Building a model with ideal conditions will be misleading and not sufficient. It is also necessary to point out at which points the model is deflected from reality. For example, including or neglecting the environment effects in the model affects the results of the analysis process.
- Multiple independent models: one view cannot illustrate all the perspectives of the model. Also it is needed to show different views that can be separately built with relation to each other to show different details of the model.

2.6.3 Modelling Viewpoints

The models of the system are constructed based on the purpose of the modelling and the required viewpoints. The model viewpoint is determined by the stakeholder concerns of the system of interest. It defines how to view the system and which properties and system aspects the model should express. 'A viewpoint specifies a reusable set of criteria for the construction, selection, and presentation of a portion of the information about a system, addressing particular stakeholder concerns' [50].

Based on the required model viewpoints, the modelling language and the tools are selected. For example, building a model for real-time analysis cannot be used for thermal analysis without introducing the model elements for thermal analysis. It is always needed to define the required viewpoints before constructing the model.

In SoS there are numerous required viewpoints due to the large variety of the SoS stakeholders. It is important to map the different viewpoints to each other to ensure that all stakeholders have a common understanding, and to provide traceability between model elements to avoid problems when changing any part of the model.

3 State of the Art: System of Systems Engineering Practices

In the development of monolithic systems, a system engineering process is used to facilitate the development and implementation of the system. It starts with the requirements elicitation of the customer needs and ends with the final validation of the delivered system. However, in SoS the system engineering process has to address the specific characteristics of SoS such as emergence and dynamicity.

Modelling plays a key role in the development of SoS to support the analysis of the SoS requirements, for designing the architecture of the SoS, verifying and validating the architecture and as the foundation for the realisation of the constituent systems. Modelling frameworks were developed to organise the SoS models and to describe the models using different viewpoints.

This chapter presents related work. It starts by explaining the system engineering process followed by illustrating the specific aspects of the SoS engineering in section 3.1 and discussing the SoS engineering activities in section 3.2.

In section 3.3 different modelling frameworks are described and the capabilities of these frameworks to support the SoS engineering process are summarized. The section is followed by a discussion of a variety of modelling languages that are used in systems and SoS modelling in section 3.4.

Finally the research gap analysis and the discussion of the state of the art are contained in section 3.5.

3.1 SoS Engineering

3.1.1 System Engineering

System engineering is a process concerned with the delivery of successful systems through planning and development activities to fulfil the customer needs. The INCOSE defines system engineering as ‘an interdisciplinary approach and means to enable the realisation of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance training and support, test, manufacturing, and disposal’ [51].

System engineering aims to satisfy the customer requirements by considering the system as a whole and defining the relation between the components and the overall system using different means such as discovering, learning, and diagnosing of the real-world [51]. It includes different kinds of systems including hardware, software, and humans [52].

3.1.2 SoS Engineering Definition

SoS deal with the integration of constituent systems into large scale complex systems. The process of SoS development needs to analyse the integration of different constituent-system capabilities to achieve the desired emergent services. The traditional system engineering processes do not fit to the characteristics of SoS.

Each of the constituent systems is autonomous and interacts with the others by exchanging information. The communication between these constituent systems is the backbone of the SoS, without it the SoS concept cannot be achieved. Thus, the networks between the constituent systems should meet specific design aspects (e.g. relia-

bility, connectivity, timing) to achieve the purpose of the SoS. In addition, the SoS engineering process must also consider:

- **SoS dynamicity over the life cycle**: Constituent systems can leave the SoS and new constituent systems may join.
- **SoS evolution**: Many SoS exhibit continuous evolution over time. The SoS may change due to new environmental conditions and the diversity of the SoS stakeholders. Thereby also the SoS design specifications change over time. The constituent systems themselves evolve at different rates to match the rapid changes of technologies and to meet their local customer requirements. The engineering activities must be able to ensure interoperability of constituent systems over the SoS life cycle and provide the flexibility to adapt the SoS structure.
- **Emergent services**: The SoS engineering must facilitate the desired emergent services of the SoS and avoid undesired emergent effects. The engineering process has to set up a clear plan and strategies to overcome the effects resulting from the occurrence of undesired emergent behaviour.

The SoS engineering process was developed to provide means and methodologies for the SoS development under these constraints. The National Centre for Systems of Systems Engineering defines SoS engineering as 'the design, deployment, operation, and transformation of meta-systems that must function as an integrated complex system to produce desirable results. These meta-systems are themselves comprised of multiple autonomous embedded complex systems that can be diverse in technology, context, operation, geography, and conceptual frame' [53].

3.2 Activities in SoS Engineering

The SoS engineering includes multiple activities that help in defining the development process of the SoS. These activities are organised into seven core elements [24]:

1. Translating SoS capabilities objectives into high level SoS requirements: The realisation of the SoS is initiated by the SoS stakeholder's demand for SoS capabilities that are satisfied by the integration of multiple constituent-system capabilities. The SoS engineering team must understand the SoS stakeholder's needs and expectations of the SoS capability and translate them into a set of technical requirements. It is a continuous process that starts at the beginning of the SoS development and continues during the entire SoS life cycle to identify the requirements changes upon SoS evolution. The requirements identification is followed by the operational activities analysis, where operations are established using a set of operational flow scenarios to satisfy the SoS requirements.
2. Understanding the constituent systems and their relationships: The constituent systems are the core of the SoS. These systems must be understood and identified by the SoS engineering team by recognising:
    - Constituent-system types and capabilities.
    - Constituent-system contributions to the SoS capability.
    - The relationships and interdependencies between the constituent systems.
    - Constituent-system environments and stakeholders.

3. Assessing the extent to which the actual SoS performance meets the capability objectives: The SoS engineering team must establish SoS performance metrics [54].
4. Developing, evolving, and maintaining an architecture for the SoS: The SoS architecture is a representation of the interactions and interdependencies between the constituent systems of the SoS. The architecture must describe how the concept of operations is realised by the interactions between constituent systems. It represents a technical framework for addressing the SoS requirements and supporting the SoS evolution.
5. Monitoring and assessing potential impacts of changes on the SoS performance: this activity is related to the SoS dynamicity and evolution where the SoS is maintained under dynamic changes in its structure and environment. Due to the autonomy of the constituent systems their changes and evolution are not controlled by the SoS management. The constituent systems must be monitored and evaluated by the SoS engineering to analyse their impact on the SoS performance and also to assess opportunities for integrating new constituent systems into the SoS.
6. Addressing SoS requirements and options for solutions: The SoS development process is affected by SoS requirements and constituent-system requirements. The SoS engineering team must collect and analyse these requirements during the SoS life cycle by reviewing, evaluating, prioritising, and determining the strategies and plans for satisfying these requirements.
7. Orchestrating upgrades to SoS: This activity is related to the actual implementation of the SoS, where the SoS engineering team needs to coordinate different parties that are involved in the SoS development decisions such as SoS owners and sponsors, SoS managers, constituent-system owners and managers, and constituent-system engineers to conduct the implementation activities and to perform the constituent-system integration.

Figure 3-1 shows the relationships between the SoS engineering core elements. The activities are grouped into four categories:

- New SoS role: the purpose of this activity is to conduct a new SoS development activities or SoS evolution.
- Persistent SoS modelling framework: a modelling framework is required to facilitate the development of the SoS architecture. In this activity the developers define or select the SoS architecture framework that will include the SoS models at different development levels (e.g. operational, functional, behavioural).
- SoS upgrade process: The SoS life cycle typically extends over years. During this period the SoS evolves to consider new requirements and technical changes of the constituent systems. This activity addresses the SoS changes and dynamicity.
- External influences: the core of this activity is to analyse the effect of the external environment on the SoS development and evolution.

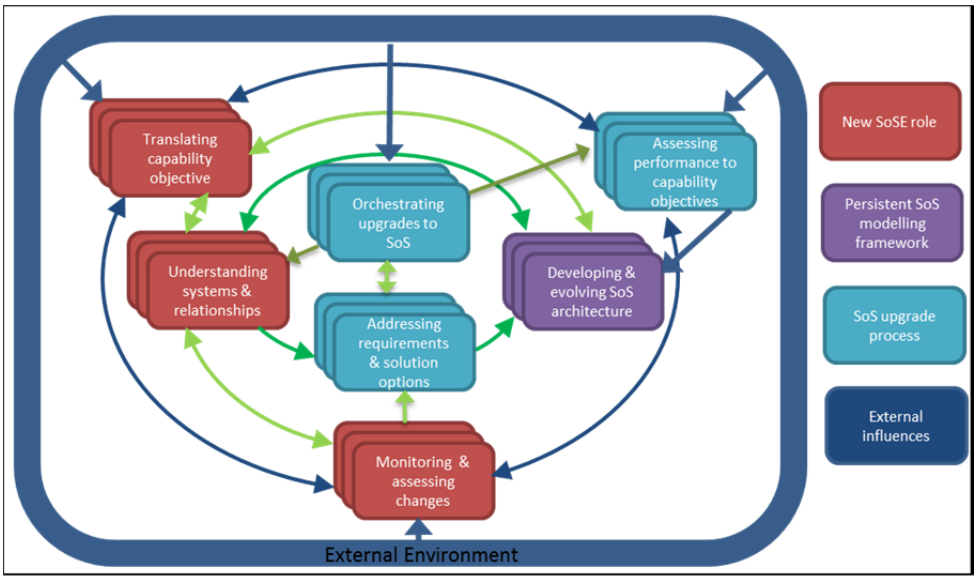

Figure 3-1 SoS engineering core elements [23]

The SoS life cycle is characterised by a continuous process over time. The SoS engineering activities overlap with each other while the constituent systems are under dynamic change and development. Figure 3-2 illustrates the SoS development process with the corresponding constituent-system engineering in a Vee representation.

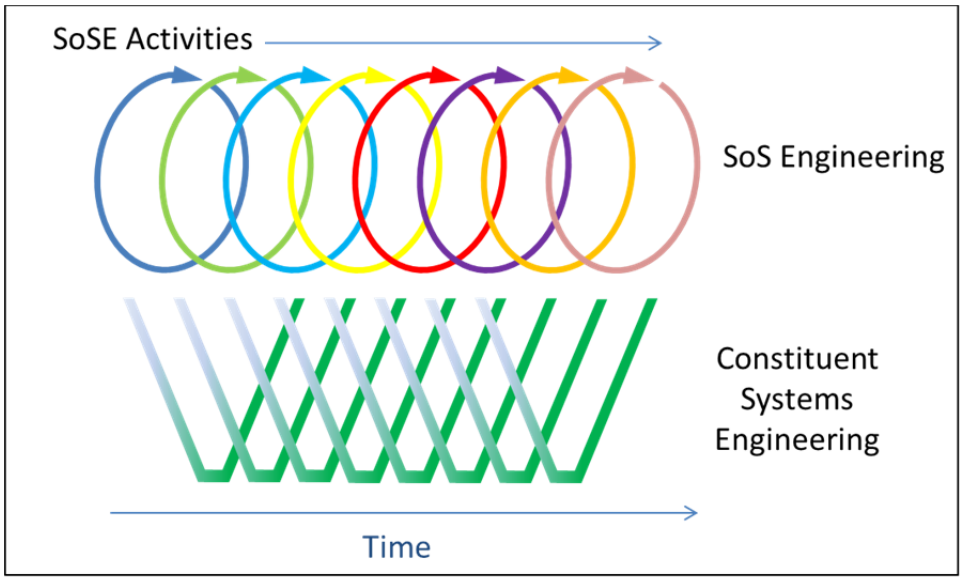

Figure 3-2 SoS engineering activities and constituent-system engineering

3.2.1 System of Systems Engineering vs. System Engineering

The scope of systems engineering is different than the one of SoS engineering. System engineering starts with well-defined requirements and finishes with a system with expected behaviour, a defined management authority, known outputs and inputs. In contrast, SoS have emergent behaviour and typically no centralised management authority. At the same time the SoS environment can change according to the evolution of the SoS and its constituent systems which leads to variable requirements. However, the SoS engineering process may include some of the procedures and steps adopted from system engineering.

In Table 3 Norman and Michael [55] differentiate between Traditional System Engineering (TSE) and SoS engineering in their work at the Air and Space Operation Center (ASOC). They differentiate the system properties when applying either TSE or SoS engineering.

The products of TSE are monolithic systems that have defined boundaries and are realised to exactly meet their pre-defined specifications. SoS engineering products are large-scale systems. SoS engineering considers the dynamic evolution of the SoS specification and capabilities and establishes a continuous development process.

Another comparison done by [56] is presented in Table 4 This comparison is based on the main drivers of system engineering and SoS engineering. The system engineering focuses on a single system that solves a defined problem and has a fixed structure. In system engineering the system structure is considered as hierarchical, while in SoS engineering it is viewed as a network of constituent systems.

Table 3 Differences of enterprises and products in TSE and SoS

Traditional System Engineering	System of System Engineering
Products are reproducible.	No two enterprises are alike.
Products are realised to meet pre-conceived specifications.	Enterprises continually evolve and increase their own complexity.
Products have well-defined boundaries.	Enterprises have ambiguous boundaries.
Unwanted possibilities are removed during the realisation of products.	New possibilities are constantly assessed for utility and feasibility in the evolution of an enterprise.
External agents integrate products.	Enterprises are self-integrating and re-integrating.
Development always ends for each instance of the product realisation.	Enterprise development never ends and enterprises evolve.
Product development ends when unwanted possibilities are removed and sources of internal friction (e.g. differing interpretation of the same inputs) are removed.	Enterprise depends on both SoS external cooperation and internal coordination between the constituent systems to stimulate their evolution.

Table 4 Analysis of main drivers in system engineering and SoS engineering [56]

	System Engineering	**System of Systems Engineering**
Focus	Single system	Integration of constituent systems
Problems	Defined	Emergent
Boundaries	Static	Dynamic
Structure	Hierarchical	Network
Goals	Unitary	Pluralistic
Approach	Process	Methodology
Timeframe	System life cycle	Continuous
Centric	Platform	Network

3.3 Modelling Frameworks

The ISO/IEC 42010 defines the architecture as the 'fundamental organisation of a system embodied in its components, their relationships to each other and the environment, and the principles guiding its design and evolution' [57]. In SoS the architecture represents a description of the constituent systems and their interaction. The architecture is used for multiple purposes:

1. To ensure the same understanding of the SoS concepts between the SoS stakeholders.
2. Facilitate communication between the developers and the owners of the SoS.
3. Documentation of the SoS development process.
4. Architecture analysis (e.g., simulation, model checking) for validation and verification.

According to the purposes of the architecting process the abstraction level of the models is determined. For example, using the architecture for simulation requires a behavioural model of the constituent systems. Applying a static optimisation with respect to the costs of the system architecture requires the modelling of the constituent-system attributes and properties.

Architecture frameworks are defined in ISO/IEC 42010 [57] as follows: 'An architecture framework establishes a common practice for creating, interpreting, analysing and using architecture descriptions within a particular domain of application or stakeholder community'. The architecture framework is a standard approach for describing the system architecture and its contents. The goals for the architecture framework are as follows [58]:

- codify best practices for architecture description by connecting various parts of the architecture model under one framework.
- deliver the required architecture information in an appropriate format to the customers.

- facilitate comparative evaluation of architectures and their properties.
- improve productivity of developers by expressing the designed elements in a standardized language.
- improve interoperability of systems by following a standard way of describing elements.

An architecture framework is used to identify the system stakeholders and their concerns in the systems such as cost, reliability and functionality. The architecture framework consists of architecture viewpoints and views. Architecture views contain architecture models that express the architecture of the system of interest from the perspective of a set of system stakeholders with specific concerns. The architecture viewpoint is a set of conventions for constructing, interpreting, using and analysing one type of architecture view [57]. It includes modelling methods, concepts and techniques for building the view. For example, the SoS customer is concerned with the operational scenarios of the SoS and its high level behaviour. The customer is not interested in the technical details of the constituent systems. Therefore, the operational view describes the operational behaviour and depicts the SoS based on the customer concerns, while the system view describes the constituent-system interactions and expresses the SoS based on developer concerns. The design rules of an architecture framework enforce relations between architecture view elements or between the architecture views.

Figure 3-3 describes the relation between the architecture framework and the stakeholders. The architecture framework addresses the concerns of the stakeholders and the architecture viewpoint frames these concerns.

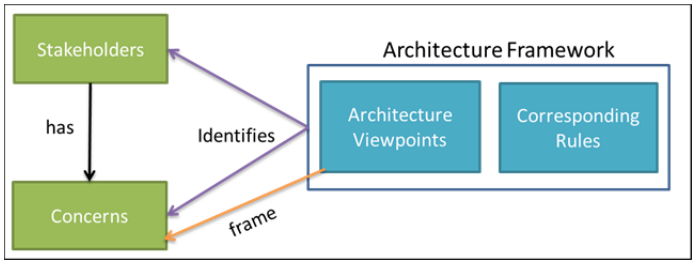

Figure 3-3 Architecture frameworke concept

In the last decades, several architecture frameworks emerged in different sectors. In this thesis we consider the most important SoS architecture frameworks: U.S Department of Defence Architecture Framework (DoDAF), the Ministry of Defence Architecture Framework (MODAF), the NATO Architecture Framework (NAF), and The Open Group Architecture Framework (TOGAF).

3.3.1 DoDAF

The US Department of Defence Architecture Framework [59] was originally developed to support Command, Control, Communication, Computing, Intelligence, Surveillance, and Reconnaissance (C4ISR) systems [58] and to improve their interoperability. Nowadays the framework supports various domains of complex systems and SoS.

DoDAF depicts the system architecture from different perspectives organised in architecture views as shown in Figure 3-4. It serves as a catalogue of viewpoints, which express the system architecture using a set of diagrams at different levels of abstraction. In addition, DoDAF serves as a guide for the development of integrated architectures. The overall viewpoint (called ‘all viewpoint’) provides general information about the SoS and the mod-

els. The capability viewpoint shows the constituent-system capabilities and their relations. Based on these capabilities and interactions, the data and information viewpoint depicts the information exchange between the constituent systems. The operational viewpoint expresses the SoS with respect to the customer concerns as operational scenarios. Project viewpoints are the management views of the SoS. All standards that are required during the SoS design process are documented in the standards viewpoint. The services viewpoints depict the services offered by the SoS or between the constituent systems. Finally the system viewpoint presents the architecture of the SoS with the interactions and connections between the constituent systems.

DoDAF defines and conceptualizes in its meta-model the architecture elements that are used in the architecture descriptions. The meta-model also identifies the dependencies and connections between these elements and the mapping between different architecture views.

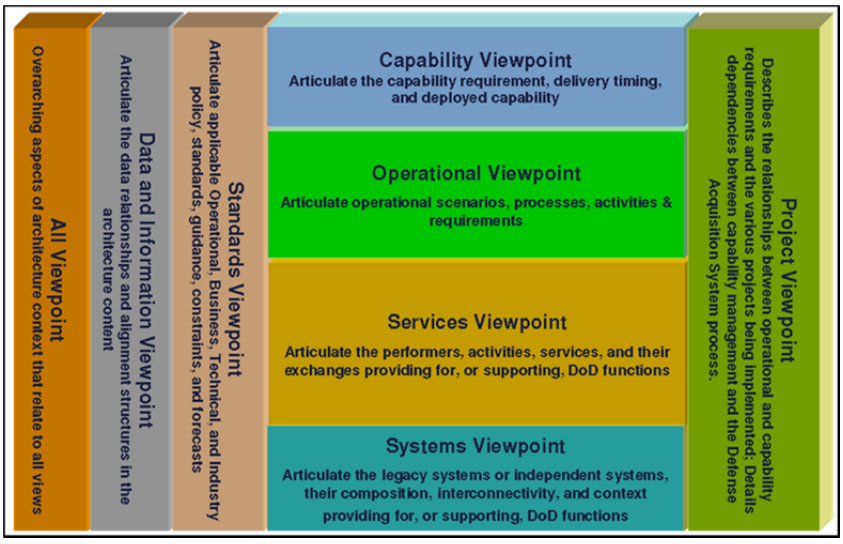

Figure 3-4 DoDAF viewpoints [59]

All Viewpoint (AV): The AV contains text information about the model. It expresses the model information in two views: The overview and summary information view (AV-1) defines the scope of the architecture, while the integrated dictionary view (AV-2) includes an integrated dictionary with architecture metadata and references.

Capability Viewpoint (CV): A capability is a required ability to perform a set of activities under defined conditions to achieve desired effects. The CV addresses the capability taxonomy and capability evolution of constituent systems and SoS architectures. The CV introduces a high-level description of the emerging capabilities of a SoS architecture in the vision view (CV-1). The same view presents the scope, the long term strategic planning, and the evolution of the SoS capability. The constituent-system capabilities are organised in a hierarchical representation that shows their relations and taxonomy. The 'capability taxonomy view' (CV-2) maps these capabilities to their constituent systems and to the overall SoS capabilities.

The SoS development process extends over time and is part of a continuous development. The SoS capabilities change over the time with the development and engagement of new constituent systems. The 'capability phasing' view (CV-3) provides a detailed plan for the capability achievements over time, including capability requirements and specifications.

The capability of the SoS emerges from the integration and interoperation of the constituent systems and their capabilities. When a constituent system joins the SoS, it establishes relations and dependencies between its capabilities and the capabilities of other constituent systems. The ‘capability dependencies‘ view (CV-4) enables the SoS developers to model the dependencies between the constituent-system capabilities, as well as between the SoS capability and the constituent-system capabilities.

The capabilities in SoS are mapped to three elements of the SoS architecture:

- *Performers* are the constituent systems that provide the capabilities
- *Operational activities* are defined as sets of tasks that transform inputs to outputs based on the state and the progression of time
- *Constituent-system services* are contributed by the constituent systems to the SoS

The ‘capability to organisational development mapping’ view (CV-5) models the mapping between the capabilities and their performers, while the operational activities are mapped to the required capabilities that enable the SoS to perform these activities in the ‘capability to operational activities’ view (CV-6). Finally the capabilities are mapped in the ‘capability to service mapping’ view (CV-7) to the constituent-system services that are enabled by these capabilities.

Data and Information Viewpoint (DIV): the design specifications must identify the required data and information that enable the constituent systems to contribute their services to the SoS. The data and information viewpoint describes the information requirements and rules that are used as constraints in the organisations and business activities that are involved in the SoS. In the ‘conceptual data model’ view (DIV-1) high-level data and information concepts and their relations are represented. This view answers the questions about what data is available, what data and information is needed for different SoS stakeholders, and what are the dependencies between different types of information.

The data requirements and structural business process rules are introduced in the ‘logical data model view’ (DIV-2). This view maps the conceptual data level to the physical level. The detailed formats and structures of the exchanged data and information and their specifications are modelled in the ‘physical data model’ (DIV3) view.

Operational Viewpoint (OV): this viewpoint presents a detailed description of the SoS operations and shows how the system operations are carried out. Figure 3-5 shows an example of one of the operational views, namely the ‘high level operational concepts’ view (OV-1) for an emergency response SoS [7].

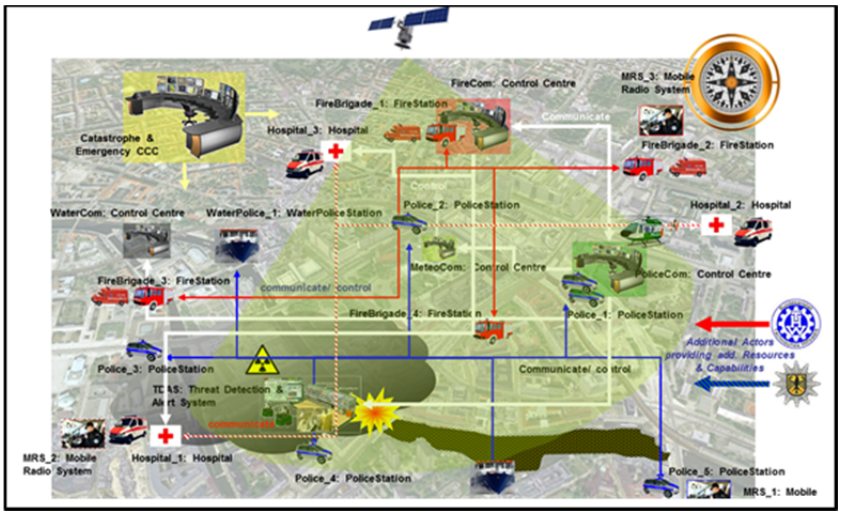

Figure 3-5 DoDAF operational view of emergency response SoS

As shown in the figure, OV-1 is a graphical representation of the SoS operations. It provides a high-level description of the SoS services and functionality. The figure illustrates the different performers and organisations involved in the SoS such as police stations, fire brigades, and their interactions at a high level of abstraction. It also provides an overview of the emergency response operations.

In the 'operational resource flow description' view (OV-2) the flow of the resources (e.g. information, funding, and materials) between the performers of operational activities is depicted. The view describes the collaboration needs between these performers. These actual interactions between the performers of operational activities are presented in a matrix representation in the 'operational resource flow matrix' view (OV-3).

The structure of the SoS, the organisations involved in the SoS and their relationships are modelled from a management perspective in the 'organisational relationships chart' view (OV-4).

The operation activities views OV-5 and OV-6 are the most important views for understanding the SoS operations, which are implemented by constituent systems and their interactions. OV-5 and OV-6 are composed of the following three views:

- 'Operational activity decomposition tree' view (OV-5a): This view provides a hierarchical organisation of the operational activities and capabilities.
- 'Operational activity model' view (OV-5b): This view models the scenarios of operational activities and illustrates the interactions between these activities, the information flow, inputs and outputs.
- 'Operational rules model' view (OV-6a): This view is another description of the operational activities. It contains the business rules that constrain the operations.

Figure 3-6 depicts an example of the OV-5b view for the emergency response SoS. It is a representation of a house fire scenario from an operational perspective. The diagram shows the flow of operational activities and the interaction of these activities. It also maps the operational activities to their performers. In this example, the operational activities flow starts with the notification of a fire initiated by an external person. The command and control center collects the available information about the emergency case, and based on the collected information it decides whether it is an emergency case or not. In case of an emergency case, the command and control

center starts to analyse the situation to define the first response actions. This activity is followed by issuing a Common Relevant Operation Picture (CROP) report that contains details about the emergency case and the required response actions. The CROP is distributed to the police and fire headquarters to perform the required actions and update the CROP report.

The operational scenarios are important in the SoS development process, where these scenarios are modelled at the level of the customer domain and discussed with the customer to ensure the fulfilment of the customer requirements before shifting to the technical domain.

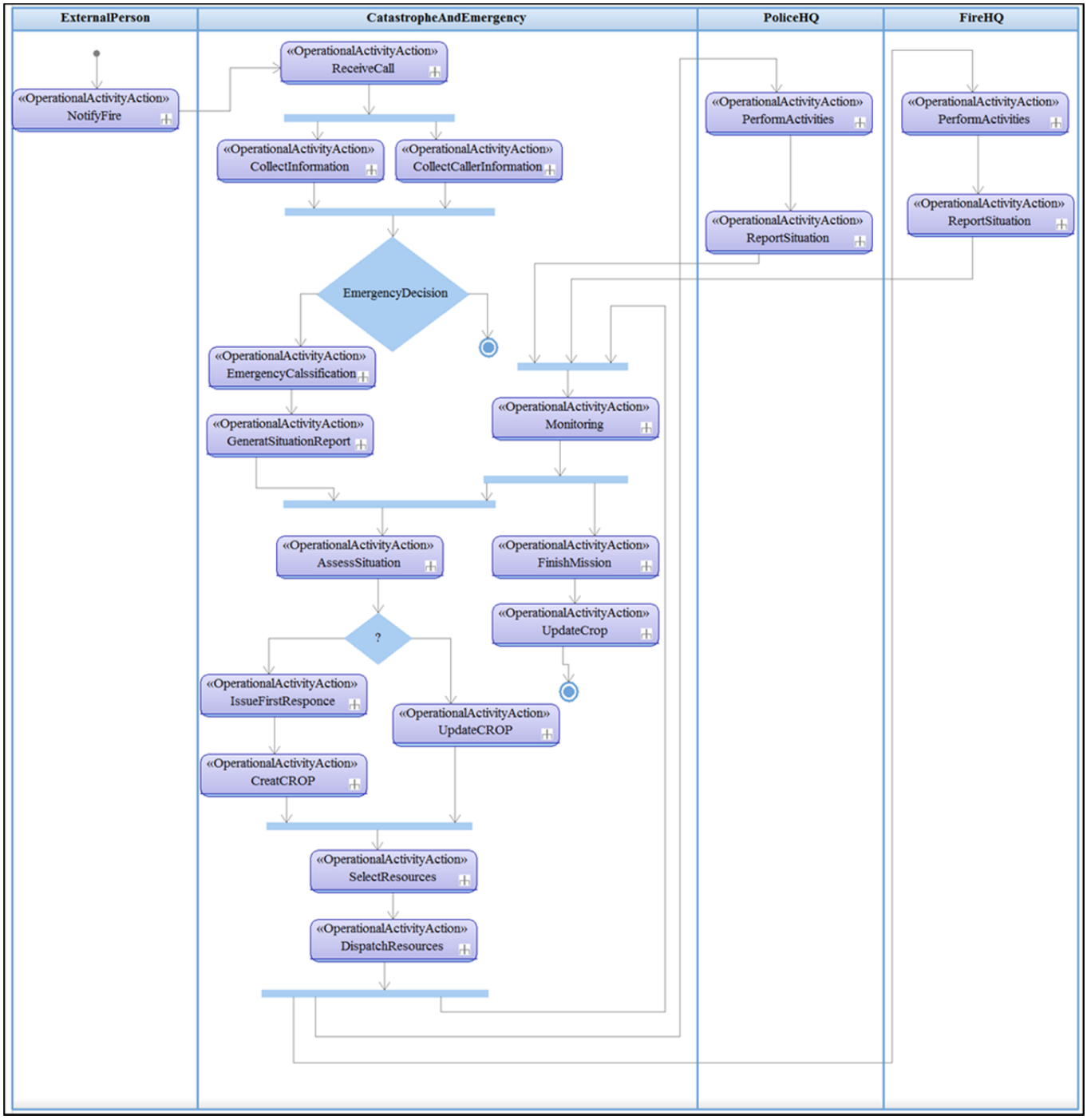

Figure 3-6 Operational activity model of a house fire scenario

The OV-6 views address the operational activities in two detailed views that describe the behaviour of the operational activities. These views are:

- 'Operational rules model' view (OV-6a): the OV-6a identifies the business rules that constrain the operational activity.
- 'Operational state transition description' view (OV-6b): The view shows the operational activities in response to events.
- 'Operational event trace description' view (OV-6c): This view depicts the sequence of events that occur during the implementation of the operational activity.

Project Viewpoint: This viewpoint is the management perspective of the SoS. The SoS development and evolution are considered as a project and the view describes how the project contributes to the delivery of the SoS architecture capabilities. It considers the organisations involved in the SoS development. The 'project portfolio relationships' view (PV-1) identifies these organisations and their contribution to the project. PV-1 contains the organisational structures needed to manage the project portfolio.

The project planning and the time line are introduced in the 'project time-lines' view (PV-2). The view contains the planned milestones and the activities of the SoS development and the evolution process over time.

The last project view is the 'projects to capability mapping' view (PV-3). This view addresses the relationship between the project and the SoS capabilities. It shows how the considered project helps in achieving the required emerging capability of the SoS.

Services Viewpoint: The SoS delivers services to the environment and other systems according to the specifications. It also uses the services provided by its constituent systems. The Services Viewpoint (SvcV) models the services offered by the SoS and the constituent systems as well as their relations. The first view is the 'services context description' view (SvcV-1). The emerging services of the SoS depend on the integration of the constituent-system services and their interactions. The 'services resource flow description' view (SvcV-2) depicts the resource flow and the dependencies between the services and illustrates the required data exchange to enable the constituent systems and the SoS to provide their services.

The SoS services are dependent on the constituent systems. In the 'system services matrix' view (SvcV-3a) theses dependencies and the relationships between the constituent systems and the services are explained. In the 'services-service matrix' view (SvcV-3b) the dependencies and the relationships are shown in a matrix representation.

The services are enabled by the functions of the constituent systems. The 'services functionality description' view (SvcV-4) maps the constituent-system functions to the enabled services, while in the 'operational activity to service traceability matrix' view (SvcV-5) the services are mapped to the operational activities.

Another concern of the services is the relationship between the services and the resource flow, and how these resources are exchanged to provide the requested services. The 'services resource flow matrix' view (Svc-6) provides a detailed representation of the resource flow between the services based on the SvcV-2 contents.

Measurements of the SoS services are performed in order to evaluate the SoS performance. The methods and the metrics for the SoS-service evaluation are denoted by the 'services measure matrix' view (SvcV-7).

In the services viewpoint there are two views that model the evolution of the SoS services. The first view is the 'services evolution description' view (SvcV-8). It depicts the evolution of services in the SoS over the SoS lifecycle. The second view is the 'services technology & skills forecast' view (SvcV-9). This view is a time-framed plan for the expected technologies and services that will be available and considered in the future.

The last three views of the services viewpoint describe the service functionality:

- 'Services rules model' view (SvcV-10a): This view contains the constraints imposed by the system design or the implementation on the system functionality, thus affecting the offered services.
- 'Services state transition' view (SvcV-10b): This view depicts the service response to events.
- 'Services event-trace description' view (SvcV-10c): the view defines the service refinements of critical sequences of events that were previously defined in the operational viewpoint.

Standards Viewpoint: This viewpoint captures the standards and rules that are used to organise the representation of the architecture and the documentation. It is separated into two views: The first view is called 'standards profile view' (StdV-1) and lists all used standards that are applied on the architecture. The second view is the 'standard forecast' view (StdV-2) containing the emerging standards and their impact on the architecture elements.

Systems Viewpoint: The systems viewpoint is the main viewpoint for describing the SoS architecture. It illustrates the constituent systems and their connections, interactions, functions, behaviour and relation to the SoS.

Figure 3-7 shows an example of the systems viewpoint, namely the 'systems interface description' (SV-1) for the emergency response SoS. As shown in the figure, the view identifies the constituent systems, their interactions, and their interfaces. The connections between the constituent systems represent the resource flow and the dependencies between the constituent systems. This view represents the basis for the SoS simulation as it contains the SoS architecture.

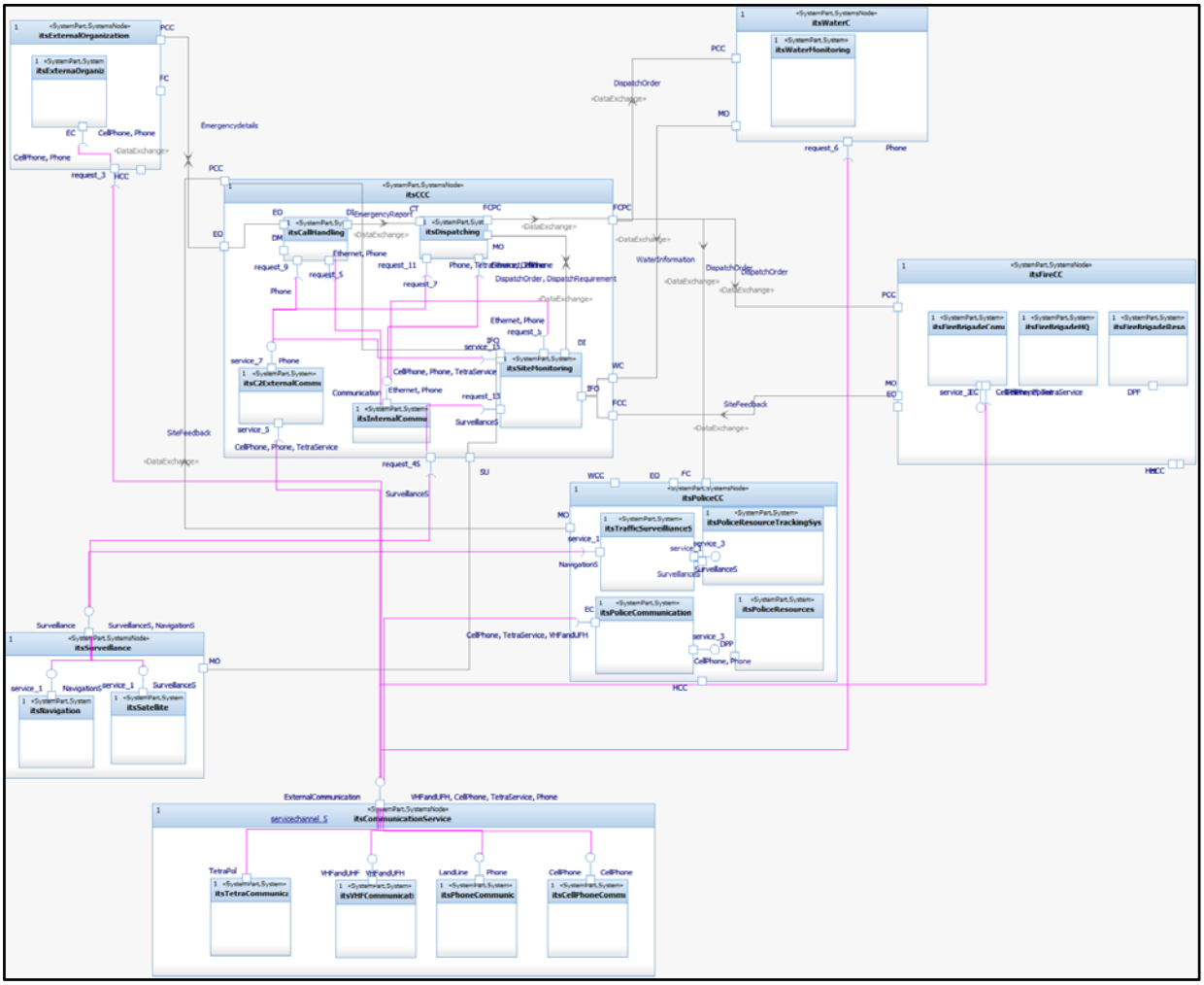

Figure 3-7 System view (SV-1) for the emergency response SoS

In the previous viewpoints, the resource flow was modelled between the operational activities, between the services or between the functions. The resource flow between the constituent systems is also required in the SoS design to define the dependencies between the constituent systems, the required interfaces, and the rules for the resource exchange. The 'systems resource flow description view (SV-2) defines the resource flow and the data connections between constituent systems and the 'systems resource flow matrix' (SV-6) presents detailed information about the resource flow between the constituent systems using a matrix representation. The relationships between the constituent systems are also presented in details in a matrix representation using the 'systems-systems matrix' view (SV-3).

The functions of the constituent systems implement the SoS operational activities. The functionality of the constituent systems and the relationship between the functions and operational activities are organised in the SV-4 and the SV-5a views. As illustrated in Figure 3-8, the 'systems functionality description' view (SV-4) identifies the SoS and the constituent-system functions as well as the functional flow. The 'operational activity to systems function traceability matrix' view (SV-5a) maps the operational activities to the functions that implement these activities.

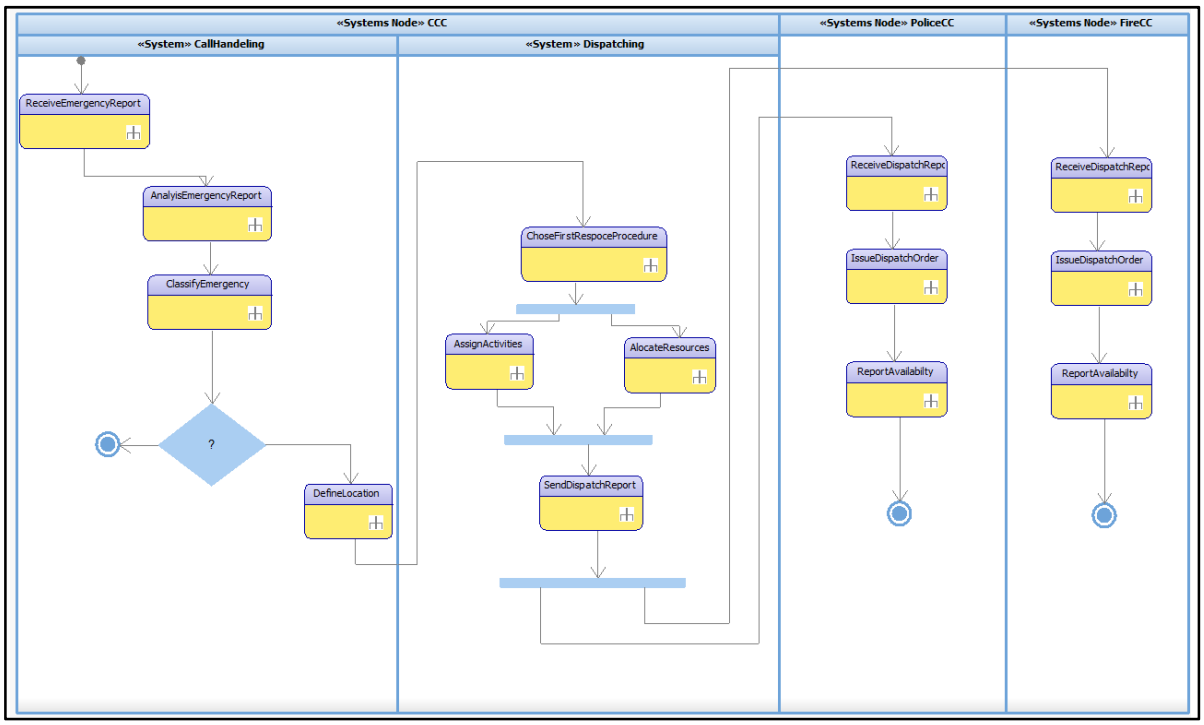

Figure 3-8 System functionality view of the emergency response SoS

The constituent systems are also mapped to the SoS operational activities and to their capabilities. The 'operational activity to systems traceability matrix' view (SV-5b) allows the modeller to perform the mapping and to show the result in a matrix representation.

The SoS engineers keep evaluating the contributions of the constituent systems to the overall SoS capabilities and operations in order to identify possible service evolutions and SoS capability development. The first step towards the evaluation process is to define the metrics for the constituent-system performance. The 'system measure matrix view' (SV-7) contains time-framed metrics for the constituent systems, while the constituent-system evolutions and the dynamicity of the SoS are depicted in the 'systems evolution description' view (SV-8). The SoS also has plans for future developments and strategic plans for the adaptability to new technologies. These plans are presented in the 'system technology and skills forecast' view (SV-9), which contains plans for the incorporation of technologies and developments that are expected to be available in the future.

The last three views are important for modelling the constituent-system behaviour. In combination with the SoS architecture (SV-1), the modeller is able to simulate and analyse the SoS. These three views are:

- 'Systems rules model' view (SV-10a): list the constraints of the constituent-system functionality
- 'Systems state transition description' view (SV-10b): a state diagram representation of the constituent-system behaviour
- 'Systems event-trace description' view (SV-10c): depicts the system refinements for critical sequences of events that were previously defined in the operational activity viewpoint.

3.3.2 MODAF

The Ministry of Defence Architecture Framework (MODAF) [60] was developed for the United Kingdom Ministry of Defence based on version 1 of the DoDAF framework. The MODAF framework added a few modifications to DoDAF and most of these modifications were included in the last version of DoDAF. The differences are based on the terminology and some architecture viewpoints were added.

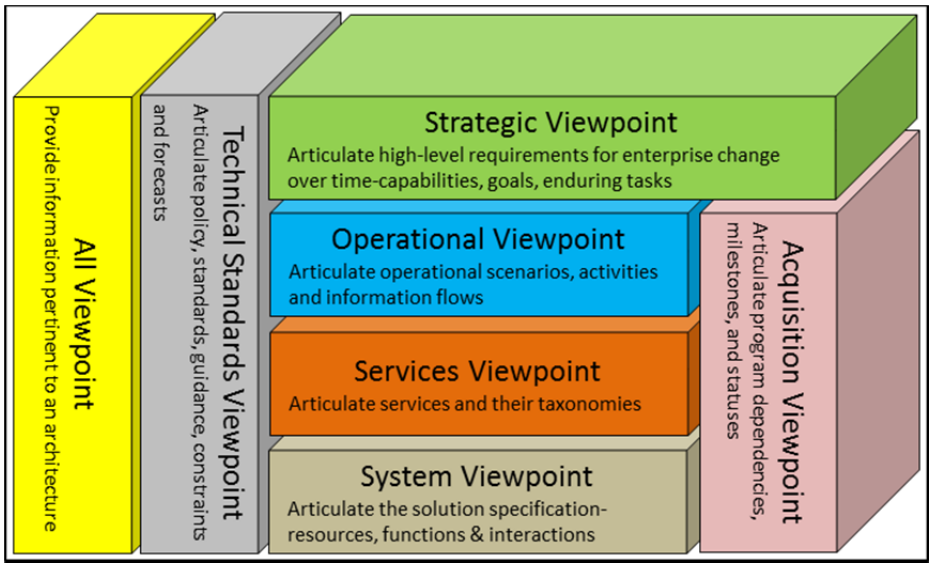

Figure 3-9 MODAF architecture viewpoints

Figure 3-9 displays the MODAF architecture viewpoints. The main differences between DoDAF and MODAF are as follows:

1- Terminology: The terminology was modified to comply with the ISO 42010 standard.

2- Strategic Viewpoint (StV): The capabilities views were not included in the earlier versions of DoDAF. The strategic viewpoint added 6 views that support the high level planning and the concerns of capability managers. The first view, i.e., 'enterprise vision view' (StV-1), presents the vision of the SoS capability. In the 'capability taxonomy view' (StV-2) the SoS capabilities are identified in more details with capability's requirements and a gap analysis. The 'capability phasing' view describes the capability planning and the capability integration planning. The capability management and dependencies are presented in the 'capability dependencies' view (StV-4) and in the 'capability to organisation deployment mapping' view (StV-5). The latter view analyses the SoS capability with respect to deployment and integration. Lastly the 'operational activity to capability mapping' view (StV-6) maps the operational activities to the SoS capability.

3- Acquisition Viewpoint (AcV): This viewpoint was added by MODAF to support the acquisition programs in the military operations. It is similar to the project viewpoint in DoDAF, which was added in version 2 of DoDAF. The acquisition viewpoint supports the concerns of the project managers. It has two views: the 'acquisition clusters' view (AcV-1) identifies the project organisations and the program management, while the 'program time-lines' view (AcV-2) addresses the project management activities such as project control, risk identification, and project portfolio management for the SoS acquisition.

Most of the MODAF amendments to the DoDAF were included in the last versions of the DoDAF (V2.02) under different viewpoints. In addition to the added viewpoints, MODAF modified also five existing DoDAF views, i.e. OV-7, SV-11, SV-2, and SV-12. The 'information model' (OV-7) specifies the information aspects from the operational domain perspective. In the 'system port specification' view (SV-2), MODAF added more detailed views that depict the connectivity between the constituent systems:

- System port specifications (SV-2a)
- System connectivity description (SV-2b)
- System connectivity (SV-2c)

The 'system data model' (SV-11) is a detailed description of the data structures that are used for the data exchange between the constituent systems. SV-12 presents the connections between the constituent systems and the services through two views: the services provision (SV-12a) and the service composition (SV-12b).

3.3.3 NAF

The NATO Architecture Framework (NAF) [3] serves the NATO strategic planning and terminology. It is an extended framework that adapts views from both DoDAF and MODAF with a modified terminology according to the NATO standards as shown in Figure 3-10.

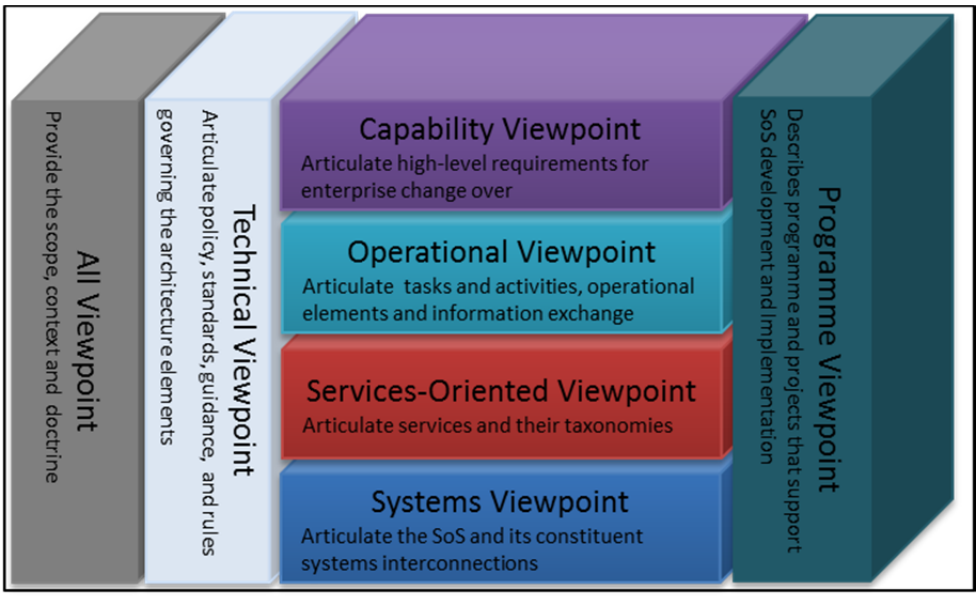

Figure 3-10 NAF architecture viewpoints

There are minor differences between NAF and the previously mentioned architecture frameworks. These differences are in the terminology of particular elements in order to ensure compliance with the NATO standards. The NAF also includes more detailed views on the communication and connection concerns between the constituent systems.

In the overall viewpoint (AV), NAF adds two NAV-3 metadata views: The 'architecture compliance statement' view NAV-3a contains data that certifies the architecture compliance within the NATO standards, and the

'metadata extensions' view NAV-3b describes the architecture deviations according to the standard NAF guidelines.

In addition to the adaptation of the MODAF views, the NAF introduced additional system-connectivity views and a technical view. NAF adds the 'system communication quality requirements' view (SV-2d) that specifies the quality requirements of the communication between the constituent systems. The technical viewpoint of NAF adds the 'technical standard configurations' view (NTV-3), which specifies the standard configurations that are used in the SoS architecture.

3.3.4 TOGAF

The Open Group Architecture Framework (TOGAF) [61] is an enterprise architecture framework and method. It was originally developed as a framework and methodology for the development of technical architectures before it evolved to a framework with methods and tools to support enterprise architectures [62].

TOGAF consists of four parts:

- ***Architecture capability framework***: a set of reference materials identifies the organisation structures, processes, roles, responsibilities, and skills to realize the architecture.
- ***Architecture Development Method (ADM)***: It describes a method for developing and managing the lifecycle of enterprise architectures to meet the business and IT needs based on the TODAF architecture framework elements and assets.
- ***Architecture content framework***: This framework provides a structural model for the enterprise architecture. It considers the enterprise architecture from four perspectives: business, data, application and technology.
- ***Enterprise continuum***: The enterprise architect needs to deal with many related architectures to meet all stakeholders requirements. The enterprise continuum comprises various reference models that describe the evolution and development of these architectures across a wide range from generic to specific ones.

TOGAF presents not just an architecture framework but also an iterative architecture development method as illustrated in Figure 3-11.

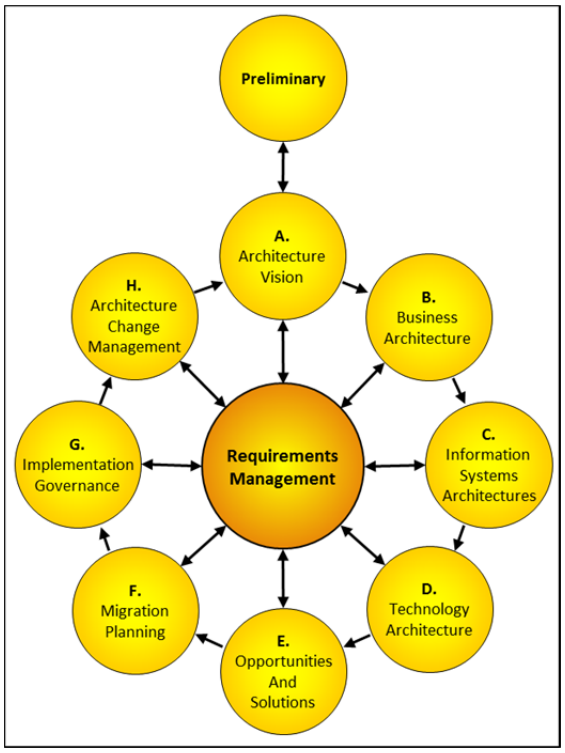

Figure 3-11 TOGAF architecture development method [61]

The development of the enterprise architecture may require:

- Multiple concurrent ADM phases: More than one ADM phase is being implemented in parallel.
- Cycles between the phases: The enterprise architecture may include cyclic paths between different phases.
- Backtracking to previous phases to update information: The output of one of the phases may require updating the information that was provided in previous phases, thus requiring to re-run the development cycle again.

The ADM iterates over the whole development process of the enterprise architecture, between the architecture development phases and within each phase. In each iteration of ADM, a decision must be taken regarding the scope, abstraction level, time frame, and the architecture assets.

The ADM is a generic method for the application of the TOGAF architecture framework. It can be applied for various industrial applications and can be tailored to the developer's needs.

3.3.5 Other Architecture Frameworks

Zachman: it was first introduced by John Zachman in 1987 as a framework for the architecture of information systems [63]. Zachman evolved to an enterprise architecture framework that presents the architecture from various perspectives as shown in Figure 3-12. Zachman presents the system architecture in a structural representation based on the logical connections between different viewpoints.

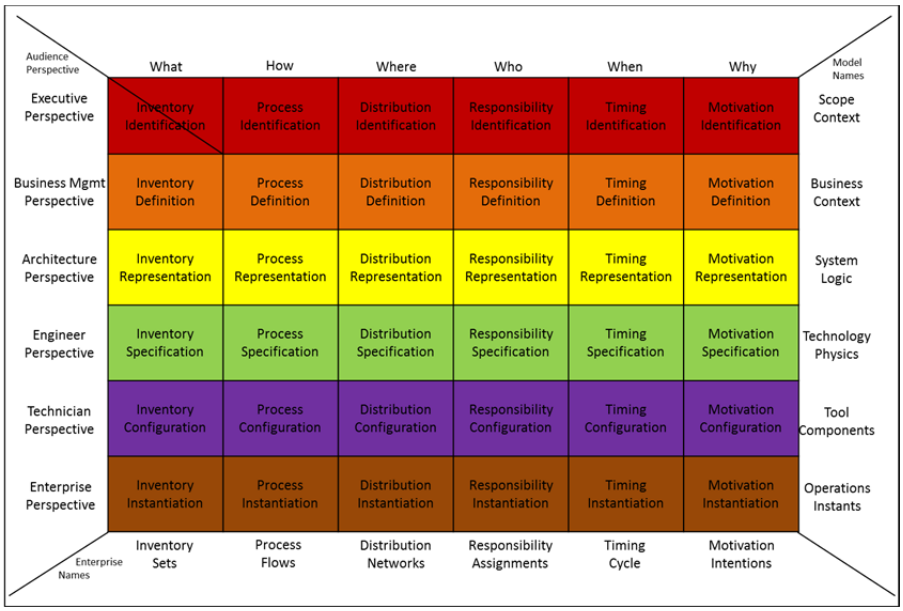

Figure 3-12 Zachman architecture framework

IEEE 1471, ISO 42010: This framework was developed by the IEEE Architecture Planning Group (APG) as an architecture standard for software-intensive systems (e.g. information systems and embedded systems) with support for the definition, analysis, and description of systems architectures. In 2007 IEEE 1471 was adopted by the ISO as ISO 42010.

The standard does not present views or architecture methodologies but provides definitions and terms used in the description of the system architecture as well as the relationship between these terms. It gives a recommended practice of how to use the architecture concepts for creating an architecture description, and describes the role of the system's stakeholders in the creation of the system architecture.

Figure 3-13 depicts the concepts and terms used in the architecture description and the relationships between different parts according to IEEE 42010. The architecture description contains an architecture viewpoint that depicts the stakeholders concerns for the system of interest through architecture views. The architecture description is governed by corresponding rules that define the relations between the architecture elements and their interactions. The architecture view is comprised of architecture models that use multiple notations to describe a specific stakeholder concern.

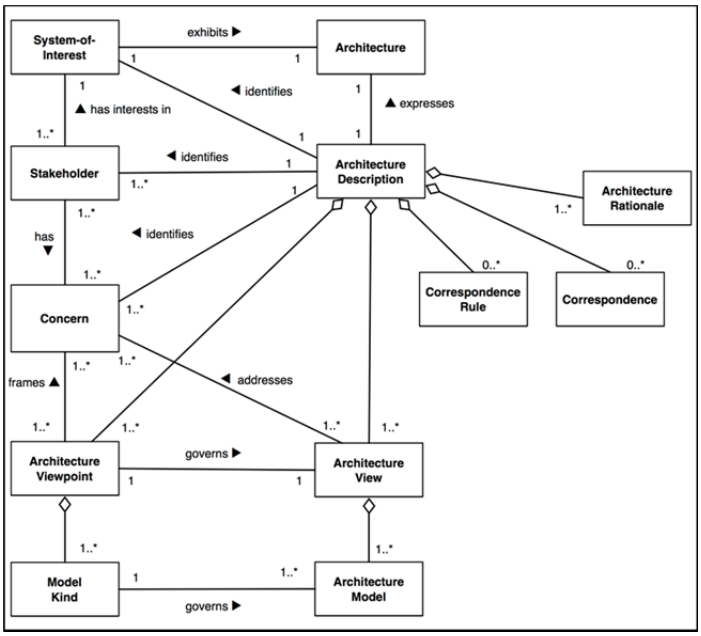

Figure 3-13 Core of the architecture description according to IEEE 1471 [57]

ISO Reference Model for Open Distributed Processing (RM-ODP): Another architecture description standard was developed by the International Organization for Standardization (ISO) as a Reference Model for Open Distributed Processing (RM-ODP) [64]. It consists of four parts [65]: an overview of the reference model, a descriptive model, a perspective model and architectural semantics. The RM-ODP standard provides the following elements:

1- Architecture viewpoints: Architecture viewpoints simplify the description of complex systems by considering the system from different perspectives. Each of the viewpoints reflects one set of design concerns. These viewpoints are enterprise, information, computational, engineering, and technology.

2- Conceptual framework: The conceptual framework supports different aspects related to the distribution and interoperation of distributed systems.

3- Concepts: A set of concepts provides definitions and semantics used in the development of the distributed system architecture.

Unified Architecture Framework (UAE) [66]: The US department of defence recently started working on the development of a unified architecture framework that considers the ISO standards while also adopting the DoDAF, MODAF, and NAF architectures.

3.3.6 Summary

Each of the presented architecture frameworks was developed to support specific stakeholders concerns and particular system types. The decision for selecting one of the architecture frameworks depends on the following aspects:

1- System complexity: In complex systems it is required to distribute the system architecture over multiple related architectures to control and reduce the system design complexity.

2- System stakeholders: The concerns of the system stakeholders define which architecture viewpoints are required to describe the systems. For example, the viewpoints required by the system management with managerial concerns are not the same as the viewpoints required by the development engineers with technical concerns.

3- System type: There are specific architecture frameworks such as Zachman that are useful in supporting software systems and are very complicated to be used in describing a complex SoS with different concerns due to the huge number of required modelling elements and the difficulty of mapping different modelling elements in various viewpoints.

4- Modelling purpose: Some of the framework architectures support views with behaviour description capabilities that enable modellers to perform behaviour analysis through building executable models, while other frameworks just support static analysis.

SoS are complex systems with large stakeholder communities. To describe the SoS development process, the architecture framework must be able to present the system in two dimensions. The SoS dimension encompasses the architecture, capabilities, operations, and services, while the constituent-system dimension is concerned with their services, capabilities, and functions. The DoDAF, MODAF and NAF architecture frameworks are based on military operations and missions. Many military operations are implemented using a SoS. TOGAF is an enterprise architecture framework, which offers general architecture views that depend on an iterative approach. This approach is very complicated for designing complex systems such as SoS in industry where time and cost are limited and a clear methodology is required for reusability and consistency. The other frameworks (i.e., Zachman, IEEE 1471) focus on software systems and become inefficient due to model complexity for describing SoS.

Although the development of NAF and MODAF is based on DoDAF, there are some differences with respect to terminology and semantics. The services and capabilities views are different in theses frameworks. Nevertheless, there is a common agreement on the definition of operational activities, SoS architecture, constituent systems, and functions.

Based on the previous discussion, we use the DoDAF architecture framework in the methodology of this thesis for the following reasons:

1- The viewpoints of the DoDAF architecture framework are sufficient for the industrial concerns of SoS, which exhibit managerial concerns, engineering concerns, and owner concerns.

2- DoDAF supports the logical mapping between the DoDAF views, thereby facilitating the development process and the methodological flow.

3- The DoDAF views support the behavioural description of constituent systems, which is an important foundation for executable models and simulation.

4- A language is available that supports the modelling using DoDAF, namely the Unified Profile for DoDAF and MODAF (UPDM).

5- The new version of DoDAF (V2.0) [59] adopts the enhancements added in MODAF and NAF.

3.4 Modelling Languages

The system architecture is typically expressed using a combination of textual and graphical representations. The textual representation uses the English language and textual modelling languages with standardised expressions. In graphical representations graphical modelling languages are used to express the system architecture in diagrams using symbols and graphical notations.

There are various modelling languages that address different architectural concepts and different domains. For example, in software applications the Unified Modelling Language (UML) is used to facilitate the software development and to express the software functionality. In the mechanical domain, Modelica is used to model the mechanical and electrical functionality of the system.

Modelling languages can be effectively used for supporting various system domains such as enterprise information systems, banking and financial services, telecommunication, defence/aerospace, medical electronics, and distributed web-based services.

3.4.1 UML

The Unified Modelling Language (UML) was developed by the Object Management Group (OMG) group [4] as a general-purpose graphical modelling language. It is used to specify, visualize, construct and document the artefacts of a software system [67]. UML focuses on the conceptual and physical representation of the systems and is not a programming language. It is used to illustrate multiple stages of the system's development. It provides different kinds of diagrams that help in better understanding the system and supports the controlling and maintaining process for the constructed systems. It is an executable language that supports visual modelling tools with four main parts:

- Static structure: this part identifies the components which are important for the system implementation and their relationships.
- Dynamic behaviour: the UML behaviour part depicts the system behaviour over time and the communications links between components that are required to achieve the tasks.
- Development environment: this part of UML identifies the system development environment
- Organisational constructs: the organizational constructs are used to arrange models into different packages in order to facilitate the development process between development teams.

The UML unifies and clearly defines the modelling concepts gathered from different object-oriented methods and it uses well-defined notations and terminology. It is suitable to be used in all stages of the system development with flexibility in transferring information from one stage to another as the same notations and concepts are used. The UML was developed to be flexible in the use for different types of systems, e.g. simple systems, large systems, complex systems, distributed systems. It is also usable for systems implemented in different programming languages and using different run-time platforms.

Purpose

UML is typically used for visualizing, specifying, constructing and documenting [49]:

- Visualizing: the graphical notations convert the ideas and thoughts into visual models and enable the model to be shared, reviewed, improved, and tested.
- Specifying: the UML helps in building precise, unambiguous and complete models using a set of diagrams that address different concerns.
- Constructing: models that are constructed using UML can be directly connected to programming language which gives the ability of generating source code out of the models.
- Documenting: UML addresses the documentation of different parts of the system development process, e.g. system requirements.

Diagrams and views:

UML notations support different types of viewpoints that depict the various concerns of the system through a set of diagrams. The major areas of the system architecture that are supported include the structural viewpoint, dynamic viewpoint and model management [67]. Table 5 shows the UML diagrams organised according to their views and application areas.

UML was constructed to serve as a general-purpose modelling language. In addition to the existing UML notations, UML can be extended for a special domain by introducing a profile extension. A profile is a declaration of a set of extensions to the UML language that will enable the UML to become more application or domain-specific [68]. To define a profile there are three main extensible constructs in UML:

1- Constraints: A constraint is an expression that is used to imply a semantic relationship. It is a textual statement represented informally by enclosing text within curly brackets or formally using specified language like the Object Constraint Language (OCL) [69]. There are three types of constraints:
   - Invariants are used for properties that cannot change
   - Preconditions are connected to the operations and related to the conditions that must be satisfied before the operation is executed.
   - Post-conditions are connected to the operations and related to what will be available after the execution of the operation.

2- Stereotypes: A stereotype defines a new UML element that is constructed by the modeller or it is based on an existing model element by tailoring the UML meta-model.

3- Tagged values are used to add information to an existing model element.

Using the profile extension concept, the OMG developed other modelling languages that are specified for different domains like the System Modelling Language (SysML), the Service Oriented Modelling Language SoaML and others.

Table 5 UML diagrams

Major Area	Diagrams	Usage
Structural	Class diagram	Identification of the concepts in the application domain and internal concepts connected to the implementation of the application.
	Use case diagram	A description of the functionality of the system as perceived by the users.
	Component diagram	Illustrates the dependencies between the system components and the system tasks.
	Deployment diagram	Depicts the way that the system will be physically deployed in the hardware environment.
Dynamic	State chart diagram	Contains multiple states that model the system during a period of time under certain conditions.
	Activity diagram	The dynamic view of the system that shows the flow of activities and the operational activities involved in performing the system tasks.
	Sequence diagram	Models the sequences of message exchanges among components that implement the behaviour of the system.
	Collaboration diagram	A structural organisation of the objects that send and receive the messages.
Model management	Class diagram	Illustrates the organisation of the model.

3.4.2 SysML

System Modelling Language (SysML) is a general-purpose graphical modelling language that supports the analysis, specification, design, verification, and validation of complex systems [70]. The SysML is based on UML with additional extensions to make it more suitable for system engineering, e.g. using model blocks rather than model classes as in UML. It uses part of the UML and defines its own extensions to UML as shown in Figure 3-14.

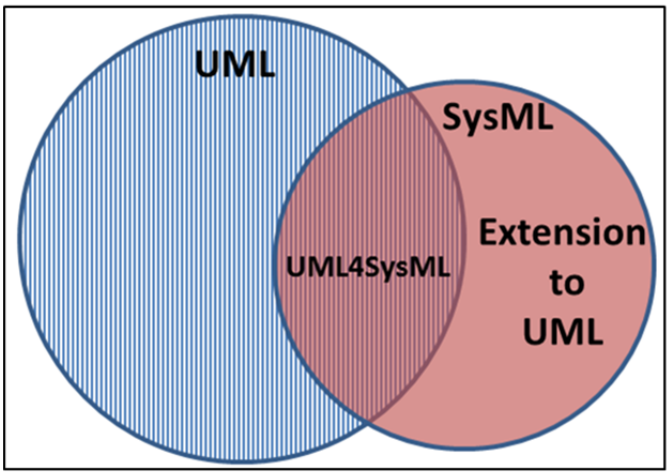

Figure 3-14 SysML profile construction

Purpose

The OMG constructed the SysML to support the system engineering process in enhancing the system quality by improving the ability to exchange system engineering information among system stakeholders and different tools [71].

SysML is typically used for [47]:

- capturing and analysing the system requirements and establishing traceability of the system requirements and system components requirements among the system design activities.
- development of system architectures by identifying the system structure, components and their interactions
- applying engineering methods for analysing trade-offs between system architecture candidates.
- implementing system verification processes on system architectures.

Diagrams and views:

The SysML is constructed out of nine diagrams that are required for system modelling. They are grouped in three categories: structure, dynamic, and requirements diagrams. Table 6 illustrates the SysML diagrams and their categories.

The SysML modelling approach is based on a package and a block definition. A package represents a container for the model elements. A block is a fundamental unit in the SysML system architecture. It is a structural element that encompasses software, hardware, data, processes, humans and elements.

SysML enables the modeller to arrange the model elements in different structures and to do a mapping between the elements. This feature allows developers to deploy elements in different architecture structures.

Table 6 SysML diagrams

Major Area	Diagrams	Usage
Structural	Block definition diagram	Defines the system hierarchy and describes the system's components.
	Internal block diagram	Describes the internal structure of a block and shows the connection and properties of its parts.
	Parametric diagram	Depicts constrains on the systems' properties through defining systems of equations that constrains the properties of the blocks.
Dynamic	State chart diagram	Portrays the different states of a block and their transitions in response to events during the block life cycle
	Activity diagram	Used to describe the flow of inputs, outputs, and control among the actions
	Sequence diagram	Depicts the interaction between the structural elements of the blocks as a sequence of message exchanges
	Use case diagram	Illustrates the relations between the system and its users
Requirements	Requirements diagram	Used to show the requirements of the system, their hierarchy and relationships.

3.4.3 SoaML

The Service Oriented Architecture Modelling Language (SoaML) [72] is a UML-based profile that combines service-oriented business processes with a technical architecture to enhance the agility, collaboration and efficiency of a business enterprise. It serves the business community and organisations by describing how the systems, organisations and people work together to achieve business goals [73].

Purpose

SoaML is used to support the modelling of services at different levels, e.g. services within an enterprise and services at the level of an SoS. It supports the following capabilities [72]:

- Service identification: The SoaML enables the modeller to identify the services provided by the systems with their dependencies and requirements.

- Service specification: Using SoaML, the modellers can specify the services by defining their functional capabilities, their used protocols and rules, the information exchange between consumers and providers, and the expectation of the consumers with respect to the service capabilities.
- Provision and use of service policies: Modellers are able to add service policies to the model.
- Classification schemes: SoaML provides the ability to define classification schemes that serve the architecture, organisational and physical partitioning schemes and constraints.

Diagrams and views:

The SoaML profile extension has no specified diagrams like SysML. The SoaML provides support for specifying the services using other modelling languages like UML or SysML. It introduces four main terminology definitions:

- Participants: a specific component that provides or uses a service.
- Ports: the interface where a component provides or consumes a service.
- Service description: specification of the service interaction by describing how participants interact to provide or consume a service.
- Capabilities: the SoaML specifies the capabilities that are required by participants to provide a service.

SoaML uses different approaches for the service description, namely simple interfaces, service interfaces and service-contract interfaces. The simple interface approach is a one-way interaction modelled using UML interfaces, whereas the service interface involves a bi-direction interaction. In the service-contract the provided or used service specifications are defined in the port in a contract format between the service provider and the service user.

3.4.4 UPDM

The Unified Profile for DoDAF and MODAF (UPDM) [4] was created to serve the modelling within the DoDAF and MODAF architectures in the SoS field. UPDM is a profile extension based on UML, SysML and SoaML.

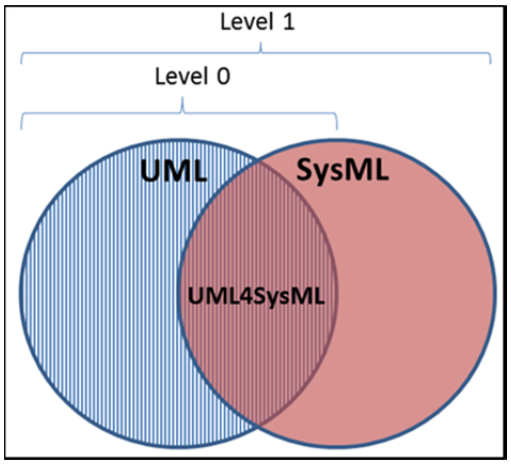

Figure 3-15 UPDM compliance levels

As show in Figure 3-15 UPDM is constructed out of two compliance levels: level 0 and level 1. Level 0 is an extension of the UML, while importing several SoaML stereotypes as illustrated in Figure 3-16. Level 1 includes Level 0 and imports the SysML sub-profiles while defining constraints that pair together the application of SysML and UPDM stereotypes.

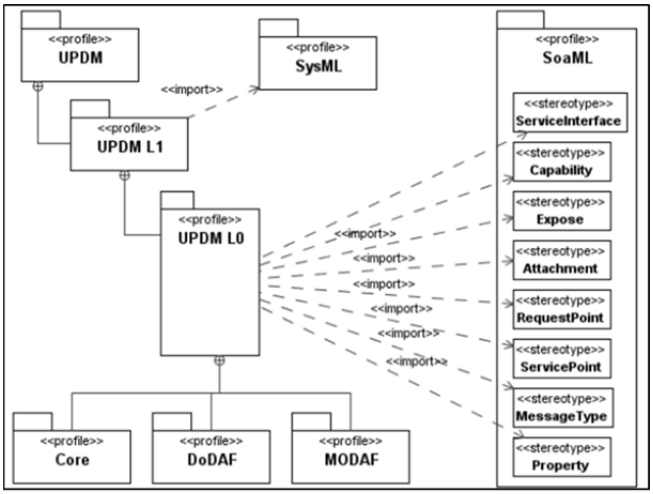

Figure 3-16 UPDM level 0 and level 1 [4]

Purpose

UPDM extends the UML and SysML with specific stereotypes that facilitate modelling using modelling frameworks. It supports:

- Modelling of a wide range of complex systems (i.e. software, hardware, organisational).
- SoS modelling with a consistent architecture under different levels of abstraction to enhance the quality, productivity, and effectiveness of SoS architecture modelling
- Analysis, specification, design and verification of complex systems by providing the ability to build service oriented architectures.
- Architecture information exchange among UML-based tools.
- Integration between SoS modelling and constituent-system modelling.
- Interoperability and communications between SoS stakeholders.
- Different tool implementations and semantics.

Diagrams and views:

UPDM defines semantics and terminologies that support the DoDAF and MODAF architecture frameworks. Its profile extensions and stereotypes are based on the terminology of these frameworks. Figure 3-17 shows the architecture views that are covered by UPDM 2.0. In addition to the supported viewpoints of DoDAF and MODAF, the UPDM enables the modellers to define their own custom viewpoints.

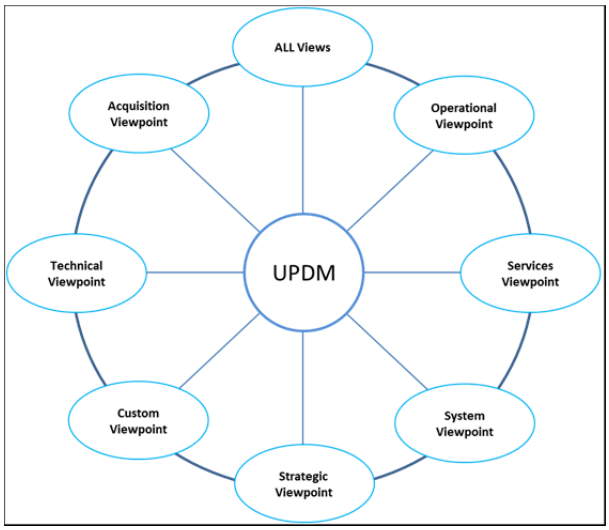

Figure 3-17 UPDM viewpoints

3.4.5 Other Modelling languages

Architecture Analysis and Design Language (AADL) [74]**:** AADL is an international standard that was developed by the Society of Automotive Engineers (SAE). It provides modelling concepts for describing and analysing real-time embedded systems and SoS with hardware and software components. AADL describes a real-time embedded system as a set of components and shows the interactions between these components through interfaces. It also supports the mapping from the software to the computational hardware units.

Modelica: Modelica is an object-oriented modelling language for heterogeneous systems (i.e., mechanical, electrical) based on a physical perspective. It was developed by the Modelica association in 1996. Modelica is a textual modelling language that identifies the components and describes their interactions. It uses mathematical descriptions such as differential, algebraic and discrete equations. Modelica models are used for simulation and analysis.

3.4.6 Summary

Modelling languages can be distinguished based on the modelling purpose. There is no general language that covers all required viewpoints of a SoS. This can be achieved by extending the existing modelling languages or by integrating the capabilities of multiple modelling languages.

Modelling SoS is challenged by multiple aspects. Due to the diversity of the involved stakeholders in the SoS development process, it is required to represent the model in multiple viewpoints and to map these viewpoints to each other. It is also needed to express the SoS at different levels of abstraction and provide links between these levels.

The modelling process is also challenged by the SoS complexity with numerous heterogeneous constituent-system models and interactions. These constituent systems have different owners and behavioural models that are developed using different tools. The SoS model is supposed to integrate constituent-system models originating from different tools and different modelling languages.

The SoS development process extends over years and involves numerous management activities and organizations. It is required to generate a holistic model that connects the technical models with the SoS management activities to have consistent information about the development activities and the management decisions.

Furthermore, the SoS model should be able to include executable models that are used for further analysis such as simulation and formal analysis. Its modelling language must be capable of adding language extensions that support the SoS development, e.g. architecture optimisation and analysis.

The UML language is very efficient in modelling software systems. The different diagrams support the operational, structural, and behaviour analysis. However, it is complicated to model a system that contains hardware and software components and their interactions. Therefore the SysML was introduced. SysML is characterised by its packages and block definitions that support the object-oriented modelling approach as in UML. It is used for describing the system as a whole using model-based systems engineering. SysML supports simulation and analysis, and has the UML features for adapting profile extensions to extend the language scope. However, describing the SoS and its constituent-system interactions using SysML is challenged by the model complexity when applying multiple levels and different constraints. Further limitations are the missing traceability between the architecture models, and the limited viewpoints that are supported by SysML, e.g. no support for management and capabilities.

The other modelling languages like Modelica and AADL concentrate more on describing physical aspects of the system (Modelica) or real-time properties (AADL). These modelling languages lack the ability to express all parts of a SoS and are difficult to be understood at the customer level.

In this thesis, the UPDM modelling language was chosen as the foundation for establishing the model-based approach for SoS due to the following reasons:

1- It supports architecture frameworks that include various viewpoints and it is widely used in industry for SoS modelling.

2- The UPDM meta-model facilitates SoS modelling by identifying the semantics and terminology to describe the SoS and its constituent systems.

3- UPDM provides the ability to combine models with different modelling languages using Functional Mock-up Interface (FMI) technology and functional mock-up units [75].

4- UPDM offers flexibility in adding profile extensions for improving the modelling process and integrating modelling technologies.

5- The UPDM terminology is easy to be understood by different SoS stakeholders at different levels, e.g. customers and developers.

While we use UPDM in this thesis to establish the model-based approach for the SoS architecture, other languages such as Modelica may be used for expressing the behaviour of the constituent systems.

3.5 Research Gap in the State of the Art

This chapter presented the state of the art of techniques and solutions used for SoS. The SoS engineering process is very complicated to be applied manually while considering all the aspects of the SoS and its constituent systems. It requires supporting tools and technical solutions that reduce the design efforts.

The model-based system engineering approach facilitates the engineering activities, documents the development process in models, enhances the communication between the system stakeholders and lets them share the same view of the expected SoS. However, the model-based system engineering requires additional efforts for the developers and it is time consuming. In particular, high efforts arise in SoS where the modelling of the constituent systems and their interactions requires large models that are difficult to be understood by the SoS stakeholders.

There are two important parts of the modelling of the SoS: the architecture framework and the modelling language that is used for modelling. Architecture frameworks like DoDAF and MODAF organise the SoS modelling and present the SoS from different perspectives in various viewpoints. Using the current architecture frameworks for building the SoS models reduces the model complexity and provides a better understandability for the model components and their interactions. However, in industrial applications the current usage of the architecture frameworks is limited due the following reasons:

- There is no clear methodology for applying the SoS engineering activities using the architecture frameworks. In some architecture frameworks (i.e. NAF) a general methodology is presented for organising the military applications, but not for SoS developments.
- The SoS architecture is built manually which makes it very difficult for the SoS developers to consider all the aspects of the SoS design in the development process while satisfying all the SoS requirements.
- The architecture frameworks are currently adopted in industrial applications using graphs and pictures. There is no further use for formal analysis. The models are lacking formal specifications of the behaviour that would allow the developers to apply simulation or formal analysis.
- Models are not reusable due to the sophisticated models of the SoS, which contain a large number of elements and connections. They are constructed for a particular SoS and are not re-used for new similar SoS.

In order to efficiently use the architecture frameworks, the current implementations must be enhanced by techniques that avoid the previous problems. In Figure 3-18 an example of the state of the art usage of the architecture frameworks is given. The architecture frameworks in this example are used to support the requirements elicitation process. The models are not used for further analysis or verification. Once the requirements elicitation process is finished, the model will not be used any more.

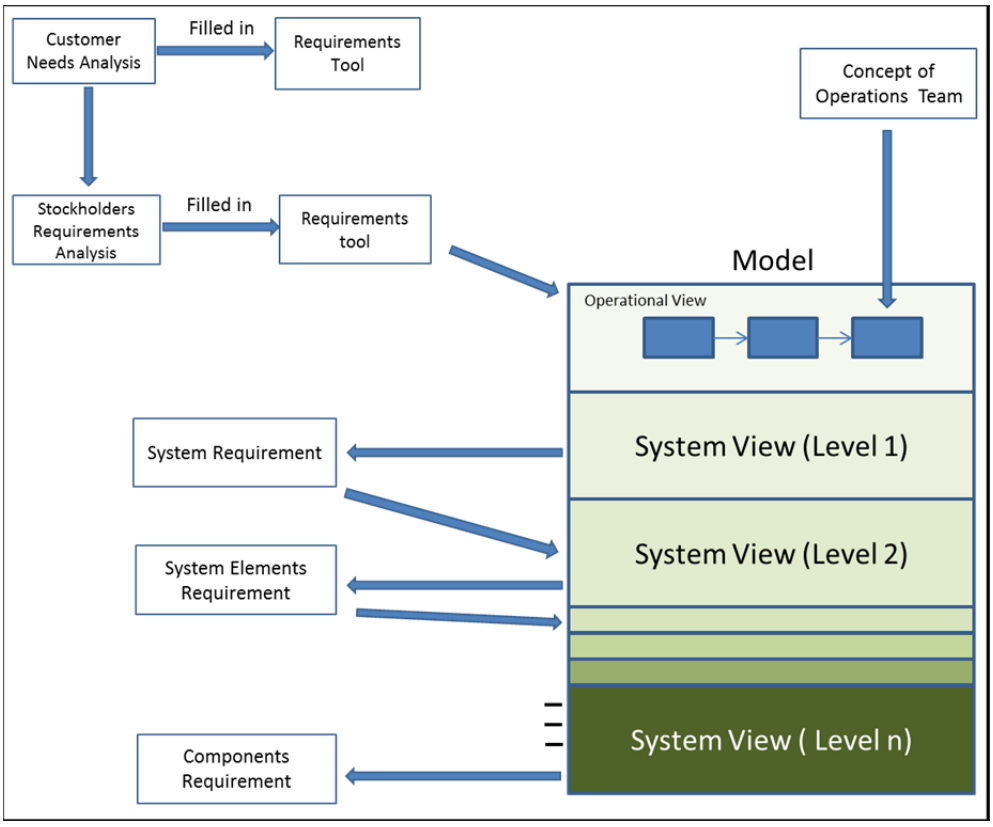

Figure 3-18 Industrial example of current architecture frameworks in SoS engineering

The UPDM as described before supports modelling using the architecture frameworks. The modelling language semantics and terminology defines the different parts and the components of the model. However, in order to apply different techniques which facilitate creating the SoS architecture (e.g. architecture optimisation) these languages need to be extended according to the required semantics used by these techniques. The current versions of UPDM do not support automated techniques for generating an SoS architecture.

The techniques for design space exploration are not connected to the SoS models that are used during the SoS development. To apply design space exploration, external models are constructed and the results are manually integrated into the SoS model in the modelling framework. This process is time consuming by doubling the modelling efforts and requiring the translation of the results to the SoS model.

The current state-of-the-art for modelling and design space exploration lacks support for the specification of timing properties in SoS models along with algorithms for model transformation and timing analysis. The violation of temporal constraints may lead to a SoS failure, since many SoS applications are real-time systems, e.g., emergency response systems and military applications. It is required to synthesize a system architecture that satisfies the temporal constraints, while also optimizing the system's overall goals such as cost and weight.

Reliability in SoS as addressed in this thesis is another open research problem. Considering system reliability in the development process becomes more complex at the level of large and complex systems such as SoS. The characteristics of the SoS result in challenges during SoS design where emergent behaviour, an evolutionary development process, and an increased state space needed to be considered. In particular, these challenges are unsolved when moving form directed SoS, where the system has central control, towards virtual SoS with no central management and no common purpose of the constituent systems. The complexity remains unsolved in

modelling and optimizing the SoS reliability using architecture frameworks. The current frameworks lack the methodology in defining and reflecting the SoS reliability at different modelling levels, and to consider the constituent-system reliability properties in the constructed SoS architecture.

4 Model-based System of System Engineering

4.1 Introduction

The most important challenges in the state of the art of SoS engineering are large and sophisticated models that are hard to read and trace, limited model reusability, insufficient design automation, and unexpected emergent behaviour. In our model-based SoS architecture methodology we address these challenges by extending modelling frameworks and modelling languages using design patterns and architecture optimisation techniques, while also considering real-time and reliability requirements.

This chapter contains three parts. The first section addresses architecture design patterns that are used to generate the SoS architecture and enhance model reusability. The second section discusses in details the concise modelling approach and its application to SoS architecture optimisation. The rest of the chapter presents a model-based SoS architecture modelling methodology based on architecture frameworks, design patterns, and concise modelling.

4.2 Architecture Patterns

4.2.1 Definition

Patterns are used in many engineering disciplines. For example, in building construction and city architectures a pattern describes a template for a solution to a problem in a way that enables it to be used multiple times [76]. A pattern in a system architecture 'refers to recurring structures, objects and events that can also be used as designs, blueprints, models or templates in the construction of other structures, objects and events' [77]. A pattern can be described using four elements [78]:

- **Pattern name**: The pattern name distinguishes the patterns from each other and provides information about the context of the pattern. The name must be unique and clear.
- **Problem**: The pattern is used as a solution for a problem. The pattern specifications must indicate this problem and the conditions under which the pattern can be applied.
- **Solution**: This is the core element where the pattern is described. If it is an architecture pattern then the component structure is described with the component interactions, connections, interfaces, dependencies, and responsibilities.
- **Consequences**: The consequences of the pattern are the results of applying the pattern, e.g., the effects on the cost, timing and reliability after applying the pattern on the system architecture. The pattern consequences make up the information which is required by developers in applying trade-offs between different solutions.

An architecture pattern in SoS is realised as a model of a part of the SoS architecture consisting of a set of constituent-system classes with their attributes and interactions. When applying the pattern, a concrete constituent system must be selected for each constituent-system class and attributes must be defined.

An architectural pattern provides a template for the structure and behaviour of a part of the system [79]. The architecture pattern is building classes for constituent systems and helps developers in making design decisions based on the system functionality, and offers a part of the architecture solution that describes a part of the SoS structure and the involved constituent systems with their parameters and interactions.

In model-based system engineering the system architecture is represented at different levels of abstractions, and from various perspectives. Architecture patterns in model-based system engineering allow the developer to organize the artefacts of the system's architecture at different levels, and provide an efficient mapping between them in standardized methods that reduce the design efforts.

4.2.2 Architecture Patterns in SoS

SoS are large-scale systems that cannot be modelled manually using methods from monolithic systems. Architecture patterns are used to automatize the selection of the constituent systems and to configure the interconnections while covering all the required SoS functions. Using architecture patterns, developers deal with the model of patterns rather than the model of constituent systems when building the SoS models, which results in the following benefits:

1. Model complexity is reduced by representing different parts of the model as patterns that are connected and presented in a standardized form. Therefore patterns enhance the traceability by using the connections between patterns and facilitate model changes by reapplying the used patterns.
2. SoS evolution can be managed in the modelling process by planning and introducing new patterns that are designed to match the legacy systems or by instantiating the used patterns with new constituent systems. The goal is to adapt the evolution of the SoS by adding new capabilities as explained in section 4.2.4.
3. Model reusability is one of the major issues in SoS modelling. The SoS models are very large and contain many modelling elements that are distributed over multiple views. By generating architecture patterns that are connected to generic SoS requirements, these patterns can be used for many SoS according to their properties and characteristics. For example, in an emergency response SoS an architecture pattern for the centralised command and control process will be generated based on the process requirements. Its specifications will be documented and once a new SoS is designed with similar specifications – which includes the command and control process - the same pattern can be used. This reduces the design cost, time and efforts and increases the efficiency of developing new SoS.
4. Undesirable emergent behaviour is minimized. Patterns are generated out of experience and through iterative methods (see Section 4.2.3). The solution associated with a pattern can be tested and analysed beforehand, thereby predicting expected behaviours. By utilizing engineering feedback during the formation and selection of patterns, undesired emergent behaviours can be avoided. However, this is limited to weak emergent behaviour while emergent behaviour of multiple patterns or strong emergent behaviour of a single pattern is still unresolved.

At the SoS level, architecture patterns are expressed by models of constituent-system classes and their connections. A constituent-system class models a general type of the constituent system. A class describes the common attributes, properties, interfaces, and ports of a constituent system. An architecture pattern presents how the involved constituent-system classes are connected, their interactions, their design constraints, and their multiplicity based on the pattern's elements (i.e., problem, solution, desired consequences). Later on these classes are instantiated by concrete constituent systems with defined values for their attributes.

4.2.3 Mining Architecture Patterns

In designing a new SoS, previously constructed SoS form a useful source of information and practices. Legacy SoS can be analysed for functionality and desired capabilities that attract the interest of experienced engineers. Extracting architecture patterns is particularly useful for solving the same problems in future SoS that provide the same capabilities. Figure 4-1 illustrates an example of a pattern mining process as described in [80], where all the activities in this example are manually applied.

The process starts with the analysis of the existing SoS and its constituent systems to identify the required information about the current structure and behaviour of the SoS. The next step is done by pattern experts to partition the SoS architecture into different architectural parts and layers based on functional and behavioural aspects. The resulting architectures are then reviewed to ensure that their required functionality and behaviour are met.

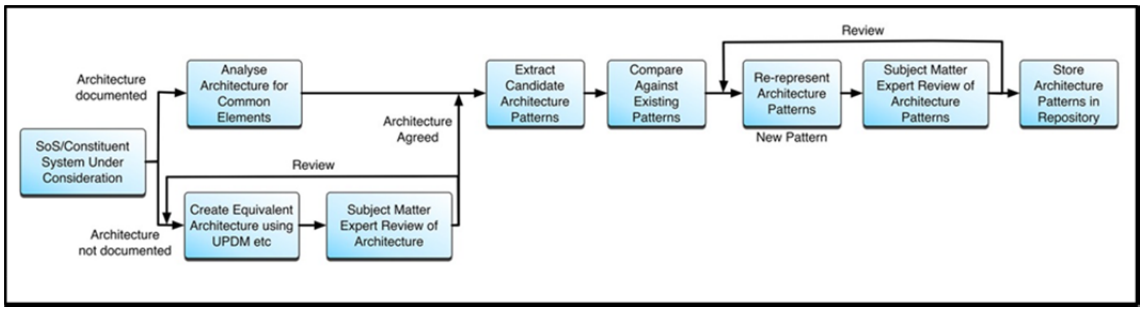

Figure 4-1 Architecture pattern mining process [80]

Architectures of previous SoS form the input to build the architecture patterns. They are analysed and patterns are manually extracted. The extracted patterns are then compared with the existing patterns to identify new patterns and add them to a repository.

In the extraction of architecture patterns the following points must be considered:

1- Architecture patterns must be generic enough to allow the use in multiple applications, e.g. using different instantiations of constituent systems.

2- The level of abstraction of the pattern must be defined. In SoS development process, architecture patterns are used to map different levels of the SoS architecture. An architecture pattern can be used as a mapping between operations and functions, functions and constituent systems, or constituent systems and their components. The extracted patterns must be classified according to the respective level of abstraction while keeping the traceability between these levels and defining their logical connections. In this thesis we consider the mapping between SoS functions and constituent systems.

3- The pattern mining process is an iterative method where the extracted patterns are evaluated, analysed and re-designed according to the analysis results.

4- The dependencies between architecture patterns must be clearly defined and documented. This applies to functional and extra-functional dependencies.

The extracted patterns are instantiated with constituent systems based on different design criteria (e.g., cost, quality) and the generated architecture parts are tested and verified. The patterns are added to the pattern repository and documented with their specifications, application conditions, constraints and analysis results.

4.2.4 Using Architecture Patterns in SoS

The purpose of the architecture patterns, as used in this thesis, is to map the functional level to the architecture level and to automatically generate the SoS architecture. The SoS functions are allocated to the constituent-system classes of an architecture pattern and by instantiating these classes with constituent systems a part of the SoS architecture is synthesized. Using the UPDM modelling language for SoS modelling, the generated architecture patterns are expressed as profile extensions, which allows the user to apply the pattern as a package of the selected view. Figure 4-2 shows how architecture patterns are used in generating the parts of the SoS architecture.

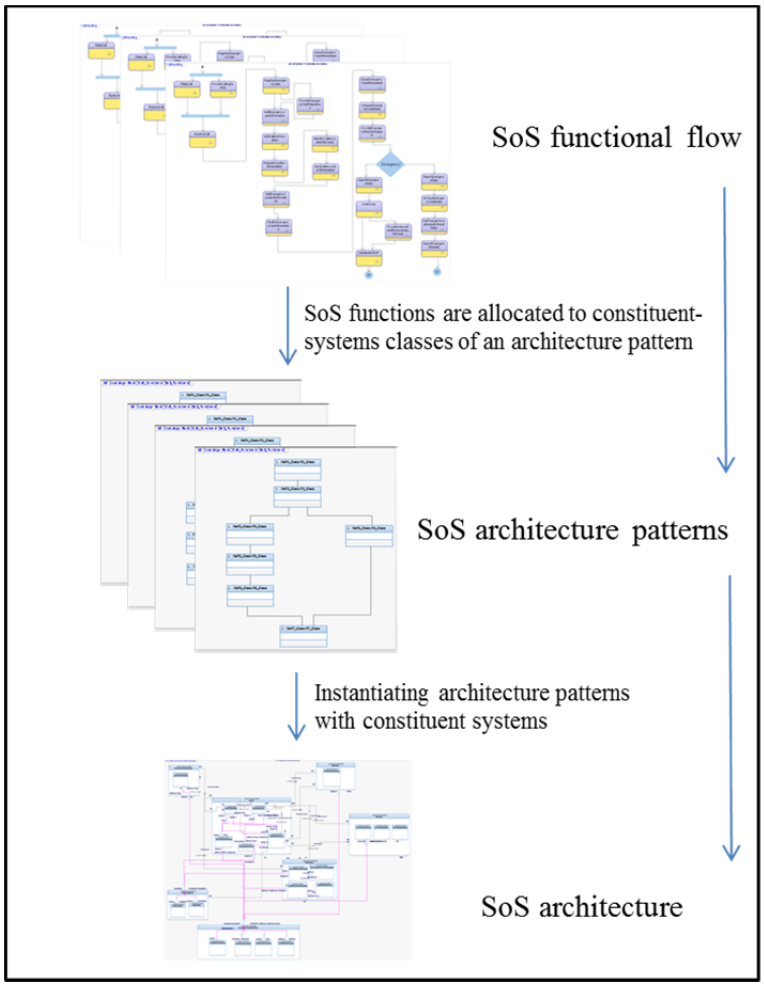

Figure 4-2 Architecture patterns at different stages of the SoS development process

During the SoS architecture development, the patterns are selected based on the requirements and the design criteria (e.g. cost, performance, quality). The process of pattern extraction is an iterative method, therefore during the development process new patterns will be added and some of the patterns will be re-designed based on the behaviour of the instantiated SoS architecture. The chosen patterns are tested within the SoS architecture and adapted to the SoS configurations. Figure 4-3 illustrates the use of architecture patterns in the SoS architecture development [81].

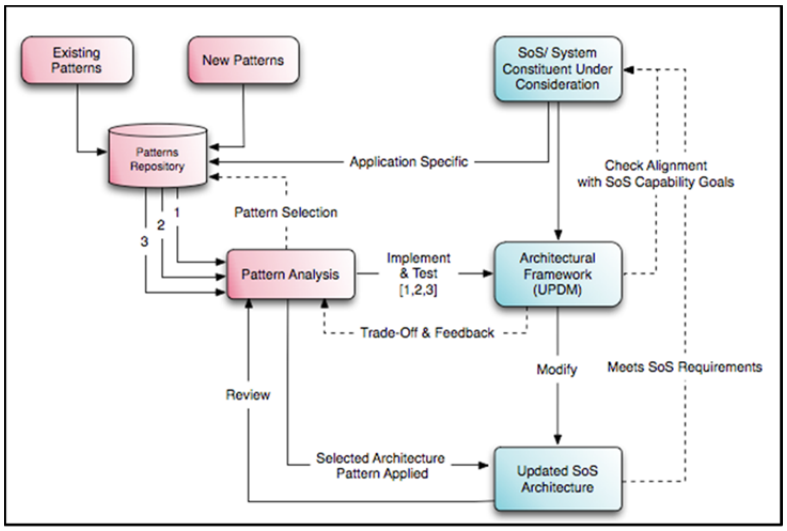

Figure 4-3 Using architecture patterns in SoS architecture development [81]

Architecture patterns are organized in the pattern repository as a pattern library. Each of the patterns is documented with its constituent-system classes and architecture information (i.e., connections and ports). To facilitate finding the required patterns, a pattern keyword is contained in the pattern information. The pattern information also includes the patterns elements (i.e., name, problem, solution, consequences). Related patterns are also listed within the pattern data with the pattern dependency information.

The extraction and adaptation of architecture patterns is a dynamic process that is part of the SoS evolution. New patterns that adopt new technologies and new constituent systems are added to the pattern library and are used to generate parts of the SoS architecture that will be integrated within the currently used SoS architecture when required.

The SoS design activity is a continuous process. Some patterns are designed to solve problems that arise when undesired emergent behaviour occurs. This kind of emergent behaviour is analysed and the solution is realized as an architecture pattern that generates SoS architectures avoiding the occurrence of similar behaviours. In Figure 4-4 the dynamic interaction between the SoS architecture and the architecture patterns over the SoS life cycle is depicted.

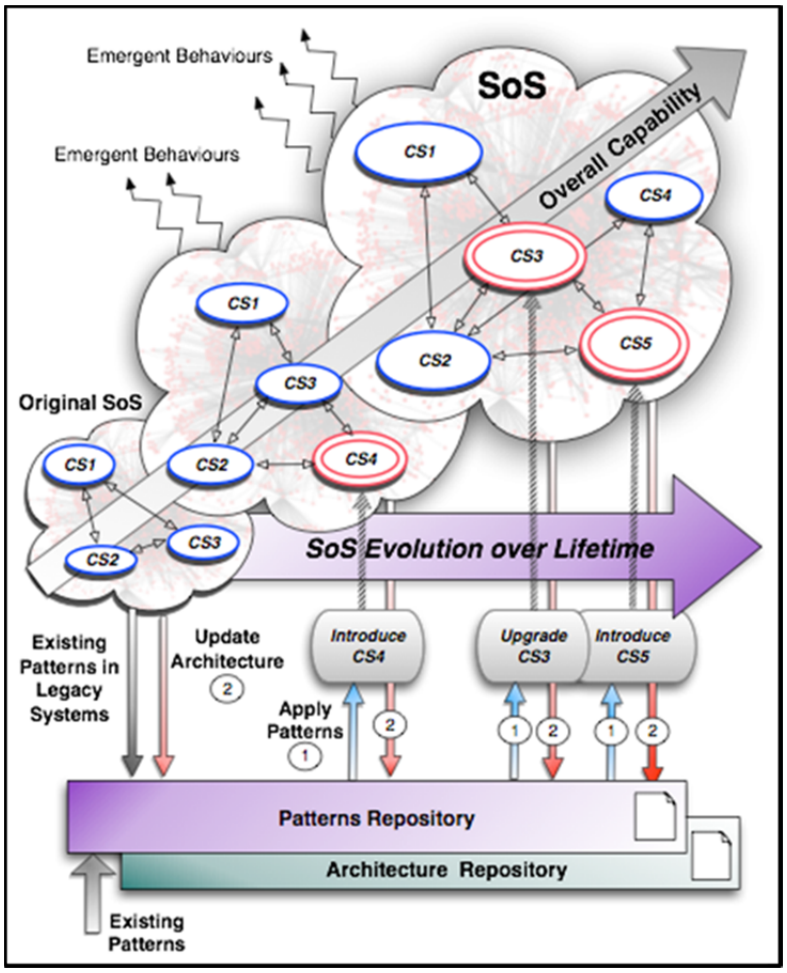

Figure 4-4 Dynamic interaction of architecture patterns over the SoS life cycle [81]

The SoS design activities are extended over the SoS life cycle to realise new constituent-system capabilities and new SoS requirements. Architecture patterns are used to facilitate the SoS architecture evolution. New patterns are designed to realise the additional requirements or old patterns are re-instantiated with new constituent system to generate SoS architecture parts with new capabilities. The new architecture parts are integrated in the SoS legacy architecture and old architecture parts are removed.

Architecture patterns are also used as a template for architecture optimisation as illustrated in section 4.3.1.

4.3 Architecture Optimisation

Present day SoS architecture processes aim to deliver the required SoS based on its functional requirements. Applying optimisation methods is limited to small parts of the architecture due to the following reasons:

1. The design space of the SoS architecture is very large due to the large number of constituent systems that are included in a SoS.
2. SoS architectures involve a large number of interactions and dependencies that are hard to be included in the architecture optimisation processes.
3. There is no methodology that considers the non-functional properties of the SoS, while applying the architecture optimisation process.

4. It is hard to define all constraints for the architecture optimisation process due to the large number of constituent systems and the complexity of their connections and interactions.
5. Time limitations in the design of SoS typically prevent optimization for delivering the architecture. Although the design process of SoS is extended over years, due to the large number of the involved constituent systems and their interactions, the focus of the developers is to deliver the SoS that has the required functionality without considering an optimized solution.
6. The architecture optimisation process is not supported by the current modelling and architecture frameworks. The architecture optimisation uses dedicated optimisation languages as an input for the optimisation engines. There is no mapping to the models expressed in the SoS architecture frameworks.

IBM research presented a Design Space Exploration (DSE) methodology for monolithic systems based on MBSE [82]. The methodology connects the requirements domain to the functional domain and generates optimum system architectures with regards to the optimisation goals. It uses SysML diagrams and semantics to define the system's functional flow and the system architecture. In this thesis we have extended this methodology to SoS using UPDM views and diagrams.

DSE includes four steps:

1- **Formalize requirements**: The SoS requirements are defined and analysed. The process includes the definition of the system functions of the required architecture and the formulation of the optimisation objectives and constraints with the required equations and parameters.

2- **Define design alternatives**: At this stage the candidate constituent systems and potential architectures are established and organized using a database that contains the required information about the alternatives (i.e. parameters and properties such as cost).

3- **Evaluate design alternatives**: Using an optimisation engine the design alternatives are evaluated and optimized with the constituent-system candidates.

4- **Choose the optimal solution**: The alternatives are presented and compared based on the optimisation objectives and the optimal solution is selected.

Figure 4-5 shows the process flow for applying the MBSE in the DSE methodology. The process starts with building the functional models according to the requirements analysis. This includes the identification of the functions that are required to fulfil the SoS requirements and the logical flow between these functions.

The next step is to define the connections between the constituent systems and the constraints of the SoS architecture. At this step architecture patterns are used to derive the potential architectures (see section 4.3.1). At this stage the constituent systems are used as classes with different possible instantiations. The constituent-system classes are then mapped to the functions that they implement.

The optimisation constraints and the optimisation goals are modelled using a concise modelling approach (see section 4.3.1) and attached to the model. A catalogue database is then generated based on the chosen architecture pattern and filled later with the candidate constituent systems.

Thereafter, the optimisation problem is automatically established and sent to the optimisation engine to compute the optimal solution. The optimal architecture is back-annotated to the system model with the constituent-system connections and the instantiations of the constituent systems.

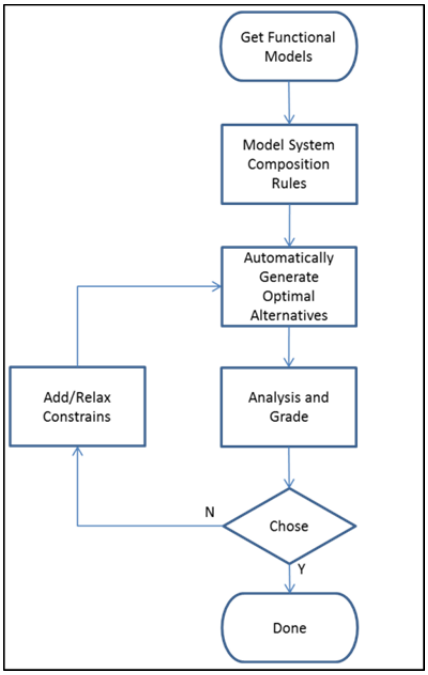

Figure 4-5 DSE methodology process flow

4.3.1 Using Architecture Patterns in Architecture Optimisation

Architecture patterns are used to describe a part of the SoS structure with its constituent-system classes and their relations. The SoS structure is modelled in a generic form to allow for a high degree of flexibility that enables the optimisation process to optimize the selection of the constituent systems and their parameters.

The architecture patterns are used in the architecture optimisation at two levels. Firstly, we chose the architecture patterns that fulfil the functional flow of the system. These architecture patterns are used to perform the instantiation of the constituent systems. Secondly, these architecture patterns are instantiated using an architecture optimization process to generate a part of the system architecture. In Figure 4-6 both levels are illustrated.

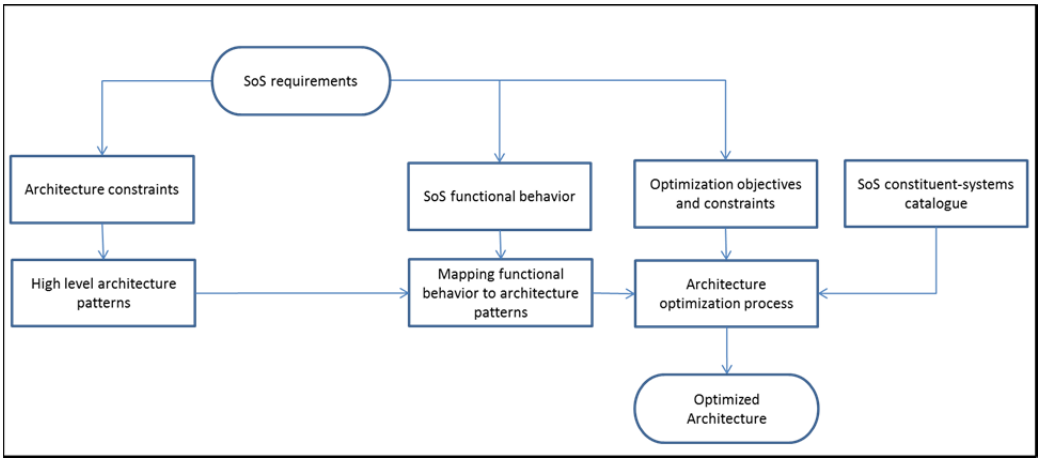

Figure 4-6 Using architecture patterns in the architecture optimisation process flow

The process starts with analysing the SoS requirements to extract the architecture constraints and to build or chose the architecture patterns from the patterns library that match these requirements. The requirements analysis process also includes defining the optimisation objectives and establishing the SoS functional flow.

The SoS functional behaviour is then mapped to the architecture pattern elements (i.e., constituent-system classes) with a degree of flexibility that allows the optimisation process to optimize the functional allocation in case of different options. Parallel to the previous steps, a database containing the candidates (i.e., constituent systems) with their attributes is established.

The architecture optimisation process uses the results of the previous steps as inputs and generates the optimized architecture based on the criteria that are defined in the previous models. The optimized results contain instantiations of the constituent-system classes with their multiplicity and function allocations to these constituent systems.

The generated architectures are used to compare and evaluate different patterns through a static analysis of their properties and performance. The architecture optimisation process instantiates the architecture patterns and evaluates their attributes based on performance metrics that are defined in the pattern model. The results are evaluated and the patterns are associated in the pattern library with their performance calculations. For example, we can consider the comparison of two architecture patterns that are both used in command and control operations. In the optimisation process of the two architectures the results show that the first one is less expensive, but with no redundancy in its architecture, which results in low reliability. In contrast, the second one is more expensive but has increased reliability due to the system redundancy. Depending on the operation's reliability requirements, the second one will be chosen, but both patterns will be recorded in the pattern library as alternatives for future applications. Figure 4-7 illustrates the process of organizing the architecture patterns in the pattern library.

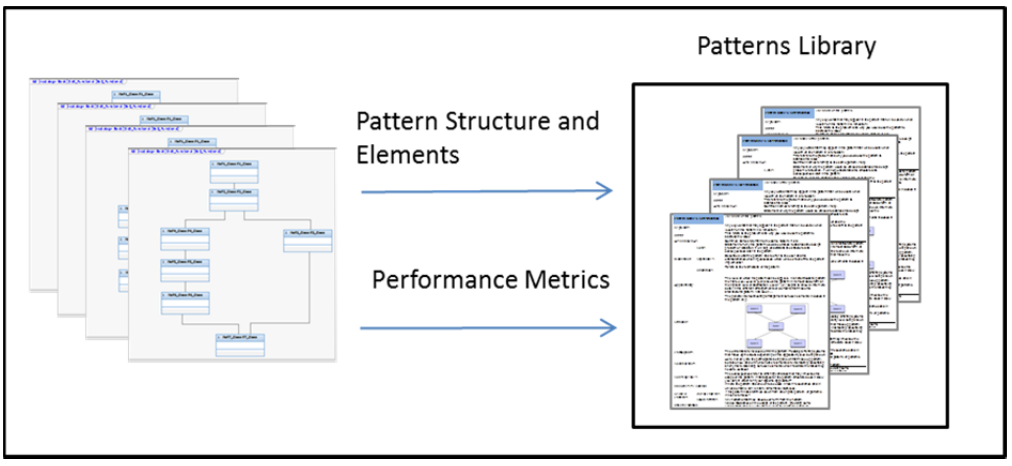

Figure 4-7 Comparing and organizing architecture patterns using architecture optimisation

4.3.2 Modelling Process

The architecture optimisation process starts with modelling the functional flow and the corresponding architecture patterns. The next step is to generate the constituent-system catalogue based on the chosen architecture patterns and to fill-in this catalogue with the constituent-system candidates and their parameters.

The following step is to express the optimization problem in the model together with the catalogue using OPL and to solve it using an optimization engine.

The last step is to back annotate the results to the structure models. Figure 4-8 illustrates the architecture optimisation steps:

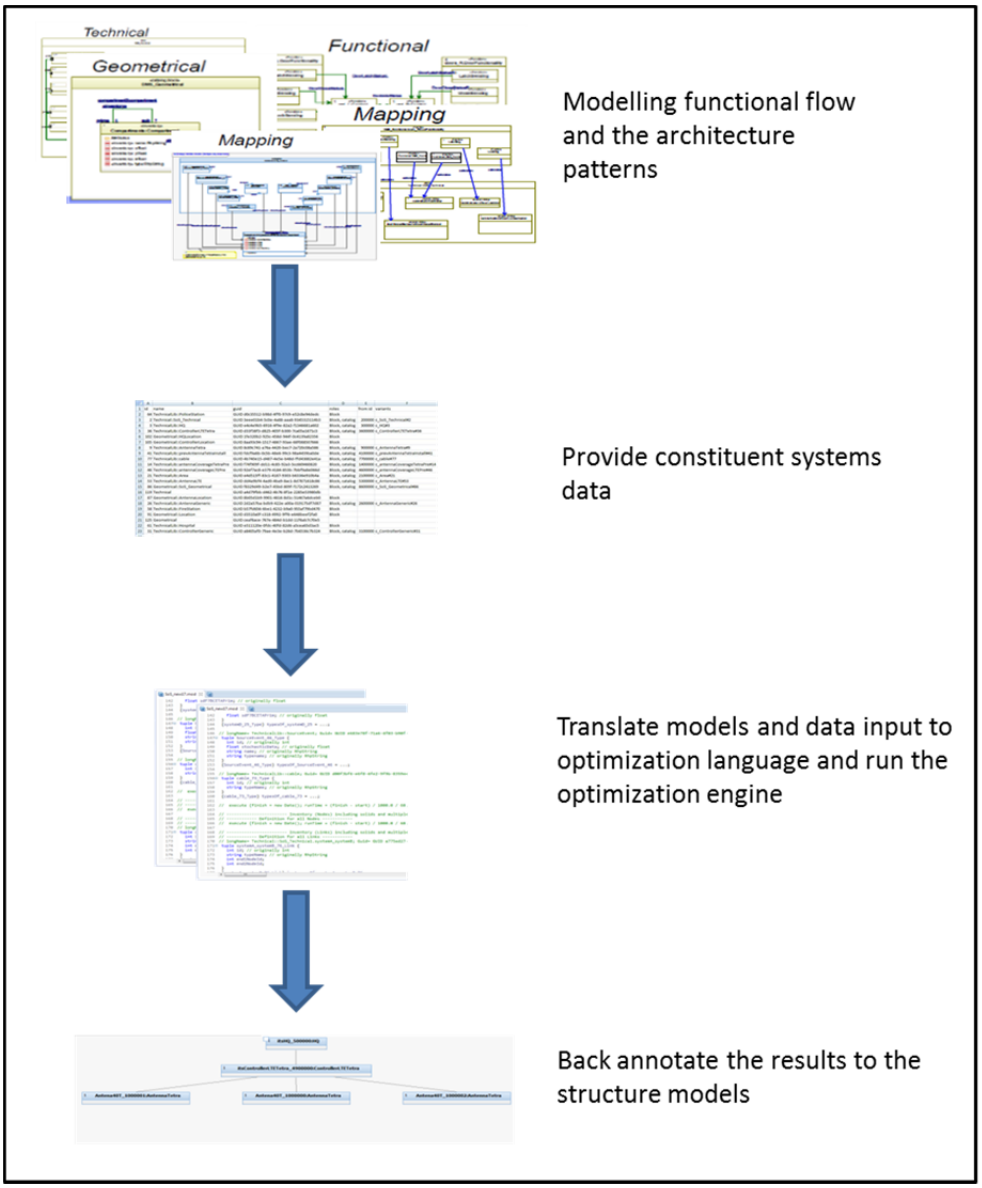

Figure 4-8 Architecture optimisation process

The models are created using the concise modelling approach [82], where the models express the rules of the system composition. The concise modelling extends the SysML and UPDM with the concise profile extensions.

The concise modelling approach is used as an interface between the models of the system expressed in SysML and the optimisation engine. The concise profile is a set of stereotypes and attributes that are used to mark the optimized elements, to define the optimization goals, and to add the optimization constraints. A concise plugin converts the model, based on the concise-profile stereotypes, to a mathematical optimisation problem expressed by an Optimisation Programming Language (OPL) [83] using a pre-defined optimisation template. The optimisation problem is then solved by an optimisation engine and the results are converted back using a concise plugin,

to an architecture model. Figure 4-9 explains the concise modelling approach. The optimisation problem is modelled in SysML and marked with concise stereotypes (see Table 7).

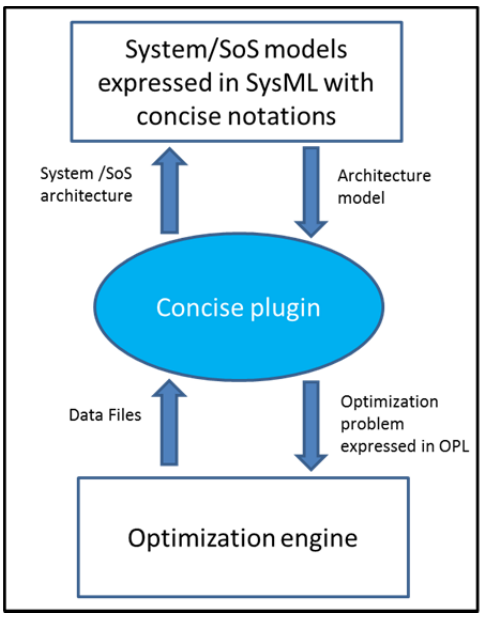

Figure 4-9 Concise plugin

In concise modelling the model is presented in three layers using the internal block diagram of SysML with blocks, parts, classes, and connectors:

- Functional layer: It models the functional behaviour of the SoS. It contains the functional flow that describes the dependencies and connections between the functions.
- Technical layer: At this layer the architecture patterns are modelled. The model represents the constituent-system classes, interactions and connections.
- Geographical (indexing) layer: The geographical layer is an additional layer for defining the geographical locations of the constituent systems. In some applications the locations of the constituent systems are considered in the optimisation. For example, when optimizing a wireless communication system the positioning of the antennas affects the coverage area and the fixed and operational cost of the system.

There are dependencies and connections between the three layers. The layers are connected through two mapping processes: functional to technical, and technical to geographical. In the functional to technical mapping process the functions represented in the functional flow are mapped to the constituent-system classes as represented in the architecture patterns according to the capabilities of the constituent-system classes.

In the technical to geographical mapping process, each of the constituent-system classes represented in the technical layer is mapped to the possible locations. The mapping is a many-to-many relationship, which means that there are multiple options for the locations of the constituent systems and the locations can be mapped to multiple constituent systems.

The concise model generates a database that documents the model information and allows the developers to add information to the constituent-system catalogue. The model elements are marked with SysML and concise stereotypes that define their roles and categories. Table 7 lists these stereotypes and their usage:

Table 7 SysML and concise stereotypes used in architecture optimisation

Stereotype	***Defined in***	***Description***
Function	SysML	Denotes that the model element is used to define the system's functions and their requirements.
Technical	SysML	Denotes that the model element is used to define the constituent systems and connections.
mappedTo	SysML	Used to indicate the mapping between functional and technical layers.
allocatedTo	SysML	Used to indicate the mapping between technical and geographical layers.
typedConnector	SysML	Denotes a link between the model elements (e.g. constituent systems). It represents a physical object (e.g. cable).
Catalogue	concise	Denotes that the model-element variants are described in the catalogue.
Optimized	concise	Denotes that the value of the model element is set by the optimisation process.
inventory	concise	Denotes that the value of the model element is already fixed and indicated in the catalogue.
expand	concise	Indicates that model elements will be displayed in the back annotated architecture.

The optimisation objectives are added to the model as attributes of the system and marked with the 'sow_goal_attribute' stereotype. This stereotype enables the attribute to possess optional tags that give the modeller the ability to choose between maximizing and minimizing the optimisation objective, and to define the op-

timisation objective priority for multi-objective optimisation. The matrices for calculating the optimisation objective are defined by attaching constraints to the optimisation objective attribute.

Further constraints to the optimisation problem can be added to the technical block that defines the architecture pattern. The constraints are marked with the 'sow_constraint' stereotype. Both the optimisation objective calculation matrices and the optimisation constraints are written in OPL [83].

After building the required models, a database that contains the model information is generated using a concise plugin in the modelling tool. The database provides the developer with a mechanism to add the different variants of the model elements and the constituent-system catalogue. The database is generated using another concise plugin. Figure 4-10 presents an example of the generated database for optimizing antenna locations of a communication system. The figure shows the links to the tables that contain the data generated from the model. For example the geographical location table contains the optional locations of the constituent systems with their identification numbers and coordinates.

	A	B	C	D	E	F
1	id	name	guid	roles	from id	variants
2	16	TechnicalLib::antennaCoveragePre	GUID f74f909f-dd11-4c85-92a3-5cc669460820	Block, catalog	2E+06	c_antennaCoveragePre#16
3	48	Geometrical	GUID ceaf6ace-767e-484d-b1dd-11f6ab7c70e5			
4	33	Geometrical::Location	GUID d3510a0f-c318-4992-9ff6-e648beef2fa0	Block		
5	23	TechnicalLib::Area	GUID e4d515ff-83c1-4187-9303-b6336e910b4a	Block, catalog	2E+06	c_Area#23
6	6	Technical	GUID a4d79fbb-d462-4b76-8f1e-2285e53980db			
7	7	Technical::SoS_Technical	GUID 3eee01b4-5c0e-4a88-aaa8-9345315114b3	Block, catalog	700000	c_SoS_Technical#7
8	8	TechnicalLib::HQ	GUID e4c4e9b5-6916-4f9e-82a2-f1346681a602	Block, catalog	800000	c_HQ#8
9	40	Geometrical::HQLocation	GUID 1fe320b2-925c-458d-944f-0c4139a82356	Block		
10	11	TechnicalLib::Antenna	GUID 8c69c741-a76a-4420-bec7-2a720c08a586	Block, catalog	1E+06	c_Antenna#11
11	29	Geometrical::SoS_Geometrical	GUID f8329d49-b2e7-45bd-809f-f172c2413269	Block, catalog	3E+06	c_SoS_Geometrical#29
12	30	Geometrical::AntennaLocation	GUID 8b05d1b9-9901-4816-8d1c-51467a6dceb0	Block		
13						
14						
15						
16						
17						
18						

Figure 4-10 Constituent-system catalogue database generated using concise plugin

4.4 Modelling Methodology

In the previous sections we presented some of the architecture techniques that are used to support the system architecture development process. In this section we describe a model-based architecture development methodology for SoS that integrates the previously described techniques within an architecture framework. Our methodology is based on the DoDAF architecture framework as the development environment. We describe the SoS models using a subset of the DoDAF viewpoints. Using a subset of the viewpoints does not mean that the other unused viewpoints are useless; rather they may be integrated within the whole approach if required. We used the viewpoints that we believe are the most important ones to describe the architecture.

In our methodology, UPDM is used as a modelling language with the developed profile extensions for UPDM that support our analysis techniques (e.g. architecture optimisation). The previously mentioned architecture optimisation techniques are integrated with the architecture patterns to automatically generate an optimized SoS architecture.

The methodology supports re-usability of SoS models by integrating architecture pattern techniques into the model development. It describes a step-by-step SoS architecture development process that enables the modellers to apply further analysis using simulation or model checking techniques.

In chapters 5 and 6 we extend our methodology to include real-time and reliability requirements in the architecture development process.

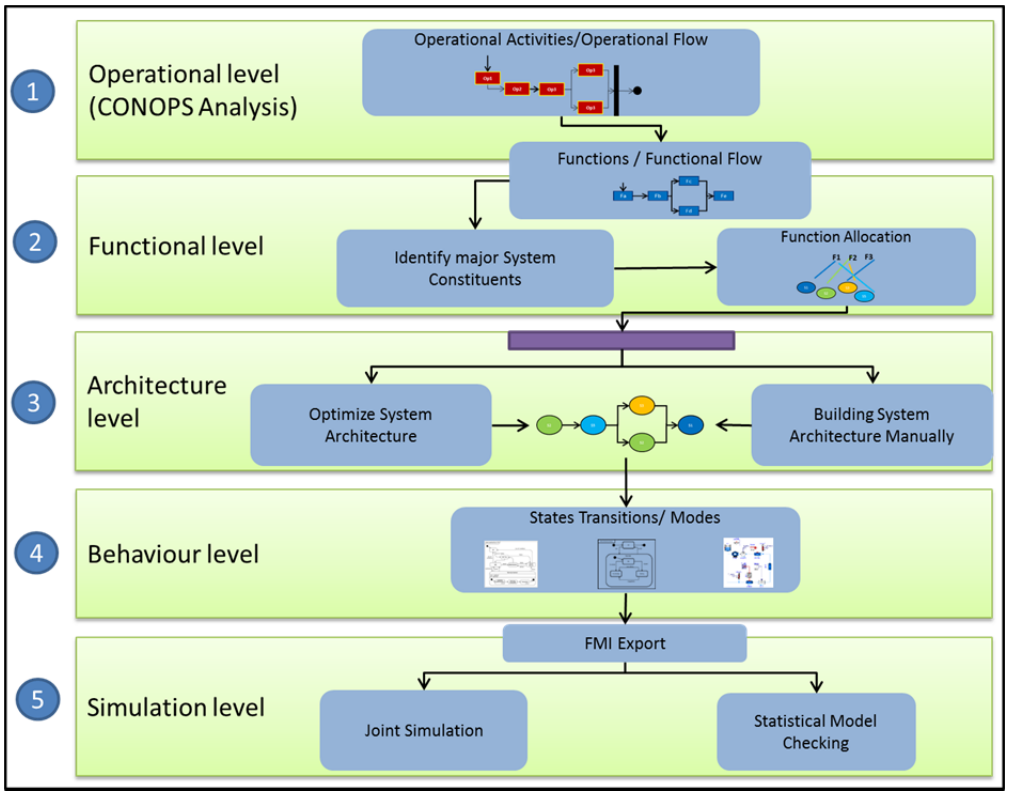

Figure 4-11 General flow of architecture modelling methodology

The modelling methodology distinguishes six modelling levels as shown in Figure 4-11:

1- Operational level: The purpose of building the operational level is to realize the system requirements in operational scenarios and to state the concept of operations (CONOPS)

2- Functional level: The functional level is the intermediate level between the operational level and generating the SoS architecture. It maps the operational domain to the functional domain and identifies the rules and constraints for building the SoS architecture.

3- Architecture level: At this level the SoS architecture is either built manually or automatically by using the architecture optimisation techniques.

4- Behaviour level: The behaviour of the constituent systems is modelled and linked to the SoS architecture.

5- Simulation level: The SoS architecture is analysed either by simulation or by statistical model checking to verify the model requirements.

The methodology defines a general approach of the SoS architecture development. We consider the practical SoS development process in industry that starts with operational scenarios, and we enhance it with modelling techniques. In the following, the modelling steps are described and mapped to the respective views in the DoDAF architecture framework (c.f. Figure 4-12).

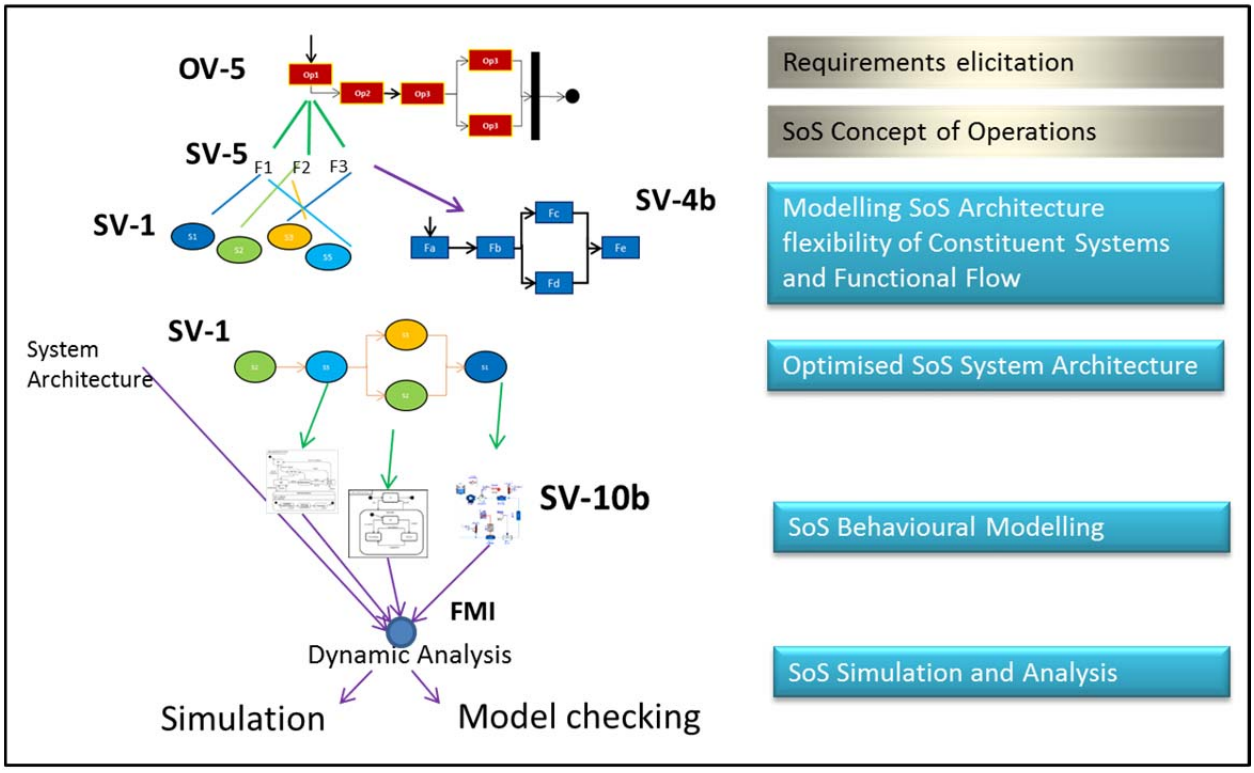

Figure 4-12 Architecture modelling methodology in DoDAF views

Operational activities flow diagram: The starting point is the system requirements analysis and elicitation. On the basis of the customer needs, the system requirements are formulated and realised in operational scenarios. The operational scenarios describe the operational activities required to fulfil the customer needs.

From the operational scenarios analysis the SoS operational activities diagrams are built at a high level of abstraction using a naming terminology that is understandable to the SoS customer. Building the operational activity diagram is an iterative process where the diagrams are discussed with the customer and refined to match the customer needs. In parallel to this process, the system requirements are also refined based on the operation activity flow.

The requirements analysis includes non-functional requirements and constraints of the required SoS (e.g. cost). These requirements and constraints are used later in the architecture optimisation process. The operational activity view in the DoDAF architecture framework (OV-5) is used to present the operation activity flow.

Figure 4-13 depicts a simple example for one of the emergency response SoS operational scenarios. It illustrates the police operational scenario at a high level of abstraction. In this example the main operational activities describe the flow of actions that are executed at the police station when receiving an order from the command and control centre, which manages the emergency response SoS. The operational activity flow starts with receiving the dispatching order, which contains information about the required resource that must be released by the police station to the emergency site. The order is checked regarding the resource availability, and based on the evaluation results the resources will be released or a shortage report will be issued. The process feedback will be sent back to the monitoring station at the command and control centre.

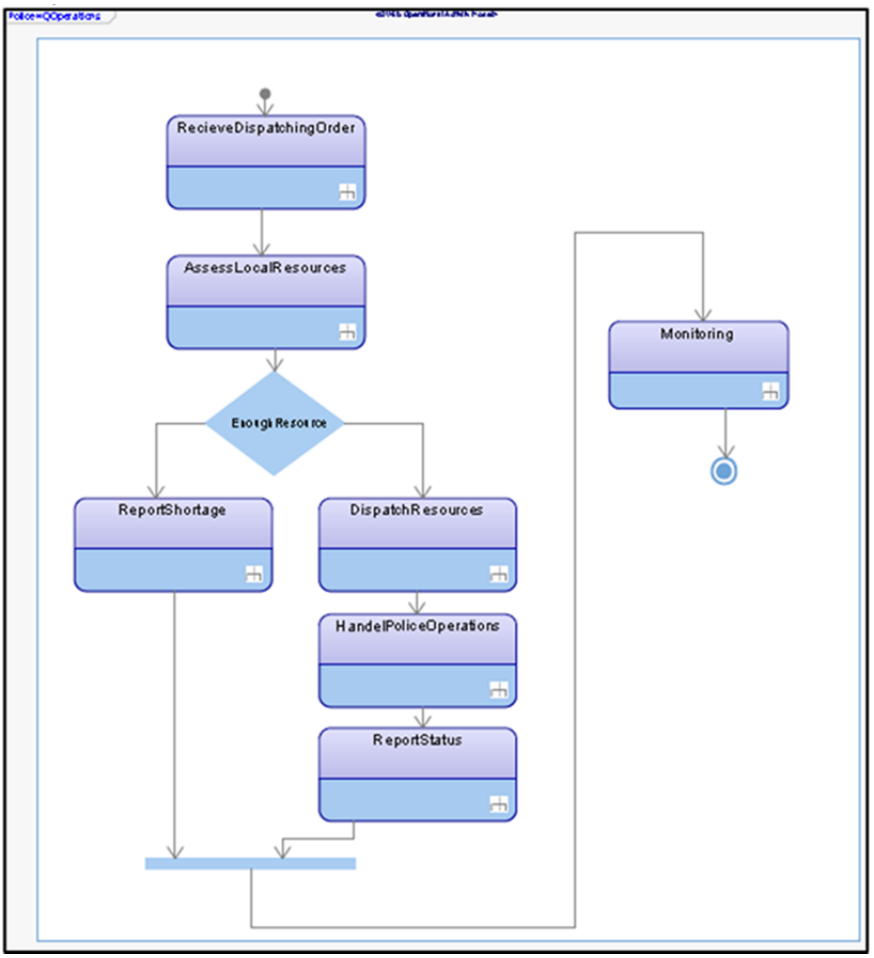

Figure 4-13 Operational activity diagram for police operational scenario

Operational activities to functions mapping: the previous step is done at a high level of abstraction which allows the diagrams to be understood by both customers and system architects. In order to move from the customer domain to the developer domain each of the operational activities is mapped to the required functions in order to implement these operations. Each of these activities must be mapped to at least one function. The functions listed under the system view package and the 'operation activities to systems and systems functions mapping' view (SV-5) are used to describe the mapping process. For example, in Figure 4-14 the operational activity 'AssessLocalResource' is mapped to four functions that are required to perform this activity: import the common relevant operational report, check the local strategy, check the resource availability, and check whether the required resources are mapped to the function. The results of the mapping process are listed in the 'operational activity to systems function traceability matrix' of DoDAF.

This step is an intermediate level between the customer and developer domains. The mapping process is established according to the discussions between the SoS architects and the developers. It requires experienced engineers supported by a library of the system functions and their capabilities.

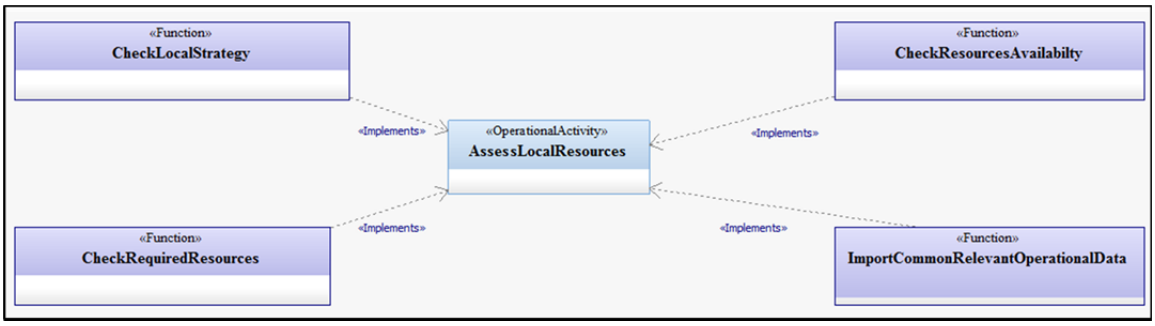

Figure 4-14 Operational activity to functions mapping

Functional flow diagram: This step is part of the developer's domain and uses the developer's terminology and definitions for functions and constituent systems. Using the operational activities to functions mapping and the operational activities flow diagram, the functional flow diagram is created. It indicates the dependencies and connections between the functions of the SoS. The 'system functionality description' view of DoDAF is used for this purpose. In this step the functional flow of the required SoS architecture is described and the constraints that must be considered in establishing the SoS architecture are identified.

Mapping of functions to constituent-system classes and identification of architecture patterns: At this level the constituent systems are modelled as classes without instantiations. Classes allow the general modelling of different types of constituent systems with variable attributes and parameters with different possible concrete instances. These values are assigned later during the specific instantiation of the constituent systems. The functions are mapped to constituent-system classes that handle these functions using the 'system interface description' view of DoDAF. For example, as shown in Figure 4-15 the local dispatching system is handling the receiving, alerting and issuing of the dispatching order as well as the releasing of the resources. The details of the local dispatching system are not specified (i.e. which type of constituent system or brand). Using this mapping and the functional flow diagram as a guide, general architecture patterns for the system architecture can be defined or chosen from the pattern library. An architecture pattern denotes a set of constituent-system classes and their interconnection to a particular function.

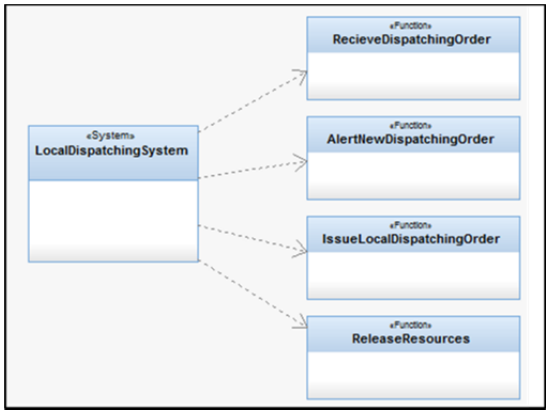

Figure 4-15 Mapping of functions to constituent-system classes

Creating the SoS architecture: The SoS architecture is either created manually using the previous diagrams as a guide or generated automatically using architecture optimisation techniques. In the architecture optimisation approach we extend UPDM with a concise profile extension (i.e. concise stereotypes for optimisation) and we

use the DoDAF framework views to support the optimisation process. Figure 4-16 shows the packages that relate to the DoDAF framework, where system packages are additionally created.

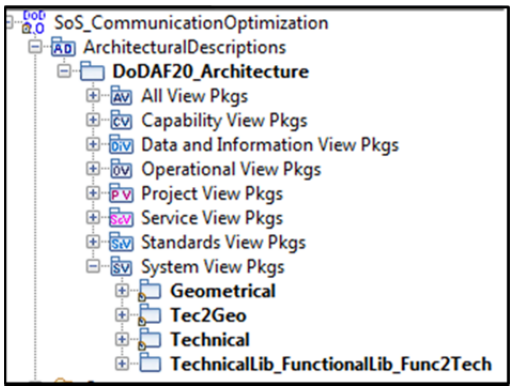

Figure 4-16 Using concise modelling packages in DoDAF

Four system view packages were created to describe the SoS model and to support the concise modelling approach:

1- **Library of constituent systems functions and constituent systems**: In this package the constituent systems and their functions are defined. In the SoS model it is indicated as 'Technical-Lib_FunctionalLib_functional2Tech' package as shown in Figure 4-17. The constituent systems are created and listed in a Block Definition Diagram (BDD). The second diagram is the system functionality diagram. Using this diagram the functional flow of the SoS is modelled as described in the previous steps. The last diagram is the mapping between the functional diagram and the constituent-system diagram, which maps the functions to the constituent-system classes as described in the 'mapping of functions to constituent-system classes' step.

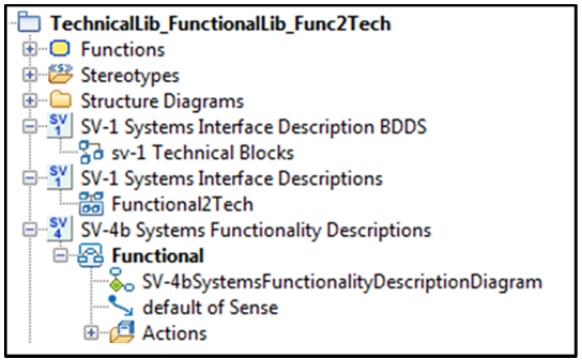

Figure 4-17 Package for constituent systems and functions in UPDM

2- **Geographical information**: one of the SoS characteristics is the geographical distribution of its constituent systems. Therefore, in many applications the architecture optimisation needs to determine the optimal positioning of the constituent systems. The geographical location of the constituent systems affects the optimisation results. The geometrical package of UPDM is used to define the positioning of the constituent systems. It contains a block definition diagram to define the location classes and a structure diagram that illustrates how these locations are related to the SoS.

3- **Mapping of constituent systems to geographical locations**: This information is indicated in the model as a package named 'Technical to geographical' (Tec2Geo). It is used to model the mapping between the geographical location classes and the constituent-system classes.

4- **SoS architecture patterns**: The technical package contains the architecture patterns that are used in the architecture optimisation process. The architecture patterns refer to the constituent-system classes and describe their connections at a high abstraction level which provides the necessary flexibility in the optimisation process to define the appropriate connections.

Some of these steps are already integrated in the previous steps of the methodology (i.e. functional flow). The models elements are marked with concise stereotypes or UPDM stereotypes. Table 8 illustrates the UPDM stereotypes and their corresponding stereotypes in SysML, which are used in the concise approach for monolithic systems that was illustrated before in Table 7:

Table 8 UPDM stereotypes used for architecture optimisation

Stereotype in SysML	Stereotype in UPDM	Description
Function	Function	Denotes that the model element is used to define the SoS functions and their requirements
Technical	System (marked as a 'resource role' when used as a part in the internal block diagram)	Denotes that the model element is used to define the connections between constituent systems
mappedTo	ActivityPerformedbyPerformer	Used to indicate the mapping between functional and constituent-system layers.
allocatedTo	allocatedTo	Used to indicate the mapping between constituent systems and geographical layers
typedConnector	typedConnector	Denotes a link between the model elements (e.g. constituent systems). It represents a physical object such as a communication network

The constituent-system classes are modelled as system blocks. We also use the same block definition to define the SoS structure stereotyped as a 'System' in the block definition diagram. The optimisation goals and con-

straints are added to the SoS class that contains the SoS architecture patterns using attributes and constraints as shown in the example in Figure 4-18. In this example the optimisation objective is to minimize the SoS cost, while satisfying the constraints. The constructed system's network coverage must exceed a defined value which is added to the model as a coverage constraint.

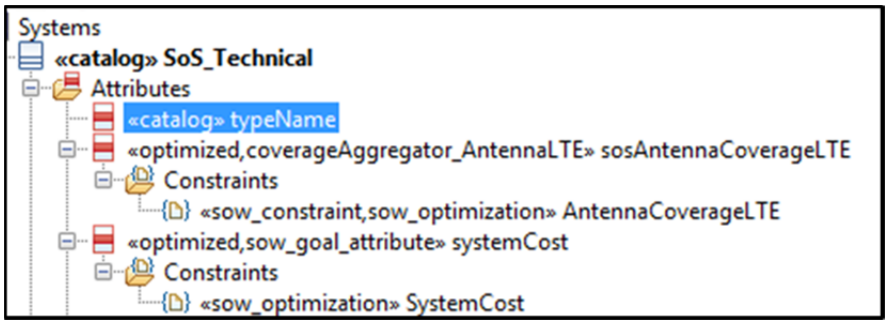

Figure 4-18 Modelling optimisation objectives and constraints

The next step of the architecture optimisation process is to generate a database that allows the developer to add the candidates of the constituent systems and their possible locations. Figure 4-19 presents an example for adding the locations of the constituent systems in the optimisation database. In this example the communication antennas must be distributed over a defined area and the optimisation problem is to achieve the best coverage area with minimum SoS cost. There are 23 possible locations for the antennas. The optimisation process must be chosen to achieve the optimum positioning for the optimisation goals within the defined constraints.

	A	B	C	D	E	F	G	H	I	J	K
1	int	int	int	int	int	int	RhpString	RhpString	int	float	float
2	id	owningPart	antennaLocationId	maxElements	x	y	type	lname	locationId	CoordinateNorth1	CoordinateEast1
3	9000000	8100000	1	2	1	9	antennaLocation	antennaLocation1	78	52,57968436	13,23777328
4	9000001	8100000	2	1	2	2	antennaLocation	antennaLocation2	68	52,498573	13,21475431
5	9000002	8100000	3	2	3	17	antennaLocation	antennaLocation3	34	52,52561012	13,16871638
6	9000003	8100000	4	2	7	13	antennaLocation	antennaLocation4	134	52,53912868	13,30683017
7	9000004	8100000	5	2	11	9	antennaLocation	antennaLocation5	207	52,52561012	13,39890603
8	9000005	8100000	6	1	12	16	antennaLocation	antennaLocation6	265	52,55264724	13,46796293
9	9000006	8100000	7	2	14	14	antennaLocation	antennaLocation7	206	52,53912868	13,39890603
10	9000007	8100000	8	2	14	4	antennaLocation	antennaLocation8	243	52,60672148	13,44494397
11	9000008	8100000	9	1	19	17	antennaLocation	antennaLocation9	322	52,53912868	13,56003879
12	9000009	8100000	10	2	19	9	antennaLocation	antennaLocation10	52	52,498573	13,19173534
13	9000010	8100000	11	2	19	9	antennaLocation	antennaLocation11	43	52,40394308	13,16871638
14	9000011	8100000	12	2	19	9	antennaLocation	antennaLocation12	88	52,44449876	13,23777328
15	9000012	8100000	13	2	19	9	antennaLocation	antennaLocation13	24	52,45801732	13,14569741
16	9000013	8100000	14	2	19	9	antennaLocation	antennaLocation14	170	52,51209156	13,3528681
17	9000014	8100000	15	2	19	9	antennaLocation	antennaLocation15	373	52,39042452	13,65211466
18	9000015	8100000	16	1	19	9	antennaLocation	antennaLocation16	387	52,4309802	13,69815259
19	9000016	8100000	17	2	19	9	antennaLocation	antennaLocation17	121	52,498573	13,28381121
20	9000017	8100000	18	2	19	9	antennaLocation	antennaLocation18	302	52,44449876	13,51400086
21	9000018	8100000	19	2	19	9	antennaLocation	antennaLocation19	278	52,5661658	13,4909819
22	9000019	8100000	20	1	19	9	antennaLocation	antennaLocation20	308	52,55264724	13,53701983
23	9000020	8100000	21	2	19	9	antennaLocation	antennaLocation21	194	52,44449876	13,37588707
24	9000021	8100000	22	2	19	9	antennaLocation	antennaLocation22	197	52,40394308	13,37588707
25	9000022	8100000	23	2	19	9	antennaLocation	antennaLocation23	188	52,52561012	13,37588707

Figure 4-19 Adding locations of constituent systems to the optimisation database

The described steps were implementing using different tools, which include IBM Rational Rhapsody, Excel and the CPLEX optimizer. Rhapsody is used for building the SoS UPDM model. A concise plugin connects the excel tables that contain the constituent-system catalogue to the UPDM model and allows the user to create an optimisation problem for the CPLEX optimizer. To solve the optimisation problem in CPLEX, the model is converted based on pre-defined templates into a Mixed Integer Linear Programming (MILP) [84] optimisation problem. The numeric parameters and the algebraic definition of the decision variables are extracted from the model and the catalogue. The concise plugin generates two files, a data file and an OPL file. Using these files the user runs the optimisation process in CPLEX and the results are back-annotated to the UPDM model as a system architec-

ture model with the specified constituent systems and their interfaces, connections, relationships, and parameters in the 'system interface description' view (SV-1).

The outputs of CPLEX are two text files containing the pattern instantiations according to the optimisation results and the values of the optimisation goals for the instantiations. The files contain information about the chosen constituent systems with their numbers and parameters. These files are converted using the source data sheet and the model data by the concise plugin into a system architecture.

If the user is not satisfied with the generated SoS architecture, the previous steps are repeated by either changing the architecture patterns or by changing the constituent-system catalogue. It is also possible to change the optimisation goals and constraints.

The structure is presented as an internal block diagram in the system view of DoDAF (SV-1) regardless of whether the SoS architecture is manually or automatically generated. The structure depicts the constituent systems and their interactions and connections.

In order to simulate the SoS architecture a behavioral model of the constituent systems is required. There are two approaches to model the behaviour of the constituent systems, namely using UPDM state charts in DoDAF (SV-10b) or using external tools such as Modelica.

The Functional Mock-up Interface standard (FMI) [75] is used for supporting the simulation. It is an interface standard for the coupling of simulation tools. The behaviour of the constituent systems is exported as a Functional Mock-up Unit (FMU). The FMU is a compressed file that contains XML data defining all exposed variables in the FMU and related static information. The XML data also includes the required model equations for the co-simulation tool and parameters for a communication module. The related FMU for each constituent system is added to the constituent-system block using a Uniform Resource Identifier (URI) which is a string that identifies the FMU. The constituent-system block is marked with the FMI stereoptype. The FMI stereotype allows the modeler to add the URI of the constituent-system behaviour as illustrated in Figure 4-20.

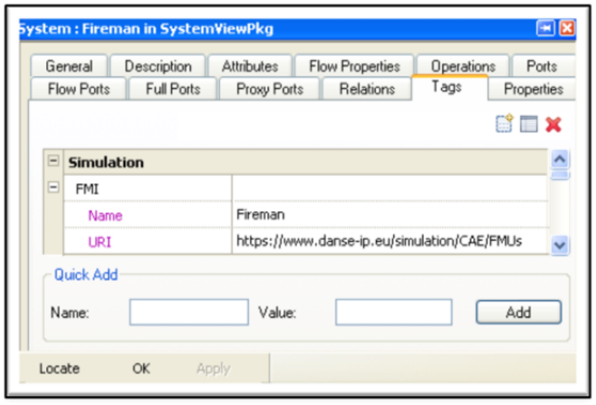

Figure 4-20 Adding the FMU URI to the constituent-system block

The SoS architecture is exported as an XMI file and imported together with the constituent-system behaviour by any simulation tool that supports FMI simulation. In Figure 4-21 a generic flow of the architecture modelling methodology is illustrated.

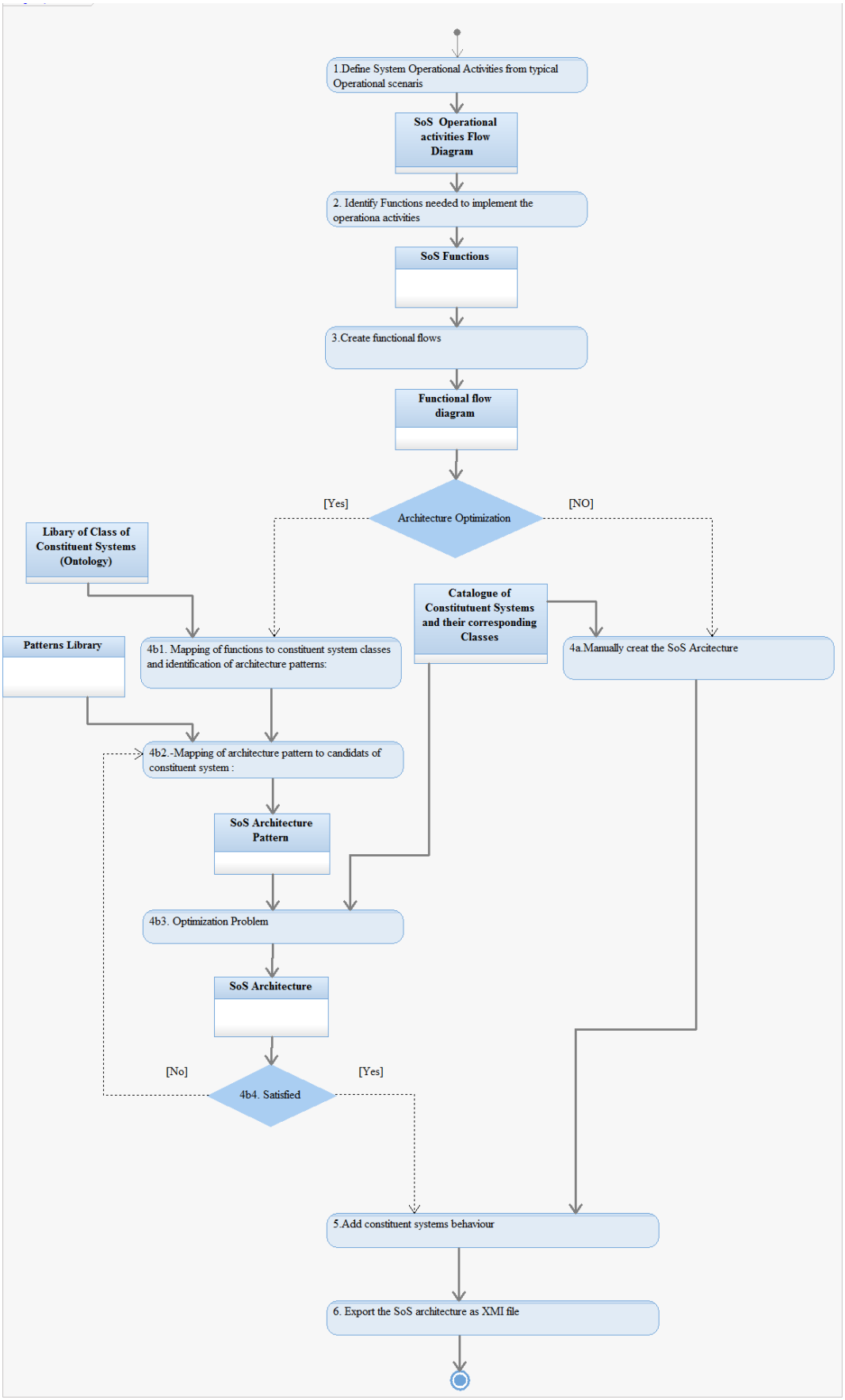

Figure 4-21 Generic architecture methodology flow

5 Timing Analysis and Optimisation

5.1 Introduction

A significant research gap of SoS development methods is the missing support for the specification of timing properties in SoS models along with algorithms for model transformation, optimisation, and timing analysis. The violation of temporal constraints may lead to a SoS failure, since many SoS applications are real-time systems, e.g., emergency response systems and military applications. It is required to synthesize a system architecture that satisfies the temporal constraints, while also optimizing the system's overall goals such as cost and weight.

This chapter illustrates how to apply SoS timing analysis in the SoS architecture development using our architecture methodology. In Section 5.2 our SoS timing analysis process is presented while considering the SoS characteristics speciality and complexity. Section 5.3 illustrates how to integrate the timing analysis in our architecture methodology and Section 5.4 describes the optimisation procedure.

5.2 SoS Timing Analysis and Model Definition

We start our analysis by considering the SoS as a hierarchy of constituent systems taking into account the specific SoS characteristics such as operational and managerial independence, evolutionary development and geographical distribution. Due to the SoS complexity, modelling and analysis of the SoS stays at a high level of abstraction. The SoS timing analysis process must consider the lack of centralised control by the SoS owner (if existent) over constituent-system functions and the restricted level of information about the constituent-system operations. However, it is still possible to create a high-level SoS operational view that leads to constituent-system functions for handling these operations.

In our approach, timing calculations are attached to each function at the level of the functional analysis [51]. The constituent system provides its functions to the SoS once it is chosen to join as illustrated in Figure 5-1.

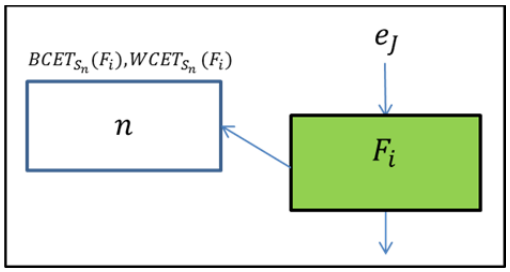

Figure 5-1 Timing properties attached to the system function

We assume that each function in each constituent system has a Worst Case Execution Time (WCET) and a Best Case Execution Time (BCET) according to the constituent-system specifications and the input events for this function. Since a linear programming optimizer (i.e., CPLEX) is used in our methodology for the optimisation process and due to linear optimisation process limitations, we describe the WCET and BCET for event-triggered constituent systems as follows:

$$BCET_{S_n}(F_i, e_j) = A_{S_n}(Fi)e_j + B_{S_n}(Fi) \quad (3)$$

$$WCET_{S_n}(Fi, e_j) = A'_{S_n}(Fi)e_j + B'_{S_n}(Fi) \quad (4)$$

Where $BCET_{S_n}$ and $WCET_{S_n}$ denote the respective execution times for the function F_i within constituent tem S_n. F_i is function number i in the SoS functional analysis, $A_{S_n}, B_{S_n}, A'_{S_n}$ and B'_{S_n} are system parameters. e_j is the function input value. For example, function input values of the communication system would be the number of placed calls.

Time-triggered systems have another formulation, where the time calculation depends on the system specification and is characterised as $\left(p_{S_n}(F_i), j_{S_n}(F_i), d_{S_n}(F_i)\right)$ with p_{S_n} denoting the activation period of the constituent system s_n, j_{S_n} is the jitter and d_{s_n} denotes the delay. The calculation of the WCET depends on whether the systems that are connected to the time-triggered system are synchronized with it. In case of synchronization the WCET is defined as follows:

$$WCET_{S_n}(Fi, e_j) = j_{S_n}(F_i) + d_{S_n}(F_i) \quad (5)$$

And for unsynchronized systems the WCET is:

$$WCET_{S_n}\left(Fi, e_j\right) = p_{S_n}(F_i) + j_{S_n}(F_i) + d_{S_n}(F_i) \quad (6)$$

The BCET is equal to the WCET of a synchronized constituent system:

$$BCET_{S_n}(Fi, e_j) = j_{S_n}(F_i) + d_{S_n}(F_i) \quad (7)$$

Based on the system functional flow diagram we define the system functional graphs $\mathcal{F}_i$. A functional graph is defined as $\mathcal{F} = \{\nu, \varepsilon\}$, where ν is the set of functions and $\varepsilon = \{(\tau_l, \tau_k)|\tau_l, \tau_k \epsilon \nu\}$ is a set of edges representing the execution dependencies [85]. The set of functions for each functional graph is defined based on the system functional flow and the precedence constraints between functions. Figure 5-2 shows an example of a functional graph.

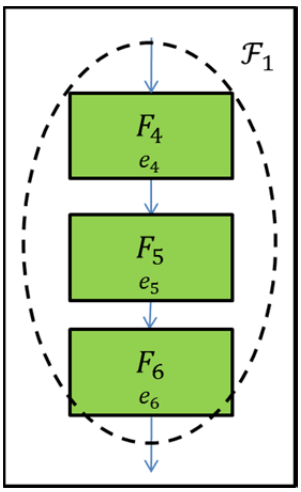

Figure 5-2 Functional graphs

The functional flow diagram is the combination of several functional graphs. The overall system time $t_{\mathcal{F}_{total}}$ is the critical path delay in the functional flow diagram according to the dependencies between the functional graphs. In case of forks or distribution nodes, the time of the functional graph with the maximum delay is considered.

In our UPDM model we distinguish two levels, namely the constituent-system level and the SoS level. At constituent-system level we create UPDM classes for constituent systems that will be referenced by the architecture patterns. Constituent systems are defined at a high level of abstraction with generalized properties and constraints. At this level the timing (i.e., BCET and WCET) of each constituent system associated with each function is added to the UPDM class as an attribute with a certain stereotype. The catalogue of stereotypes from concise modelling is used to specify the constituent systems and their parameters in the constituent-system catalogue (see Figure 5-3).

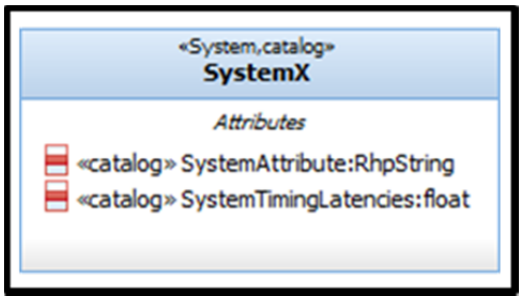

Figure 5-3 UPDM class for constituent systems

Further attributes are also added such as systemName or systemId to give unique values for constituent-system instances.

At SoS level timing constraints such as deadlines and synchronization requirements are translated manually into mathematical expressions and added as attributes with constraints to the UPDM class of a SoS. The same is applied for optimisation goals but with different marked stereotypes (e.g., 'sow_goal'). Optimisation goal calculations are expressed as system-attribute constrains and anchored to the SoS UPDM class (see Figure 5).

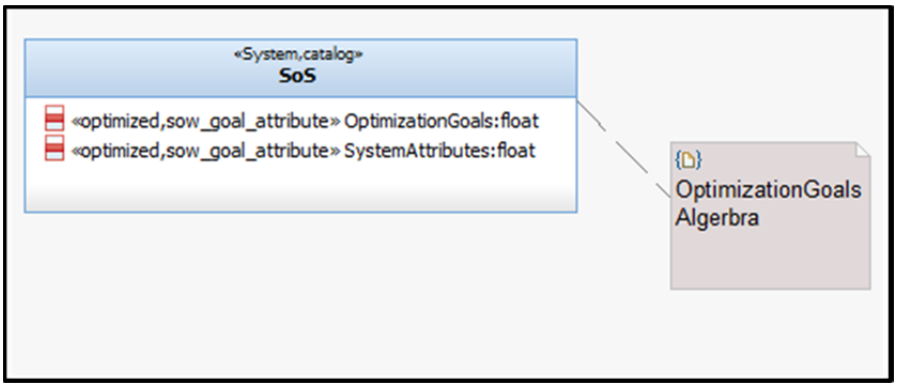

Figure 5-4 UPDM class for SoS

5.3 Integrating Timing Analysis in the Architecture Methodology

Figure 5-5 illustrates the integration of the timing analysis in the architecture methodology:

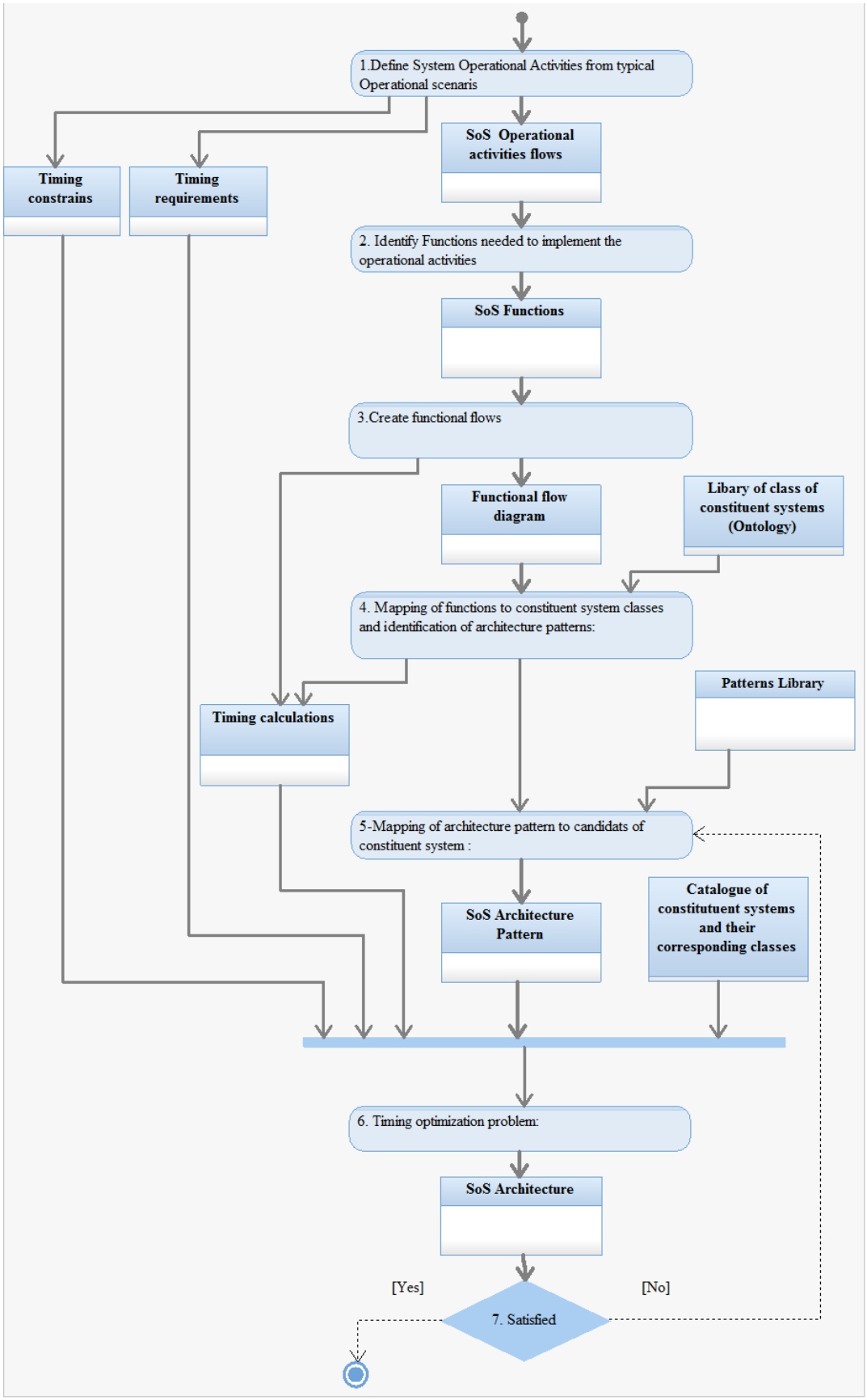

Figure 5-5 Integrating timing analysis in the architecture methodology

The first step of the methodology is the analysis of timing requirements. There are two sources of the timing requirements and constrains:

1- The system requirements: from the customer needs analysis timing requirements are identified at a high level. For example, in an emergency response system at least 8 functions (firemen, policemen, ambulance …) must be at the site 5 minutes after the emergency alert.

2- The operational scenario analysis: after building the operational scenario, timing requirements and constraints are identified for each scenario. For example, the latency of the operational activities after the first call of the emergency alert and before the issuing of the dispatching order, must be less than *n* minutes as illustrated in Figure 5-6.

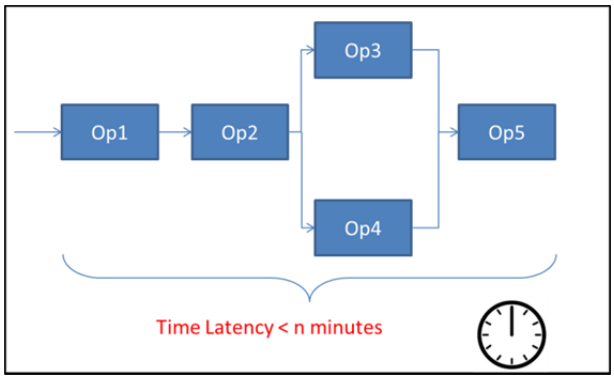

Figure 5-6 Deriving timing requirements from operational scenarios

During the functional flow identification, two attributes must be added to each function: the WCET and the BCET. Using these parameters based of the functional flow analysis and using the mapping of functions to constituent-system classes, the latency calculations are defined as depicted in Figure 5-7.

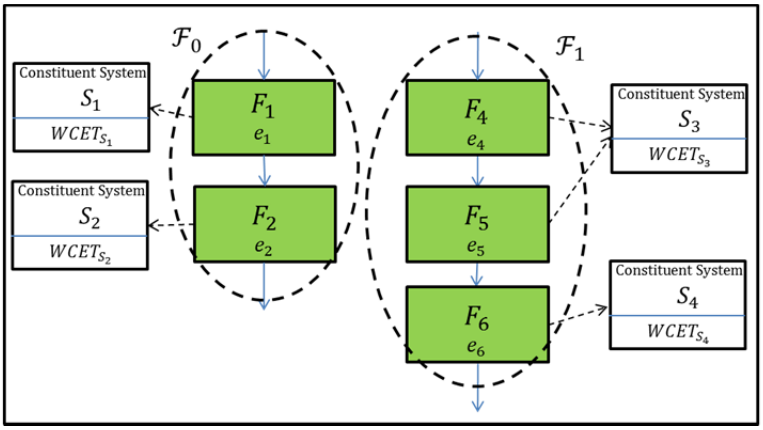

Figure 5-7 Functional graph identification

The timing calculations of the functional graphs as shown in Figure 5-7 are:

$$t_{\mathcal{F}_0} = WCET_{S_1}(F1, e_0) + WCET_{S_2}(F2, e_1) \quad (8)$$

$$t_{\mathcal{F}_1} = WCET_{S_3}(F4, e_3) + WCET_{S_3}(F5, e_4) + WCET_{S_4}(F6, e_5) \quad (9)$$

These equations are used in calculating the optimisation objectives for minimum timing latency or as optimisation constrains for the timing latencies.

5.4 Optimisation

The purpose of the optimisation is to create a SoS architecture that achieves the minimum timing latency combined with other optimisation objectives such as minimum cost. The multi-objective optimisation is done using the optimisation goal priorities. Based on the functional flow, we either import architecture patterns from a pattern library or we define new architecture patterns using the "System View SV-1" of UPDM.

As illustrated in the previous section, concise modelling stereotypes are added to the modelling elements to define the optimisation problem including the optimisation goals and the optimisation constraints.

Using the concise modelling approach a catalogue of the candidate constituent systems is generated from the architecture patterns. The WCET and BCET for each function are defined per constituent system and listed in the catalogue.

The optimisation procedure is implemented as illustrated in our methodology. The catalogue is generated based on the architecture pattern classes. The optimisation parameters (i.e. WCET, BCET, cost) are defined as attributes of the constituent-system classes and their values are taken from the catalogue.

Based on pre-defined templates, a concise modelling plugin converts the model and the catalogue information with options and parameters of constituent systems to CPLEX files. The first file contains the optimisation data (i.e. constituent-system catalogue) and the second one formalizes the optimisation problem in the Optimization Programming Language (OPL). The resulting SoS architecture, after running the optimisation in CPLEX, is then back-annotated to the model.

6 Reliability Analysis and Optimisation

6.1 Introduction

Reliability engineering as addressed in section 2.5.3 is the discipline of building reliable and maintainable systems. This is done using techniques for predicting, evaluating and demonstrating system reliability and maintainability [37]. Due to the large number of parts and connections, reliability analysis becomes more complex when it is considered in the SoS architecture analysis.

The reliability requirements are addressed at different levels of the SoS development process. In this chapter we integrate the reliability engineering process into our architecture development methodology. We apply two means for enhancing system reliability: fault avoidance and fault tolerance [86]. The goal of fault avoidance is to carefully select the type and the quality of the chosen constituent systems during the development process in order to prevent later constituent-system failures at run time. Fault tolerance aims at containing and masking failures of individual constituent systems at run time, thereby limiting their effect on the services of the SoS.

Section 6.2 illustrates how to adapt the reliability engineering steps in DoDAF architecture framework views. In section 6.3 the reliability formulation in the architecture model is illustrated. Section 6.4 shows how to use architecture patterns in reliability analysis, and section 6.5 presents the integration of the reliability analysis in our architecture modelling approach.

6.2 Reliability Block Diagram in DoDAF Architecture Framework

As presented in Section 2.5.3 the Reliability Block Diagram (RBD) is used to facilitate reliability analysis for complex systems. It is an event diagram that provides developers with the information about reliability dependencies within the system. Figure 6-1 shows how to use the DoDAF architecture views in defining the RBD.

The operational analysis is used for two purposes:

1- To elucidate the reliability requirements of the SoS

2- Analysing the SoS operations and defining the functions required for implementing the SoS operations.

The operational analysis is performed using the ‘operational activity model’ (OV-5b) in DoDAF and the operations are mapped to constituent-system functions using the ‘operational activity to systems function traceability matrix’ view (SV-5a). The functional flow diagram is modelled using the ‘system functionality description’ (SV-4) view. In the ‘system interface description’ view (SV-1), the functions are mapped to the constituent systems that implement these functions. The mapping process provides insight about constituent systems that affect the functionality of the SoS.

Using the functional flow and functions to constituent-systems mapping process as a guideline, the RBD is created according to the criteria described in section 2.5.3. The RBD is constructed using the ‘system interface description’ view (SV-1). The view’s elements are used to express the reliability dependencies between the constituent systems. Reliability attributes are added to the constituent-system block. These attributes are defined per function and are used in the reliability calculations based on the SoS functional flow.

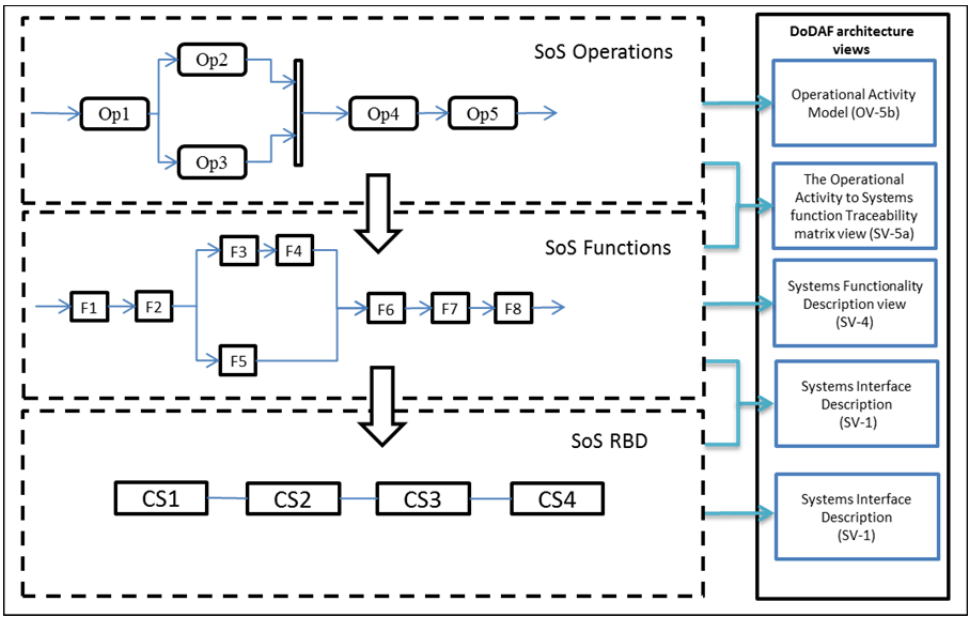

Figure 6-1 SoS reliability diagram procedure mapped to DoDAF views

6.3 Reliability Formulation in Architecture Models

Our approach is to include the reliability requirements in the architecture optimisation process. From the point of view of the dependencies between the constituent systems, different SoS structures can be distinguished for reliability optimisation [45] such as:

- Parallel–series systems: Components are connected in series and redundant components are added in parallel.
- General network systems: A constituent-system configuration with bridge networks and non-series non-parallel structures is used.

In our methodology we cover the first structure for series systems with redundant constituent systems added in parallel to enhance the system reliability.

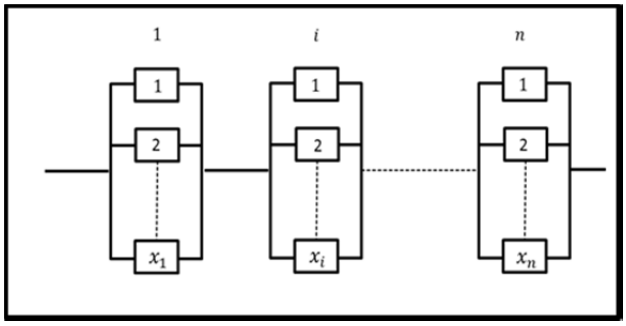

Figure 6-2 Parallel-series reliability structure

As shown in Figure 6-2, a SoS consists of multiple constituent systems that are connected in series with redundant constituent systems in parallel. It is assumed that the failure mode of the constituent systems is fail-silent (e.g., established by self-checking and local diagnosis) [33]. If R_i is the reliability of the i-th constituent system with $0 \leq R_i \leq 1$ and x_i expresses the redundancy degree for the i-th constituent system with $x_i \geq 1$, then the systems reliability R_s is [87]:

$$R_s = \prod_{i=1}^{n} [1 - (1 - R_i)^{x_i}] \tag{10}$$

The optimisation goal of the systems is:

$$\begin{aligned} &\max \prod_{i=1}^{n} [1 - (1 - R_i)^{x_i}] \\ &\text{s.t.} \\ &g_j(x_i) \leq b_j \quad \forall j \in \{1, \dots, n\} \\ &x_i \geq 1 \quad \forall i \in \{1, \dots, n\} \end{aligned} \tag{11}$$

Where $g_j(x_i)$ are the optimisation constraints and b_j is a vector of integer values.

This kind of optimisation problem is a nonlinear integer programming problem and algorithms typically cannot guarantee convergence to the global optimum within a reasonable time. The goal of our modelling approach is to facilitate the problem formulation for the system architect in an efficient manner. For this purpose, we reduce the complexity of the optimisation problem formulation. We can reformulate the problem by converting it into a linear optimisation problem by two steps:

- Use of logarithm:

$$\begin{aligned} &\max \sum_{i=1}^{n} \ln(1 - (1 - R_i)^{x_i}) \\ &s.t. \\ &\ln g_j(x_i) \leq \ln b_j \\ &b \neq 0 \end{aligned} \tag{12}$$

- Pre-calculated values: The values of $(1 - (1 - R_i)^{x_i}$ can be provided as discrete values using external calculation tools. The output of the optimisation process includes the constituent-system types and their redundancy degree. In concise modelling the input catalogue for the optimisation process is provided through a spreadsheet, were the instances of the constituent-system classes and their property values are listed. We consider the redundancy degree as one of the properties of the system and the reliability for each redundancy degree will be calculated in the spreadsheet and provided as a discrete value.

6.4 Reliability Optimisation and Architecture Patterns

In our reliability analysis approach we start by building the RBD. We formulate the optimisation problem based on the RBD and we construct the corresponding architecture patterns that will be used in the optimisation process (cf. Figure 6-3).

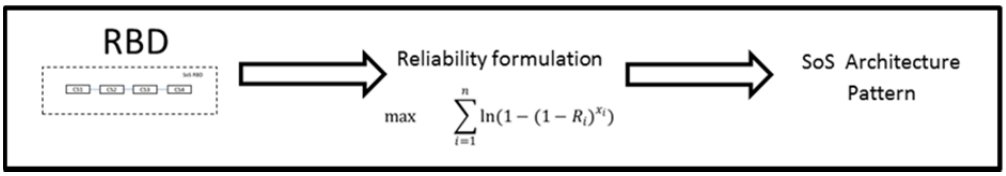

Figure 6-3 Processing steps for reliability optimisation and architecture patterns

The architecture patterns are used as guidance for the optimisation process on how to connect the constituent systems and their connections in the SoS architecture. An architecture pattern indicates the purpose of the architecture and the flexibility of choosing the constituent systems within the architecture constraints. The architecture patterns facilitate SoS evolution to meet new requirements and enhance the SoS architecture reusability when modelling new systems with the same functionality.

In our work we use architecture patterns for SoS reliability optimisations. The architecture pattern is built to allow reliability enhancements of the system either by fault avoidance (i.e., constituent system selection), fault tolerance (i.e., redundancy degree) or both. In fault avoidance we manipulate the quality and the type of the constituent systems according to the available constituent-system catalogue. In this approach the architecture pattern is built out of constituent-system classes that allow the developer to optimize which constituent-system types best match the system's reliability requirements. In fault-tolerance patterns the architecture pattern allows for system redundancy. It provides the system architect with the ability to optimize the redundancy degree for each of the constituent systems as required to achieve the target reliability. It is also possible to use patterns that consider both fault avoidance and fault tolerance in the reliability optimisation process.

6.5 Integrating Reliability Analysis in the Architecture Methodology

In the previous sections we presented the reliability analysis process using the DoDAF architecture framework and architecture patterns. In this section we present the integration of the reliability analysis into our architecture methodology. We support reliability in the architecture optimisation process and we generate an SoS architecture that satisfies the reliability requirements and constraints. We illustrate how to use the methodology steps to construct the RBD and how to use the data extracted from this diagram in the architecture optimisation problem.

The operational and functional analysis is an important step in defining the RBD. The process can also be integrated with timing analysis. The reliability calculations are added to the optimisation problem and using the constituent-system database the optimisation problem is converted to a linear optimisation problem. Figure 6-4 illustrates the additional steps to include the reliability analysis in our architecture methodology.

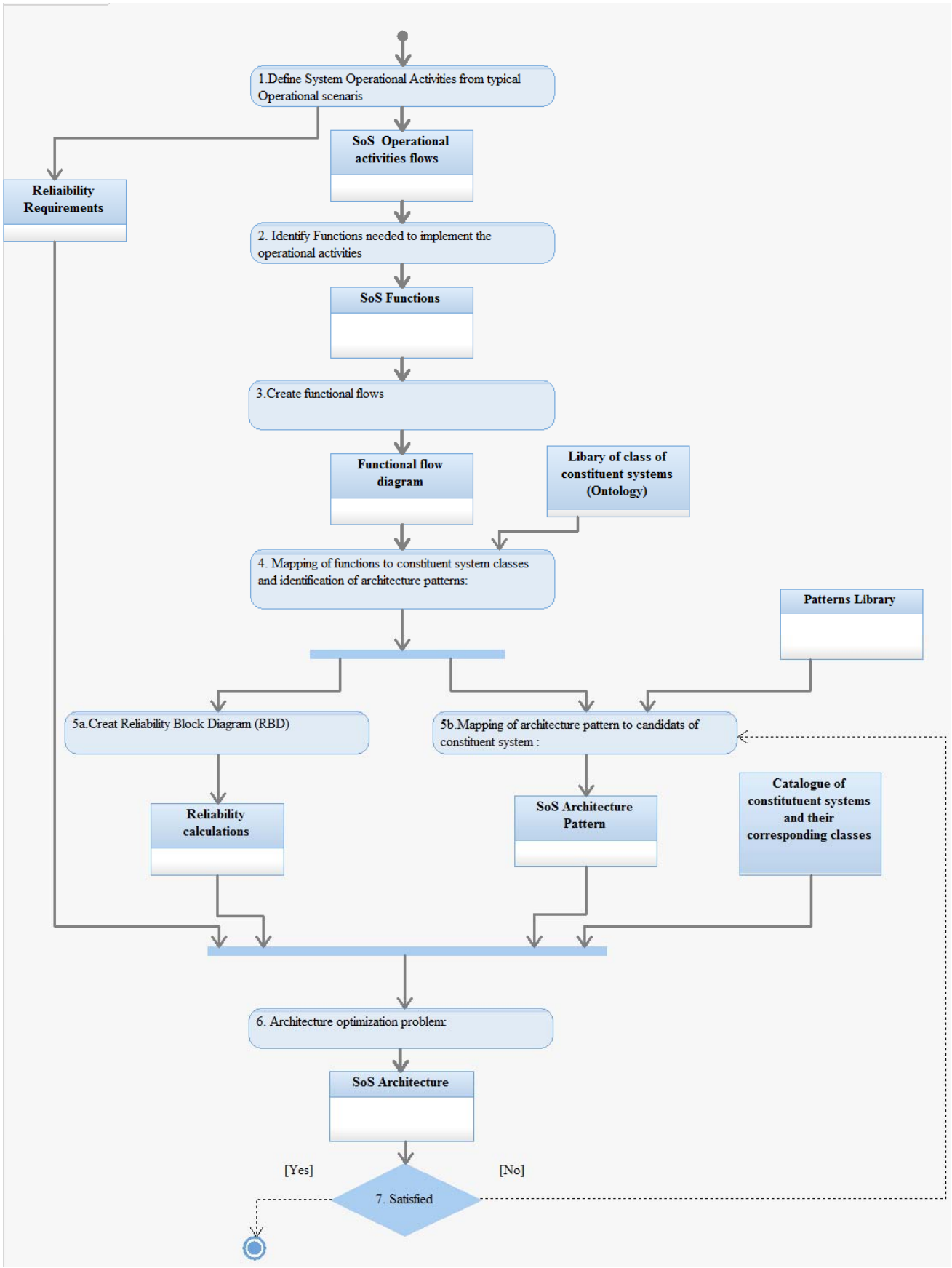

Figure 6-4 Integrating reliability analysis in the architecture methodology

At the operational level, the SoS requirements are analysed and reliability requirements are elucidated. Reliability requirements are considered as extra-functional requirements that assure the system's ability to perform the correct services with a high probability. The target value of system reliability is determined according to the system requirements. During the design process these reliability values are included in the optimisation and system development constraints. At the operational level the customer demands a certain reliability for each scenario or operation. The required reliability is then mapped to the operational activities and considered as a reliability constraint for the design.

At the next stage the process continues with the methodology steps for building the functional flow diagram and mapping the functions to constituent-system classes.

The fifth step is split into two parallel activities: choosing the architecture pattern that guides the architecture optimisation process and building the corresponding RBD based on the functional mapping and the chosen architecture pattern.

From the RBD the reliability matrices are extracted and mapped to the corresponding architecture pattern. These matrices are used either as optimisation constrains or optimisation objectives in the optimisation process. The catalogue database is generated using a concise plugin where the candidate constituent systems are added according to their corresponding classes. The redundancy values are added in the database as illustrated in Table 9:

Table 9 Adding reliability values to the constituent-system catalogue

int	float	RhpString	int	RhpString	Float
id	cost	name	systemId	typeName	Reliability
1200000	10	SB11	1	SB11	-0,020202707
1200001	20	SB12	2	SB12	-0,00040008
1200002	30	SB13	3	SB13	-8,00003E-06
1200003	40	SB14	4	SB14	-1,6E-07
1200004	50	SB15	5	SB15	-3,2E-09
1200005	60	SB16	6	SB16	-6,4E-11
1200006	8	SB21	7	SB21	-0,051293294
1200007	15	SB22	8	SB22	-0,00250313
1200008	20	SB23	9	SB23	-0,000125008
1200009	25	SB24	10	SB24	-6,25002E-06
1200010	30	SB25	11	SB25	-3,125E-07
1200011	35	SB26	12	SB26	-1,5625E-08
1200012	37	SB27	13	SB27	-7,8125E-10
1200013	40	SB28	14	SB28	-3,90625E-11
1200014	42	SB29	15	SB29	-1,9531E-12

SXnr is used to define the constituent-system class, type and number where *X* is the constituent-system class (in this example it is B), *n* is the constituent-system type and *r* is the redundancy degree. In this example the constituent-system class B has two options of instances (i.e. B1 and B2). The first type is with redundancy degrees up to 6 and the second type is with redundancy degrees up to 9. We use the equation for the reliability calculations (eq.7) in the data base.

The redundancy is realised in the architecture pattern by indicating in the constituent-system class that the number of constituent-system instances is limited to one and the variation of the redundancy degree is chosen from the catalogue table.

7 Examples and Results

Three examples were implemented to verify and evaluate our architecture methodology. The first example solves the problem of SoS evolution. This was done by considering the installation of new antenna communication systems in an emergency response SoS. The problem considers the geographical distribution of the constituent systems of the SoS and optimizes the antenna coverage and cost.

The second example illustrates how to include temporal requirements in the architecture design methodology. It presents a SoS example with timing and cost requirements that must be satisfied in the architecture optimisation process.

The third example covers the reliability requirement in the SoS architecture design. An example with reliability requirements is illustrated with detailed steps for the methodology implementation.

The results for each example were verified by building a corresponding mathematical model in Matlab.

7.1.1 Antenna Allocation of Emergency Response Communication System

7.1.1.1 Problem Definition

The data exchange between the constituent systems of the Emergency Response (ER) SoS depends on the communication links between them. The communication services and quality affects the performance of the ER constituent systems and the overall ER SoS performance. The quality of the communication depends on the type of the constituent systems that are used, the service provider and the communication coverage.

For our example we will consider the optimisation of the communication coverage for the ER SoS of Berlin City (see Figure 7-1).

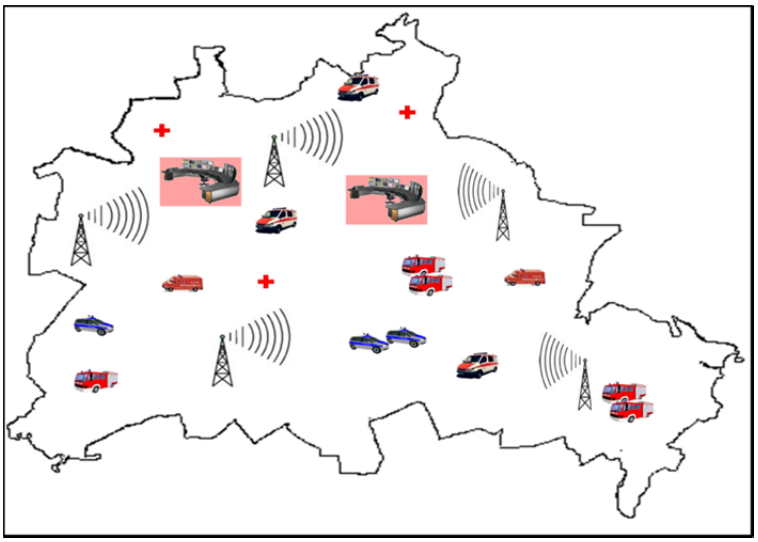

Figure 7-1 Communication coverage optimisation use case

The ER works at different locations. The communication antennas must be distributed in different geographical places across the city in order to achieve the best coverage area. At the same time the other constituent systems are not at fixed places and they change their locations according to the emergency case requirements. Beside the geographical placement of the antennas, the communication coverage also depends on:

- Radiation power level
- Antenna height
- Antenna Patterns
- Polarization
- Frequencies
- Service-dependent parameters

The previous points can be used as optimisation parameters that need to be optimized in combination with the geographical places of the communication system's antennas.

The ER SoS uses a dedicated communication system for its operation. In this study case we introduce a new communication system to the SoS as an evolution of the ER SoS. We investigate the best allocation of the antennas for the newly introduced communication system while integrating it with the old communication system.

7.1.1.2 Modelling and Calculations

First we construct an operational scenario that depicts the required operational activities and describes the purpose of the constructed SoS. As shown in Figure 7-2, the operational scenario starts with sending the data that will be received first by the nearest communication antenna. The transmitted signal will then be re-directed to the receiver address that receives the data.

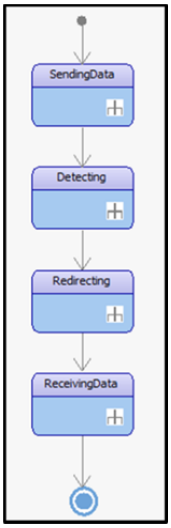

Figure 7-2 Operational activity flow diagram for antenna allocation example

From the operational scenario and the SoS requirements analysis, the SoS communication system optimisation goals and constrains are identified:

- Coverage: Achieve the best coverage area that serves the emergency response system
- Cost: Using antennas to achieve the best coverage area is constrained by the cost. We need to lower the procurement cost of the antennas.
- Quality: In addition to the optimized coverage with a minimum cost we also need a good communication quality of the communications systems.

The second level is the functional level. We define the functions that handle the operational activities and map them to these activities as illustrated in Figure 7-3. Using the mapping results and the operational activity flow we build the functional flow diagram (see Figure 7-4).

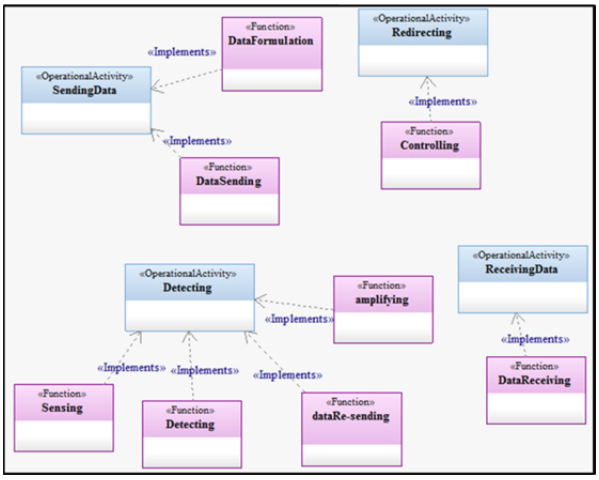

Figure 7-3 Functions to operational activities mapping for antenna allocation example

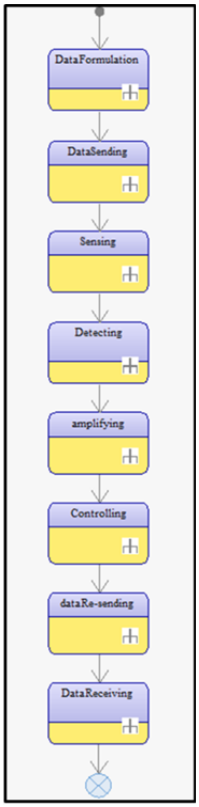

Figure 7-4 Functional flow diagram for antenna allocation example

The functional analysis is used to indicate the required connections between the constituent systems. By analysing the SoS functions and the required optimisation process an architecture pattern is constructed. The architecture pattern contains the constituent-system classes and any other required classes for the optimisation.

Using the functional flow diagram and the requirements analysis, an architecture pattern is developed that contains the required information about the involved constituent-system classes and their connections. As shown in Figure 7-5 different classes were used to represent the constituent systems.

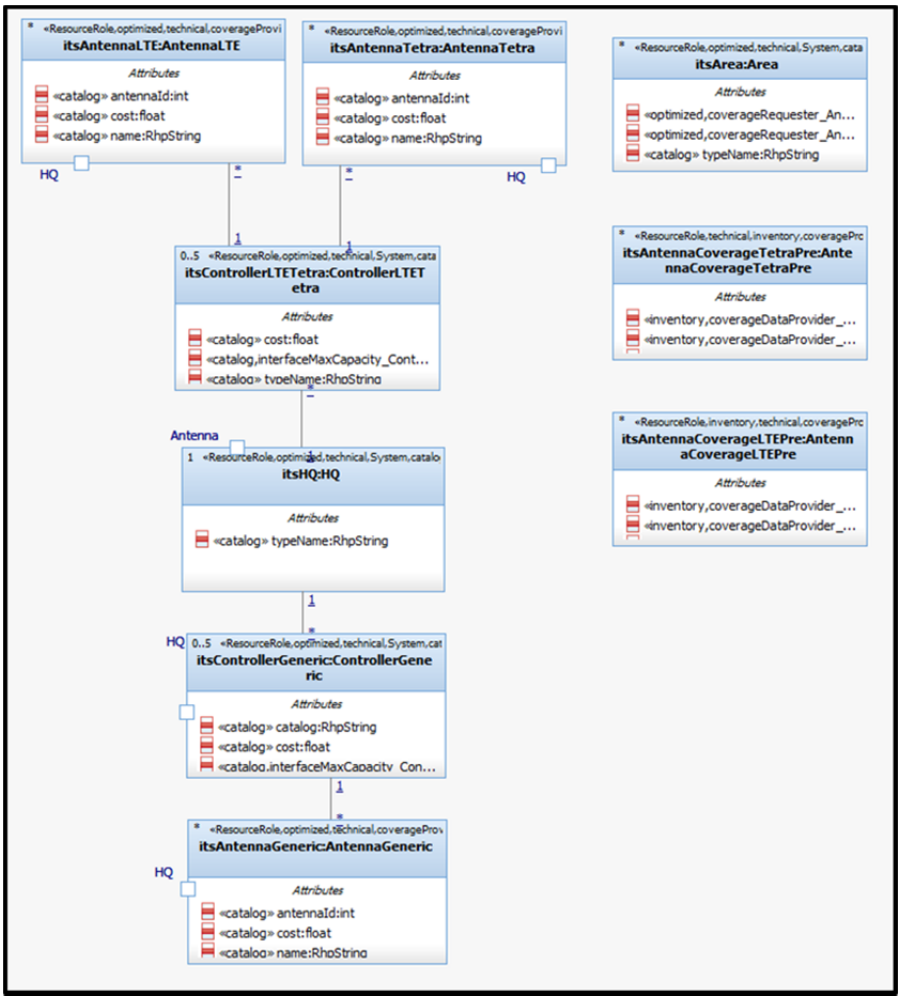

Figure 7-5 Architecture pattern for antenna allocation example

The first three classes, AntennaLTE, AntennaTetra, and AntennaGeneric represent the communication system's antennas that will be optimized. The AntennaGeneric is used to indicate the antennas that are used for both LTE and Tetra communication systems. The antenna height, the antenna type and the antenna location are the modifiable parameters of the antenna that affect the antenna coverage and cost in the optimisation process.

We assume that the antennas are controlled by antenna controllers which are located at the ER headquarter (HQ). There are two classes of antenna controllers, namely ControllerLTETetra that controls LTE and Tetra antennas, and ControllerGeneric that controls only the generic antennas. The HQ class represents the headquarter with a fixed location.

The locations of the constituent systems are included in the optimisation calculations. To identify the required information, i.e. locations coordinates and identifiers, a location class is defined for each constituent-system class. These blocks are then mapped to the corresponding constituent-system classes as show in Figure 7-6.

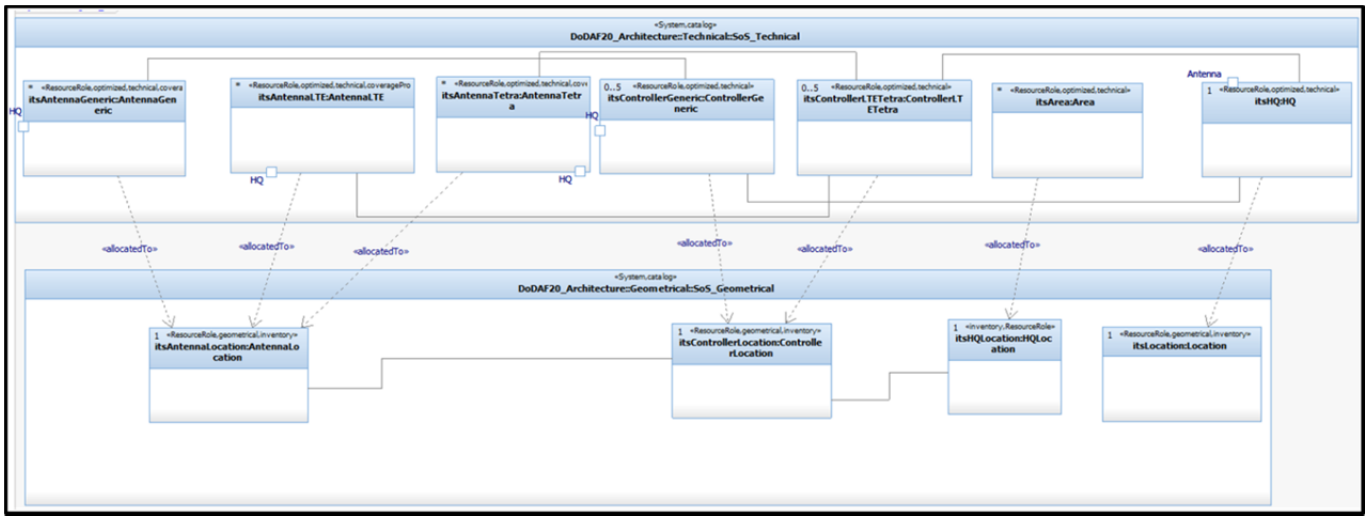

Figure 7-6 Mapping of geographical locations to constituent-system classes

The optimisation goals and constraints are added to the model using concise stereotypes. We simplify the optimisation problem for the coverage calculations by implementing the following steps:

1. The investigated area (Berlin) was divided into grid points using a Matlab script. The number and the density of the grid points define the calculation precision. The grid points are shown in Figure 7-7.

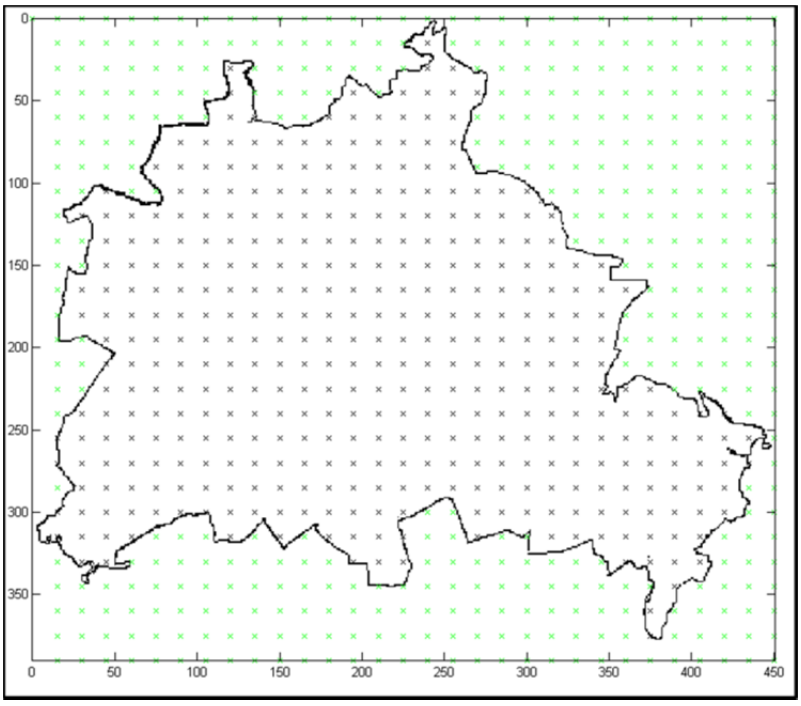

Figure 7-7 Map of Berlin divided into grid points

The data of all points is added to the model using the catalogue database that is generated by the concise plugin based on the location block definition as illustrated in Table 10.

Table 10 Adding area grid points to the model

int	Int	int	int	RhpString	RhpString	Int
id	owningPart	x	y	type	name	locationId
8200000	8100000	1	1	Location	Location1	1
8200001	8100000	1	2	Location	Location2	2
8200002	8100000	1	3	Location	Location3	3
8200003	8100000	1	4	Location	Location4	4
8200004	8100000	1	5	Location	Location5	5
8200005	8100000	1	6	Location	Location6	6
8200006	8100000	1	7	Location	Location7	7
8200007	8100000	1	8	Location	Location8	8
8200008	8100000	1	9	Location	Location9	9
8200009	8100000	1	10	Location	Location10	10

The possible places for the antenna positioning are listed with their coordinates. For each of the locations the radius of the antenna's coverage for each type and height of the antenna is calculated using the Matlab script. Thereafter, the radius of the covered area is converted to covered grid points for each type and height of the antenna options.

The antenna coverage can be calculated according to [88] using the following equation:

$$\mathrm{P_{Rx}(dBm)} = \mathrm{EiRP_{Tx}} - \mathrm{LMASK}(\theta, \varphi) - \mathrm{Lp} \tag{13}$$

where:

$\mathrm{P_{Rx}(dBm)}$ is the received power in dBm.

$\mathrm{EiRP_{Tx}}$ is the maximum Effective Isotropic Radiated Power of the cell in dBm (at the peak gain point of the antenna).

LMASK(θ,φ) is the antenna mask loss value for azimuth and elevation angles respectively in the direction of the path being calculated in dB. When the received signal is directly on the main beam of the antenna, this value will be zero.

Lp is the path loss in dB.

In order to calculate the path loss, the following equation is used:

$$\begin{aligned}&\text{Path Loss (dB)}\\&= \mathrm{k1} + \mathrm{k2}\log(\mathrm{d}) + \mathrm{k3}(\mathrm{Hms}) + \mathrm{k4}\log(\mathrm{Hms}) + \mathrm{k5}\log(\mathrm{Heff}) + \mathrm{k6}\log(\mathrm{Heff})\log(\mathrm{d}) + \mathrm{k7Diffn}\\&+ \mathrm{C_{loss}}\end{aligned} \tag{14}$$

where:

d is the distance from the base station to the mobile station (m).

Hms is the height of the mobile station above the ground (m).

Heff is the effective base station antenna height (m).

Diffn is the diffraction loss calculated using Epstein, Peterson, Deygout or Bullington equivalent knife edge methods.

k1 and k2 represent the intercept and slope. These factors correspond to a constant offset (in dBm) and a multiplication factor for the log of the distance between the base station and the mobile station.

k3 is the mobile antenna height factor. It is a correction factor used to take into account the effective mobile antenna height.

k4 is the Okumura-Hata multiplying factor for Hms.

k5 is the effective antenna height gain. This is the multiplication factor for the log of the effective antenna height.

k6 is the Okumura-Hata type multiplying factor for log (Heff) log (d).

k7 is a multiplying factor for the diffraction loss calculations.

C is the clutter specification where heights and separation are also taken into account in the calculation.

The coverage data is added to the model data-base as shown in the example in Table 11.

Table 11 Example of adding the antenna coverage data to the model

int	Int	Int	int	int	int	float
id	owningPart	c_id	antennaLocationId	antennaId	locationId	coverage
1700000		1100000	1	1	15	1
1700001		1100000	1	1	62	1
1700002		1100000	1	1	78	1
1700003		1100000	1	1	97	1
1700004		1100000	1	1	115	1
1700005		1100000	1	1	131	1
1700006		1100000	1	1	148	1
1700007		1100000	1	1	16	1
1700008		1100000	1	1	31	1

For each antenna type (indicated by antennaId) and the possible locations (antennaLocationId) the covered grid points (locationId) with the covered percentage (1 indicates 100%) are listed. This process is done for all antenna types and all possible locations.

The optimisation goals are added as an attribute to the SoS class and attached by a 'constraint' that contains the calculation formula. In Figure 7-8, three optimisation goals were added to the model: minimize the system cost, maximize the LTE coverage, and maximize the Tetra coverage. The equations that are used to calculate the optimisation goals are added as OPL code in the constraints part. For example the following expression is used to calculate the SoS cost:

'*systemCost = sum (n in attrSet_cost_float) isChosen[n] * cost[n]*'

(the SoS cost is the sum of the chosen constituent-system cost)

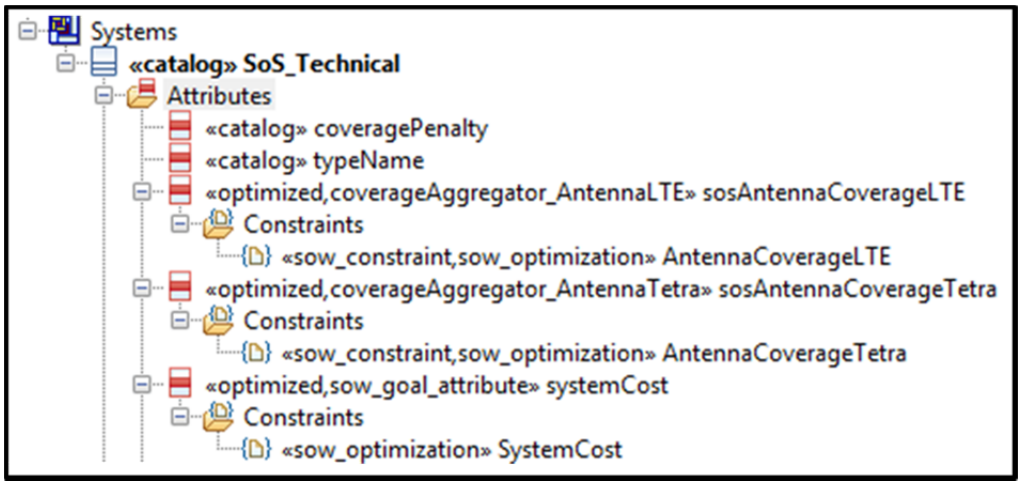

Figure 7-8 Adding the optimisation goals to the model for the antenna coverage example

7.1.1.3 Results

Using the concise plug in, the optimisation problem is translated into OPL code to be solved by CPLEX using two optimisation files: the first file contains the optimisation problem and the second file contains the catalogue data. The optimisation problem file contains the optimisation goals, formulas, constraints, and variables that are expressed in the model. The catalogue contains the antenna data (i.e. types, cost, coverage, locations). By solving the optimisation problem CPLEX generates the SoS architecture solution as a text file with the chosen constituent systems. Using the concise plug, the solution is imported into the model and presented as a SoS architecture as show in Figure 7-9 .

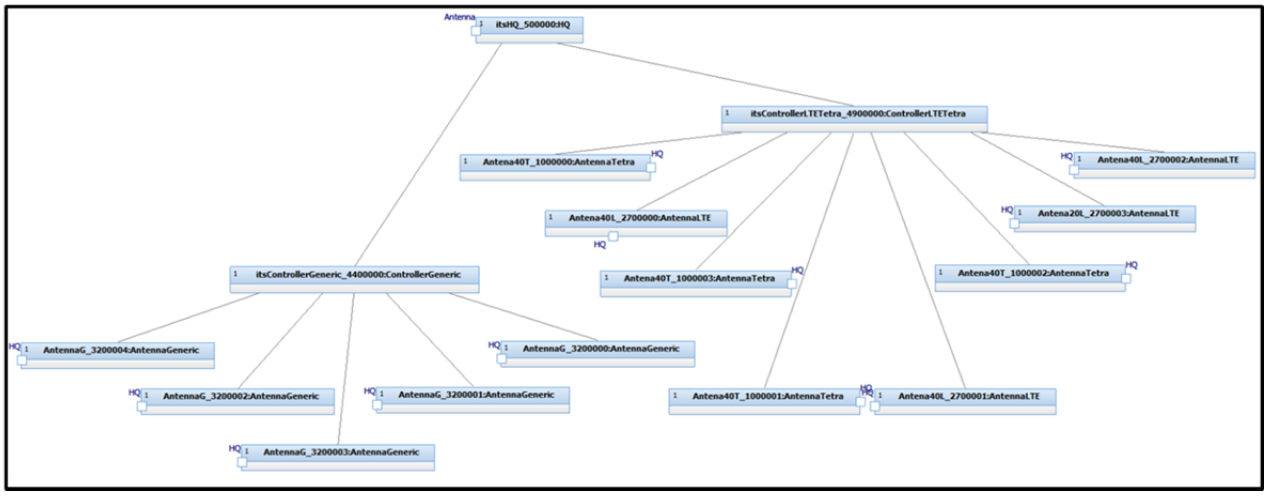

Figure 7-9 SoS antenna coverage solution

The resulting solution achieves a coverage area of 98.5 %. To verify the results we used alternative implementation, which is only feasible for a small problem size, using a Matlab model. The results were plotted in Figure 7-10 and Figure 7-11.

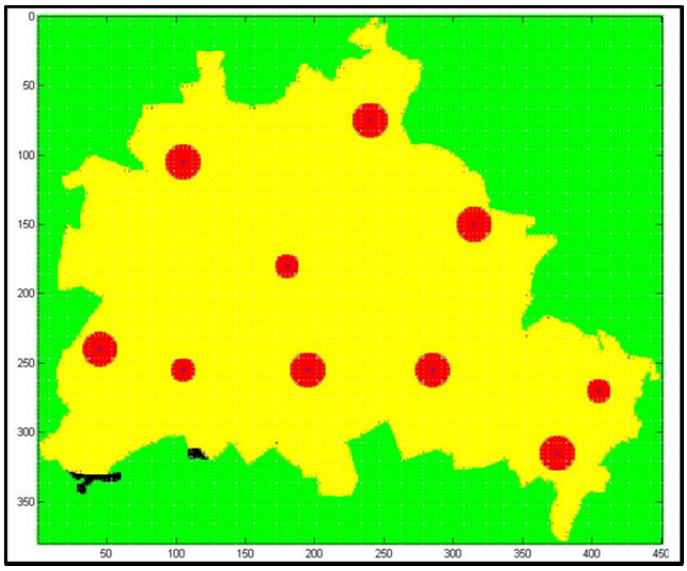

Figure 7-10 Matalb results for LTE coverage solution

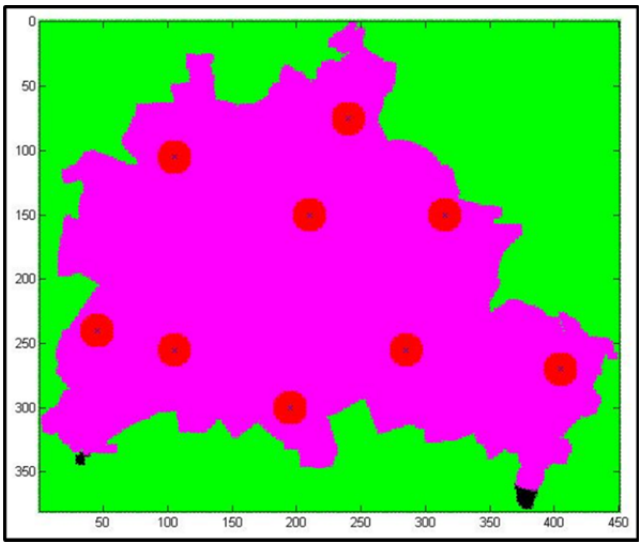

Figure 7-11 Matlab results for Tetra coverage solution

The covered area is indicated in yellow and magenta indicates the covered area, while black represents the uncovered area.

7.1.2 Real-Time Analysis

7.1.2.1 Problem Definition

In this example we illustrate how to apply the timing analysis steps in the SoS modelling. The purpose of the process is to generate an optimized SoS architecture based on timing and cost requirements and constraints. Figure 7-12 shows an example model to illustrate the concept. The SoS architecture was considered to have two types of constituent systems: time-triggered and event-triggered systems.

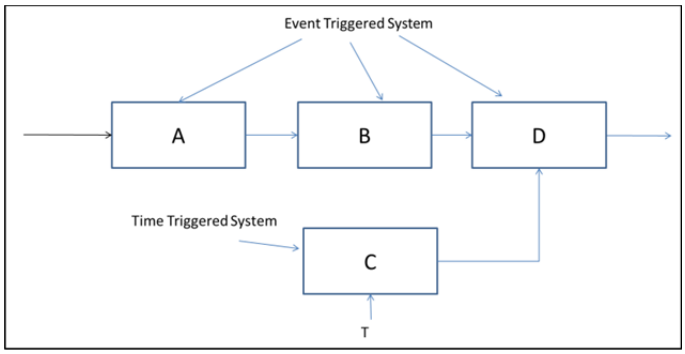

Figure 7-12 Timing analysis example model

The constituent systems A, B, and D are event-triggered systems, while constituent system C is a time-triggered system. The processing starts when an event is received at constituent system A and sent to constituent system B. The event will be processed through constituent systems B and D with input from constituent system C. The system's functional flow and their mapping to the constituent systems are depicted in Figure 7-13.

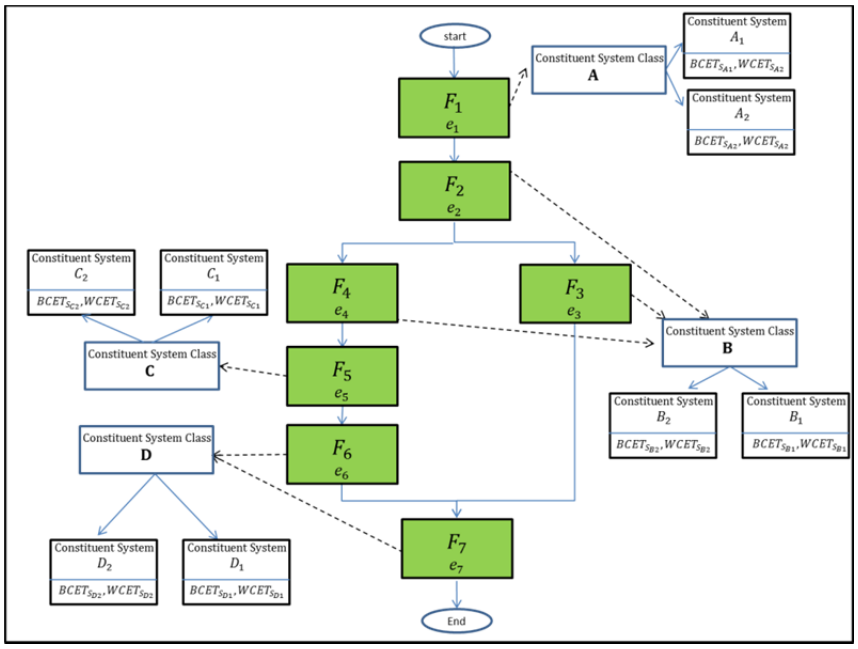

Figure 7-13 Simplified functional flow

F_1 is implemented by constituent system A, whileF_2, F_3 and F_4 are implemented by constituent system B. F_5 is implemented by constituent system C,$F_6$and F_7 are implemented by constituent system D. It is assumed that there is no functional interference between different functions on the same constituent system, so that the functions are executed sequentially and without interruption of other functions.

7.1.2.2 Modelling and Calculations

We first identified the operational scenario in Figure 7-14. It describes at a high level of abstraction the system operational activities flow. The operational activity flow is depicted in the operational activity model view (OV-5b).

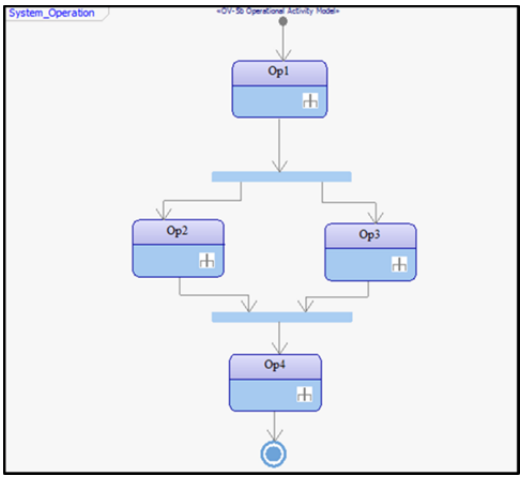

Figure 7-14 Operational activity flow diagram for the timing example

At this level we define the SoS requirements that will be used as optimisation objectives and constraints for building the SoS architecture. In this example the following requirements of the SoS were defined:

1- The timing latency from the starting operation to the end operation must not exceed *x* time units

2- The timing latency from the starting operation to the end operation must be minimized

3- The cost of the SoS implementation that is used to perform the operational activities must be minimized.

Using SV-5 "Operational Activity to Systems and Systems Function Mapping" each of the operational activities is mapped to the functions that handle the activity as illustrated in Figure 7-15. The purpose of such a mapping is to lower the level of abstraction from the operational level – which is comprehensible to the customer and the system architect – to the level of constituent systems and functions that is comprehensible to the developers.

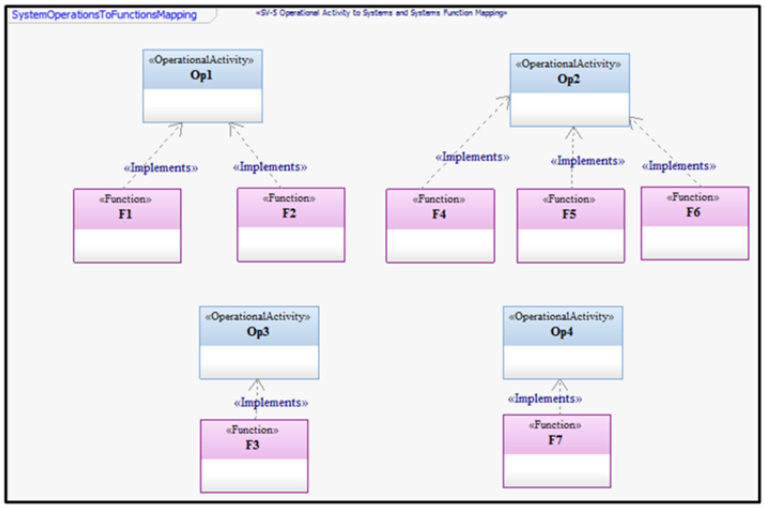

Figure 7-15 Operational activity to function mapping for timing example

Out of the operational activities to functions mapping process, the system functional flow is defined (cf. Figure 7-16). At the system functional flow the functions with their order and connections are defined. This step is used for the timing analysis process and to derive the functional graphs.

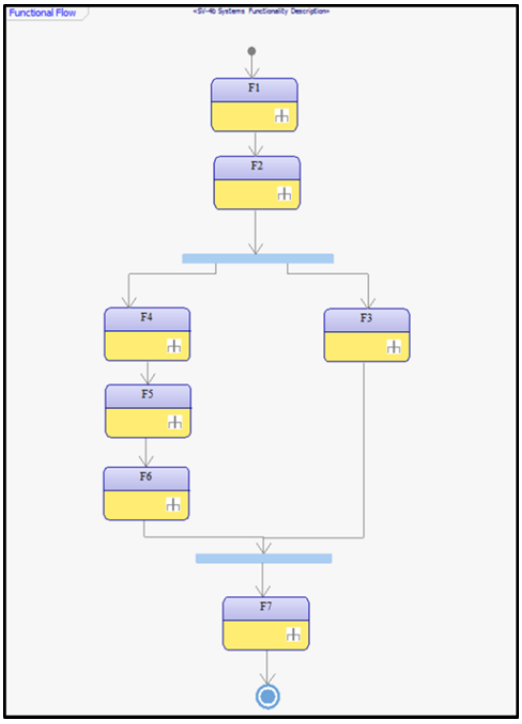

Figure 7-16 Timing example functional flow diagram

The BCET and WCET for each function are as follows:

$$BCET_{S_B}(F_2, e_1) = A_{S_B}(F_2)e_1 + B_{S_B}(F_2)e_1 \quad (15)$$

$$BCET_{S_B}(F_4, e_2) = A_{S_B}(F_4)e_2 + B_{S_B}(F_4)e_2 \quad (16)$$

$$BCET_{S_B}(F_6, e_5) = A_{S_B}(F_6)e_5 + B_{S_B}(F_6)e_5 \quad (17)$$

$$BCET_{S_c}(Fi, e_j) = j_{S_c}(F_4) + d_{S_c}(F_4) \quad (18)$$

$$BCET_{S_B}(F_7, e_6, e_3) = A_{S_B}(F_7)e_6 + B_{S_B}(F_7)e_6 + H_{S_B}(F_7)e_6 + C_{S_B}(F_7)e_6 \quad (19)$$

$$WCET_{S_B}(F_2, e_1) = A'_{S_B}(F_2)e_1 + B'_{S_B}(F_2)e_1 \quad (20)$$

$$WCET_{S_B}(F_3, e_2) = A'_{S_B}(F_3)e_2 + B'_{S_B}(F_3)e_2 \quad (21)$$

$$WCET_{S_B}(F_4, e_2) = A'_{S_B}(F_4)e_2 + B'_{S_B}(F_4)e_2 \quad (22)$$

$$WCET_{S_c}(F5, e_4) = p_{S_c}(F_4) + j_{S_c}(F_4) + d_{S_c}(F_4) \quad (23)$$

$$WCET_{S_D}(F_6, e_5) = A'_{S_B}(F_6)e_5 + B'_{S_B}(F_6)e_5 \quad (24)$$

$$WCET_{S_D}(F_7, e_6, e_3) = A'_{S_B}(F_7)e_6 + B'_{S_B}(F_7)e_6 + H'_{S_B}(F_7)e_3 + C'_{S_B}(F_7)e_3 \quad (25)$$

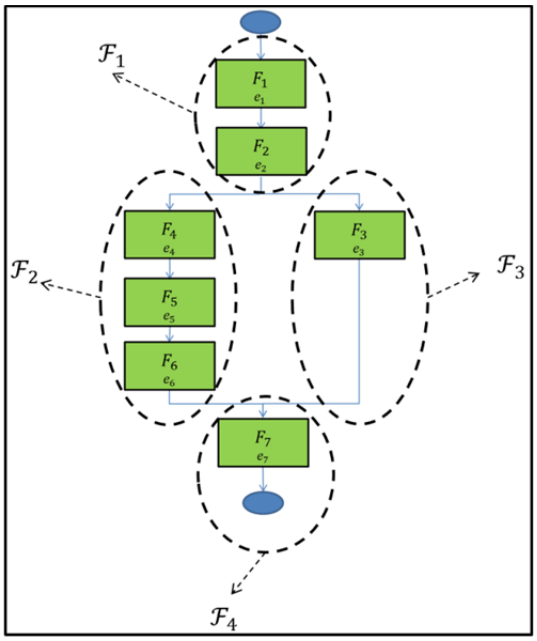

Figure 7-17 Functional graphs of timing example

For the timing latency calculations to proceed accurately, the functional flow is distributed to functional graphs as shown in Figure 7-17. We defined four timing graphs based on the functional flow and dependencies. Based on these graphs the system time calculations can be performed:

$$\mathrm{t}_{\mathcal{F}_{\mathrm{total}}} = \mathrm{t}_{\mathcal{F}_1} + max(\mathrm{t}_{\mathcal{F}_2}, \mathrm{t}_{\mathcal{F}_3}) + \mathrm{t}_{\mathcal{F}_4} \quad (26)$$

$$BCET_{total} = BCET_{\mathrm{t}_{\mathcal{F}_1}} + \max\,(BCET_{\mathrm{t}_{\mathcal{F}_2}}, BCET_{\mathrm{t}_{\mathcal{F}_3}}) + \mathrm{BCET}_{\mathrm{t}_{\mathcal{F}_4}} \quad (27)$$

$$\mathrm{WCET}_{\mathrm{total}} = \mathrm{WCET}_{\mathrm{t}_{\mathcal{F}_1}} + \max\,(\mathrm{WCET}_{\mathrm{t}_{\mathcal{F}_2}}, \mathrm{WCET}_{\mathrm{t}_{\mathcal{F}_3}}) + \mathrm{WCET}_{\mathrm{t}_{\mathcal{F}_4}} \quad (28)$$

The next step is to map these functions to the constituent systems that will implement these functions. This step will connect the functional level to the constituent-system level. The constituent-system classes are used for the mapping process:

- Constituent system class A is mapped to function F1
- Constituent system class B is mapped to functions F2, F3 and F4
- Constituent system class C is mapped to function F5
- Constituent systems class D is mapped to functions F6, F7

From that level we define the architecture pattern. The pattern describes how the constituent systems are connected to each other. It is important to define which constituent systems will interact with each other, and the type and medium of the information exchange. In our example we added a source event system, which represents the environment that triggers the first event occurrence. This data is later used in the optimisation constraints to

verify that the optimized solution is valid for all types of events. The architecture pattern is derived from the system functional flow with the support of experienced architects and developers.

System goals, goal calculations including algebraic equations, and constraints are defined and added to the UPDM SoS class. We put constraints on the timing values to ensure that the solution is valid for all relevant scenarios of the SoS.

After defining the main part of the SoS, the constituent-system candidates are added to the excel sheet catalogue. For each of the constituent-system classes we add two constituent systems with different cost and timing properties. The model with the data is exported to CPLEX and the optimisation results are back-annotated to the UPDM model (see Figure 7-18).

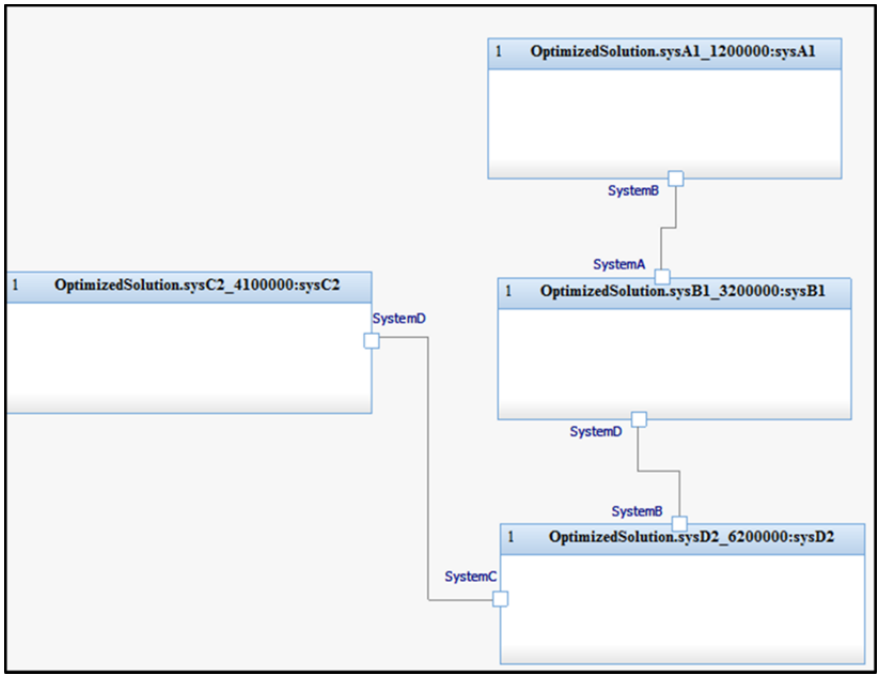

Figure 7-18 Back-annotated model for timing example

7.1.2.3 Results

In parallel to the UPDM models, the timing models were established using Matlab. We calculated all possible combinations of the four constituent systems with two options for each.

As we see from the Matlab results (Figure 7-19), the blue points represent all possible solutions, while the points marked in red are the valid solutions for the given constraints. In the model the optimisation goals were set to minimize the cost as a first priority goal, and to minimize the worst-case execution time as a second priority goal. The red circle marks the solution chosen by CPLEX which is the minimum time for the minimum possible system cost.

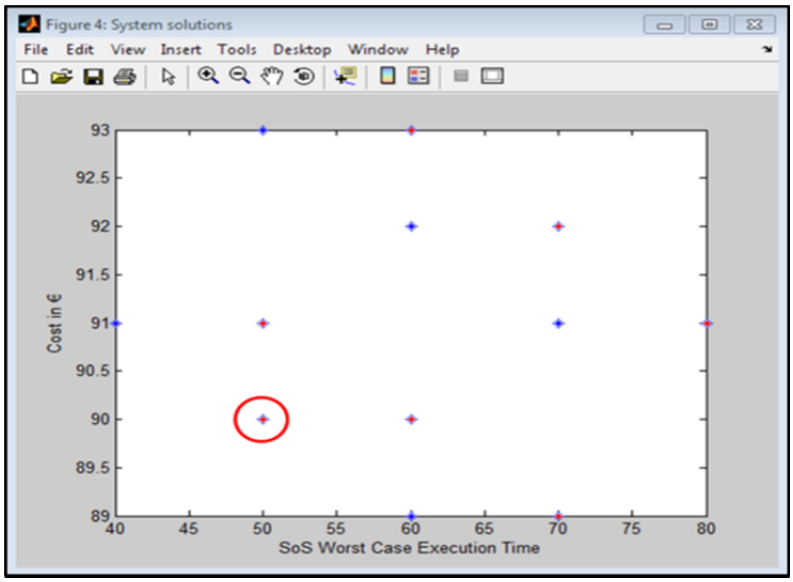

Figure 7-19 Matlab results for timing example

The resulting architecture represents an optimal solution where we assume that no inference occurs when more than one function is handled by the same constituent system. In this example we introduced timing analysis in the SoS architecture optimisation process. It is a model-based methodology were UPDM models are extended with real-time requirements. The process steps were successfully implemented and the results were verified using Matlab.

7.1.3 Reliability Example

7.1.3.1 Problem Definition

This example illustrates the integration of reliability analysis into the SoS architecture modelling and optimisation process. In our case study we consider a parallel-series SoS with redundancy. The purpose of the process is to optimize the selection of the constituent systems and to create the SoS architecture that fulfils the system requirements with minimum cost and maximum reliability. In Figure 7-20 the system pattern is illustrated, where A, B, C, D are constituent-system classes connected in series. The problem revolves around which instance of the constituent systems A, B, C and D to choose from in the constituent-system catalogue, and to determine the redundancy degree for the constituent systems C and D that achieves the required SoS reliability with the minimum SoS cost.

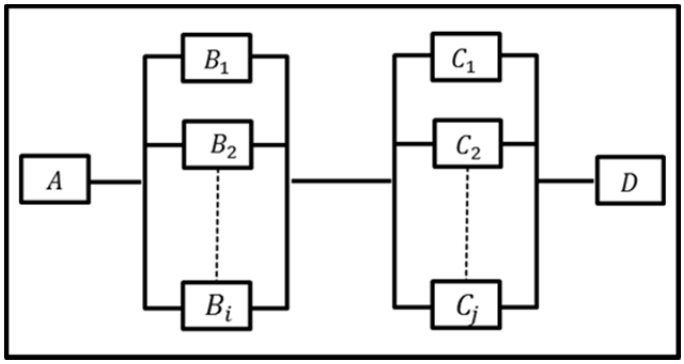

Figure 7-20 Architecture model of the reliability example

The optimisation problem is solved by formulating a multi-objective optimisation problem:

$$\max R_s = \sum_{i=1}^{n} \ln(1-(1-R_i)^{x_i})$$
$$\min g \quad = \sum_{i=1}^{n} c_i x_i \qquad (29)$$
$$s.t.$$
$$g < g_0 b \neq 0$$

Where g is the SoS cost function, c_i is the cost of constituent system i, and g_0 is the cost constraint.

7.1.3.2 Modelling and Calculations

We start with the SoS operational analysis. In Figure 7-21 the SoS operational activities are constructed after identifying the SoS operational scenarios at a high level of abstraction. At this level we identify the following requirements:

1- the reliability of the SoS that implements the operational activities must be more than z where z is an integer.

2- the SoS operations must be implemented with the highest possible levels of reliability of the SoS.

3- the cost of the SoS implementation that is used to perform the operational activities must not be more than y, where y is an integer.

4- the fixed cost of the SoS implementation that is used to perform the operational activities must be minimized.

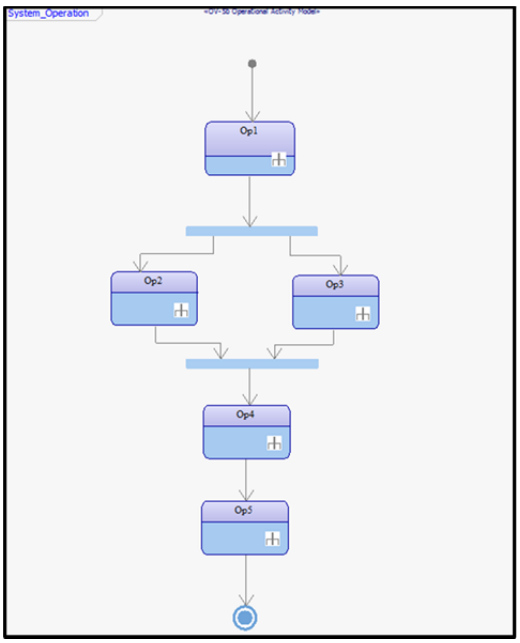

Figure 7-21 Operational activities flow diagram of the reliability example

The next step is to model the mapping of the operational activities to the functions. The mapping is shown in Figure 7-23.

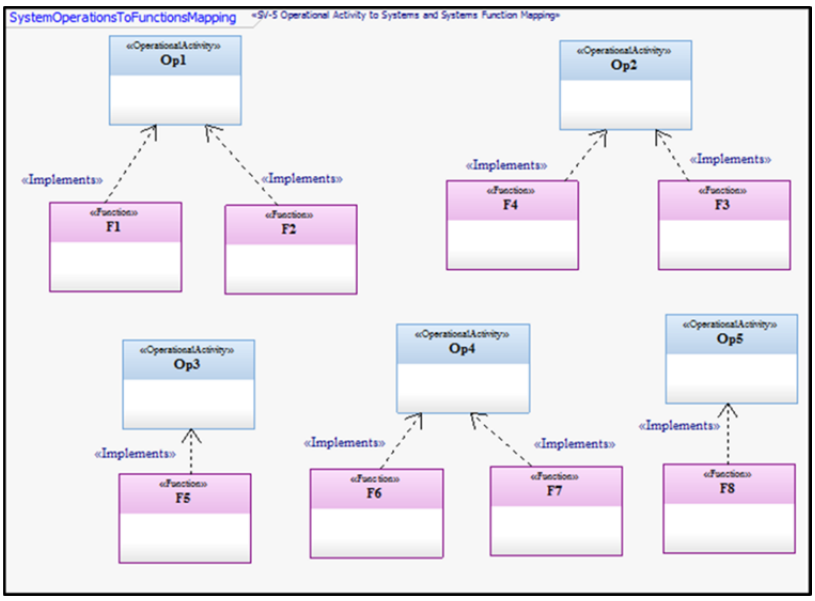

Figure 7-22 Operational activity to functional mapping of the reliability example

The purpose of the mapping process is to lower the level of abstraction and to shift from the customer to the developer domain where constituent systems are identified. Following this process the functional flow diagram is constructed in Figure 7-23. Each of these functions is mapped to the constituent system in a separate system view.

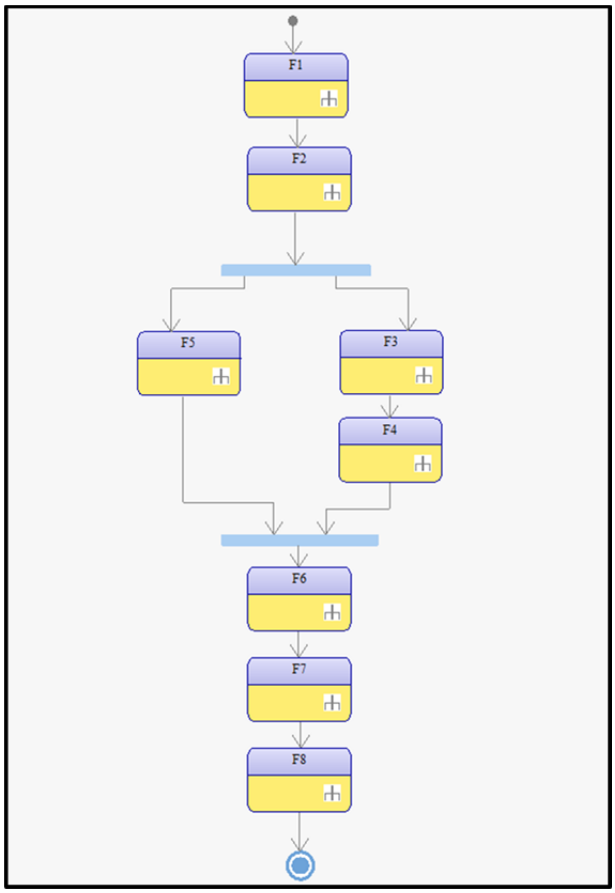

Figure 7-23 Functional flow diagram of the reliability example

Based on the functional analysis and the mapping between constituent systems and the implemented functions, we create or chose the architecture pattern that describes the connections between the constituent-system classes, and using the previously mentioned views we build the system reliability block diagram in the UPDM model as shown in Figure 7-24. Based on the connections and dependencies that are shown in the RBD of the SoS and based on equation (29) the reliability calculations are derived as follows:

$$\ln(R_s) = \ln(R_A) + \ln(1-(1-R_B)^{x_B}) + \ln(1-(1-R_C)^{x_C}) + \ln(R_D) \quad (30)$$

R_A, R_B, R_C, R_D denote the reliability of the constituent systems A, B, C and D. x_B,x_c express the redundancy degree for the constituent system B and C respectively.

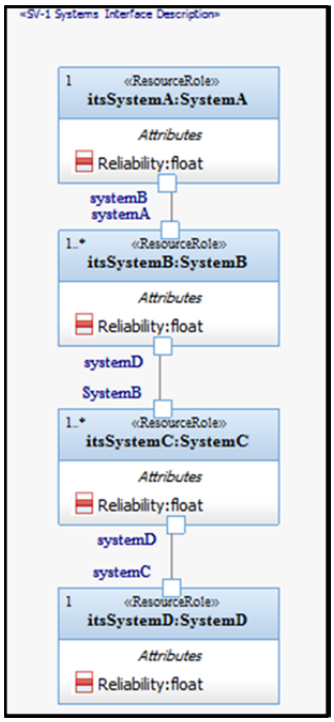

Figure 7-24 Reliability block diagram of the reliability example

We provide the constituent-system catalogue in an Excel sheet, which is generated from the model using the concise plug-in embedded in the IBM Rational Rhapsody modelling tool based on the model classes and the stereotypes that are added to these classes. For each of the constituent-system classes with different instances and different supported redundancy degrees we calculate the reliability using Excel formulas. The resulting values are inserted into the catalogue table. For example, consider constituent-system class C with two options of instances. The first type is with redundancy degrees up to 6 and the second type is with redundancy degrees up to 9. We use the equation for the reliability calculations in the Excel sheet.

Table 12 Redundancy values in reliability optimisation

int	float	RhpString	int	RhpString	float
id	cost	name	systemId	typename	Reliability
1900000	9	SC11	1	SC11	-0,010050336
1900001	18	SC12	2	SC12	-0,000100005
1900002	27	SC13	3	SC13	-1E-06
1900003	35	SC14	4	SC14	-1E-08
1900004	40	SC15	5	SC15	-1E-10
1900005	43	SC16	6	SC16	-9,99978E-13
1900006	7	SC21	7	SC21	-0,020202707
1900007	14	SC22	8	SC22	-0,00040008
1900008	20	SC23	9	SC23	-8,00003E-06
1900009	25	SC24	10	SC24	-1,6E-07
1900010	30	SC25	11	SC25	-3,2E-09
1900011	33	SC26	12	SC26	-6,4E-11
1900012	34	SC27	13	SC27	-1,27998E-12
1900013	36	SC28	14	SC28	-2,56462E-14
1900014	37	SC29	15	SC29	-5,55112E-16

The purpose of providing the redundancy values and calculations in the constituent-system catalogue is to keep the linearity of the optimisation problem. The redundancy calculations are not linear, therefore we calculate these

values in the generated data base and provide them as parameters in the optimisation engine to calculate the optimum solution.

System optimisation goals such as maximizing system reliability and minimizing system cost are defined and added to the UPDM model. For our example we added a cost constraint to the system as an upper limit to the system cost. We used the IBM Rational Rhapsody tool for building the UPDM model, MS Excel to provide the constituent-system catalogue and the CPLEX optimizer to solve the optimisation problem. A concise plug-in is used to generate the Excel sheet from the UPDM model and to create the optimisation problem for the CPLEX optimizer. After importing the created files in CPLEX, the user can run the optimisation problem and the results are exported to text files that can be imported back in the UPDM model using another concise plug-in. This process back-annotates the results as a system architecture model with the specified constituent systems and their interfaces, connections, relationships, and parameters in the 'systems interfaces description view' of UPDM.

7.1.3.3 Results

There are two dimensions of the optimisation problem: the SoS reliability and the SoS cost. To verify our results we implemented the optimisation problem also using Matlab with the same parameters. Due to the small size of the example, we could list all possible solutions. As we see in Figure 7-25 and Figure 7-26 the blue points indicate all possible solutions while the red ones represent only the valid solutions for the given constraints. The green arrow marks the solution chosen by CPLEX which exhibits the minimum cost for the maximum system reliability.

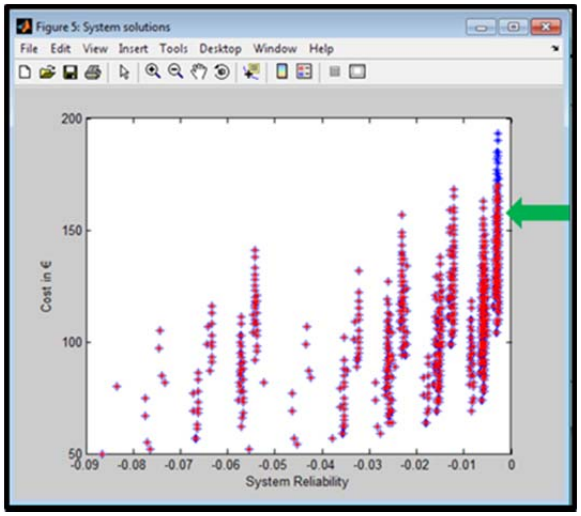

Figure 7-25 Matlab results for reliability with logarithmic values

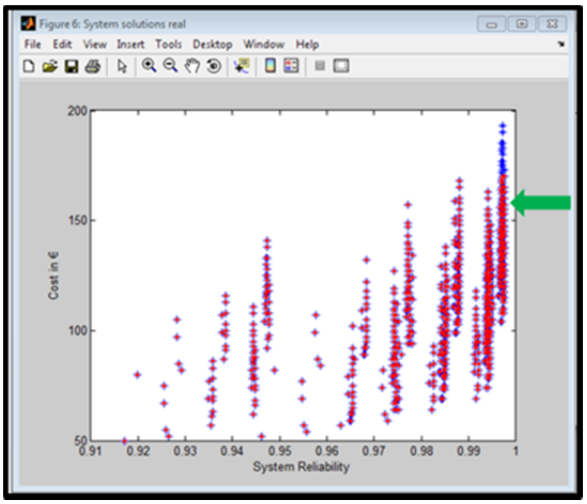

Figure 7-26 Matlab results for reliability with linear scaling

Figure 7-25 presents the results in logarithmic scaling and Figure 7-26 presents them in linear scaling. The blue points are excluded from the solution because they violate the problem constraints. The arrow indicates the optimal value, which achieves the lowest cost for the highest reliability.

8 Discussion and Conclusion

Over recent decades there has been a growing interest in the modelling and analysis of SoS in industry. Due to the inconsistent implementation of SoS design there has been a huge loss of cost and time. This has led to industrial and scientific interest in the investigation of SoS architectures concerning their functional and non-functional properties at early design phases. Present-day architecture frameworks introduce SoS models with different viewpoints, however the models are only used at a high level of abstraction and are not suitable for analysis and architecture optimization.

In this thesis we presented a SoS architecture development methodology with support for pattern-based architecture generation, while optimizing functional and non-functional properties (e.g., reliability, real time). The proposed model-based methodology describes the SoS architecture development process, starting from the SoS operational analysis and ending with the simulation of the SoS behaviour. Our methodology describes the work flow of the SoS architecture development process and defines the transition actions between the development phases. The modelling methodology uses DoDAF as a modelling framework and UPDM as a modelling language. We define a logical flow between the used architecture views and extend the UPDM profile with stereotypes that enable developers to apply architecture optimisation, the use of architecture patterns, and the simulation of the SoS behaviour.

The methodology enhances the model re-usability by introducing architecture patterns. These patterns are used for build architecture instantiations for similar SoS without the need to re-model every part of the model.

In addition, we added the timing analysis and reliability analysis to the architecture optimisation process. The methodology defines how to formulate and include the temporal requirements and reliability requirements in the architecture development and analysis.

The methodology was evaluated using three different examples, i.e., coverage analysis, timing analysis, and reliability analysis. A reference model was created in Matlab for each example to verify the resulting optimal architectures.

In future work the methodology can be extended to include more dependability attributes such as safety and availability. For timing analysis, the methodology can be extended to consider functional interference in the constraints of the UPDM model, while connecting the UPDM model with external timing analysis tools, e.g. Real-time Analyser (RTANA) to verify the fulfilment of the timing requirements.

9 References

[1] DoD Architecture Framework Working Group, "DoD Architecture Framework: Volume I: Definitions and guidelines" vol. I, technical report, April 2007.

[2] UK Government Digital Service, "MOD Architecture Framework" standard, retrieved May 27, 2015, from https://www.gov.uk/mod-architecture-framework#use-and-examples-of-modaf.

[3] NATO, "NATO Architecture Framework v.3" technical report, 2007.

[4] OMG, "Unified Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF)" technical report, 2010.

[5] M. Jamshidi, "System of Systems Engineering: Principles and Applications" Boca Raon, CRC Press, 2008.

[6] M. Maier, "Architecting Principles for Systems-of-Systems", Systems Engineering, vol. 1, no. 4, pp. 267–284, 1998.

[7] T. Lochow, I. Sanduka, R. Bullinga, A. Arnold, R. Kalawsky, and G. Cristau, "Concept Alignment Example Description" technical report, 2013.

[8] D. DeLaurentis, "Understanding Transportation as System-of-Systems Design Problem", Proceedings of the 43rd AIAA Aerospace Sciences Meeting and Exhibit, pp. 1–14, January, 2005.

[9] D. Gianni, J. Lewis-bowen, N. Lindman, and J. Fuchs, "Modelling Methodologies in Support of Complex Systems of Systems Design ad Integration : Example Applications", Proceedings of the 4th International Workshop on System & Concurrent Engineering for Space Applications, 2010.

[10] H. Ehm and W. Robert, "Galileo Europe's Share for a Global Navigation Satellite Service" technical report, Erlangen, 2011.

[11] J. King, "Overview of the Cospas-Sarsat Satellite System for Search and Rescue", Proceedings of the 6th International Mobile Satellite Conference, 1999.

[12] T. Beer, "The GMES Programme" technical report, 2010.

[13] E. Sloane, T. Way, V. Gehlot, A. Levitin, and R. Beck, "SoSE Modeling and Simulation Approaches to Evaluate Security and Performance Limitations of a Next Generation National Healthcare Information Network", Proceedings of IEEE International Conference on System of Systems Engineering, 2007.

[14] E. Sloane and T. Way, "Conceptual SOS Model and Simulation Systems for a Next Generation National Healthcare Information Network (NHIN-2): Creating a Net-Centric, Extensible, Context", Proceedings of the Systems Conference, pp. 1–6, 2007.

[15] S. Pour, E. France, and I. Works, "STANDARD ISO / IEC 15288:2008" 2nd ed, IEEE Computer Society, Aug, 2008.

[16] R. Obermaisser and H. Kopetz, "Genesys–A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems" Suedwestdeutscher Verlag fuer Hochschulschriften, October 2009.

[17] B. Zhou, A. Dvoryanchikova, A. Lobov, and J. L. M. Lastra, "Modeling System of Systems: A Generic Method Based on System Characteristics and Interface", Proceedings of the 9th IEEE International Conference on Industrial Informatics, pp. 361–368, Jul. 2011.

[18] F. Sahin, M. Jamshidi, and P. Sridhar, "A Discrete Event XML Based Simulation Framework for System of Systems Architectures", Proceedings of the IEEE International Conference on System of Systems Engineering, pp. 1–7, 2007.

[19] D. Fisher, "An Emergent Perspective on Interoperation in Systems of Systems" technical report, Software Engineering Institute, pp. 1–67, March, 2006.

[20] J. Boardman and B. Sauser, "System of Systems-The Meaning of", Proceedings of the IEEE/SMC International Conference on System of Systems Engineering, pp. 118–123, April, 2006.

[21] W. Baldwin and B. Sauser, "Modeling the Characteristics of System of Systems", Proceedings of the IEEE International Conference on System of Systems Engineering, pp. 1–6, 2009.

[22] A. Gorod and R. Gove, "System of Systems Management : A Network Management Approach", Proceedings of the IEEE International Conference on System of Systems Engineering, pp. 1–5, 2007.

[23] USA Department of Defense, "Systems Engineering Guide for Systems of Systems" technical report, 2008.

[24] USA Department of Defense, "System of Systems Systems Engineering Guide" technical report, 2006.

[25] Stevens Institute Of Technology, "Report on System of Sysems Engineering" technical report, 2006.

[26] P. A. Laplante, "Real-Time Systems Design and Analysis", 2nd ed. New York: Wiley-IEEE Press, 1997.

[27] K. M. Kavi, "Real-Time Systems: Abstractions, Languages and Design Methodologies" IEEE Computer Society Press, Los Alamitos, USA, 1992.

[28] R. Obermaisser, "Event-Triggered and Time-Triggered Control Paradigms. An Integrated Architecture" Kluwer Academic Publishers, London, 2005.

[29] H. Kopetz, “Should Responsive Systems be Event-Triggered or Time-Triggerd” technical report, November, 1993.

[30] P. Veríssimo, “On the Role of Time in Distributed Systems”, Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, vol. 84, no. 351, pp. 316–321, Tunis, Tunisia, 1997.

[31] M. Fischer, N. Lynch, and M. Paterson, “Impossibility of Distributed Consensus with One Faulty Process”, Journal of the ACM (JACM), vol. 32, no. 2, pp. 374–382, 1985.

[32] J. Liu, “Real-Time Systems” Prentic-Hall, New Jersey, USA, 2000.

[33] H. Kopetz, “Real-time Systems: Design Principles for Distributed Embedded Applications” Kluwer Academic Publishers, Boston, 2011.

[34] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstr, “The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools”, TECS ACM Transactions on Embedded Computing Systems, 2008.

[35] ARINC IA, “ARINC 800” standard, retrieved May 27, 2015, from http://www.aviation-ia.com/standards/index.html [Online].

[36] A. Avizienis, J. Laprie, and B. Randell, “Fundamental Concepts of Dependability” technical report, 2001.

[37] A. Birolini, “Reliability Engineering Theory and Practice”, Springer, Berlin, 1999.

[38] A. Avizienis and J. Laprie, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Transactions on Dependable and Secure Computing, pp. 11–33, 2004.

[39] H. Kopetz, “On the Fault Hypothesis for a Safety-Critical Real-time System”, Proceedings of Automotive Software – Connected Services in Mobile Networks, Lecture Notes in Computer Science, pp. 1–12, 2006.

[40] D. Powell, “Failure Mode Assumptions and Assumption Coverage”, Proceedings of the Twenty-Second International Symposium on Fault-Tolerant Computing, 1992.

[41] M. Hugue and R. Scalzo, “Specifying Fault Tolerance in Large Complex Computing Systems”, Proceedings of First IEEE International Conference on Engineering of Complex Computer Systems. ICECCS'95, pp. 369–372, 1995.

[42] R. Obermaisser and P. Peti, “A Fault Hypothesis for Integrated Architectures”, Proceedings of the 4th International Workshop on Intelligent Solutions in Embedded Systems (WISES'06), Vienna, Austria, 2006.

[43] J. Cook, “Multi-state Reliability Requirements for Complex Systems”, Proceedings of the Annual Reliability and Maintainability Symposium, 2008.

[44] F. Tillman, C. Hwang, and W. Kuo, “Optimization Techniques for System Reliability with Redundancy-A Review”, Proceedings of the IEEE Transactions on Reliability, vol. R-26, no. 3, pp. 148–155, Aug. 1977.

[45] W. Kuo and V. R. Prasad, “An Annotated Overview of System-Reliability Optimization”, Proceedings of the IEEE Transactions on Reliability ,vol. 49, no. 2, pp. 176–187, Jun. 2000.

[46] H. Crisp, “Systems Engineering Vision 2020” technical report, Seattle, Washington, September, 2007.

[47] S. Friendenthal, R. Steiner, and A. Moore, “A Practical Guide to SysML”, Morgan Kaufmann, Amsterdam, 2011.

[48] A. Ramos, J. Ferreira, and J. Barceló, “Model-Based Systems Engineering: An Emerging Approach for Modern”, Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, no. 1, pp. 101–111, 2012.

[49] G. Booch, J. Rumbaugh, and I. Jacobson, “The Unified Modeling Language User Guide”, Addison Wesley, Reading, Massachusetts, 1998.

[50] Object Management Group (OMG), “Model Driven Architecture (MDA)” technical report, June, 2014.

[51] C. Haskins, “Systems Engineering Handbook” v3.2, INCOSE, 2011.

[52] Defense Acquisition Guidebook, “DAG Guidebook” U.S. Dept. of Defense, Washington, 2012.

[53] T. Mazzuchi and G. Albakri, “System of Systems Engineering Facilitates Integration of Large Scale Complex Systems” INCOSE International Symposium, 2011.

[54] J. Lane and R. Valerdi, “Accelerating System of Systems Engineering Understanding and Optimization Through Lean Enterprise Principles”, Proceedings of Accelerating system of systems engineering understanding and optimization through lean enterprise principles, San Diego, CA 2010.

[55] D. O. Norman and M. L. Kuras, “Engineering Complex Systems”, In Complex Engineered Systems Understanding Complex Systems, ISBN 3540328319, Springer, 2004.

[56] A. Gorod, B. Sauser, and J. Boardman, “System-of-Systems Engineering Management: A Review of Modern History and a Path Forward”, IEEE Systems Journal, vol. 2, no. 4, pp. 484–499, Dec. 2008.

[57] ISO/IEC/IEEE 42010, “Systems and Software Engineering — Architecture Description”, IEEE, 2011.

[58] M. Maier and E. Rechtin, “The Art of Systems Architecting”, 3rd ed., CRC Press, Boca Raton, 2000.

[59] US Department of Defense, “The Department of Defence Architecture Framework (DoDAF)”, v2.0. technical report, 2009.

[60] United Kingdom Ministry of and Defence, “MODAF M3” v1.2.004, technical report, 2013.

[61] V. Haren, “TOGAF Version 9.1” Van Haren Publishing, December. 2011.

[62] M. Lankhorst, “Enterprise Architecture at Work: Modelling, Communication and Analysis. 2009” Springer Verlag: Berlin, no. 2, 2009.

[63] J. Zachman, “A Framework for Information Systems Architecture” IBM systems journal, vol. 26, no. 3, 1987.

[64] A. Vallecillo, “RM-ODP : The ISO Reference Model for Open Distributed Processing” technical report, 1999.

[65] K. Farooqui, L. Logrippo, and J. D. Meer, “The ISO Reference Model for Open Distributed Processing: An Introduction”, In Computer Networks and ISDN Systems, 1995.

[66] W. Okon, “DoDAF Strategic Direction of Moving DoDAF Towards an Unified Architecture Framework and Standard” technical report, 2013.

[67] R. James, J. Ivar, and B. Grady, “The Unified Modeling Language Reference Manual” Addison Wesley, Reading, 1999.

[68] J. Holt, “UML for Systems Engineeirng: Watching the Wheels” Second edittion. The Institution of Electrical Engineers, London, United Kingdom, 2004.

[69] Object Management Group OMG, “Object Constraint Language (OCL)” April, 2012.

[70] S. Friedenthal, A. Moore, and R. Steine, “ A Practical Guid to SysML” Morgan Kaufmann Publishers, Burlington, 2008.

[71] M. Hause, “The SysML Modelling Language”, Proceedings of the Fifteenth European Systems Engineering Conference, September, 2006.

[72] Object Management Group OMG, “Service Oriented Architecture Modeling Language (SoaML) Specification” technical report, 2012.

[73] C. Casanave, “Enterprise Service Oriented Architecture Using the OMG SoaML Standard” technical paper, Model Driven Solutions, Inc, pp. 1–21, 2009.

[74] P. Feiler, D. Gluch, and J. Hudak, “The Architecture Analysis & Design Language (AADL): An Introduction” technical report, February, 2006.

[75] Modelica Association Project, “Functional Mock-up Interface for Model Exchange and Co-Simulation” technical report, 2014.

[76] C. Alexandre, S. Ishikawa, and M. Silverstein, “A Pattern Language: Towns, Buildings, Construction” Oxford University Press, New York, 1977.

[77] R. Kalawsky, I. Sanduka, and M. Masin, “Incorporating Architecture Patterns in a SoS Optimization Framework”, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2013.

[78] E. Gamma, R. Helm, R. Johanso, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Orientated Software” Addison-Wesley, Massachutsetts, 2008.

[79] R. S. Kalawsky, D. Joannou, Y. Tian, and a. Fayoumi, “Using Architecture Patterns to Architect and Analyze Systems of Systems” Computer Science, vol. 16, pp. 283–292, Jan. 2013.

[80] H. Broodney, R. Kalawsky, and I. Sanduka, “Leveraging Domain Expertise 1in Architectural Exploration”, Proceedings of Complex Systems Design & Management, no. 287716, pp. 1–12, 2015.

[81] R. Kalawsky, D. Joannou, Y. Tian, and A. Fayoumi, “Reusable Architectural Patterns” technical report, 2013.

[82] H. Broodney and D. Dotan, “Generic Approach for Systems Design Optimization in MBSE”, Proceedings of the 22nd Annual INCOSE International Symposium, 2012.

[83] R. Martin, “A Short Introduction to OPL-Optimization Programming Language,” technical report, 2002.

[84] J. Chinneck, “Practical Optimization: A Gentle Introduction” Carleton University, Ottawa, 2006.

[85] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, “A Novel Analytical Method for Worst Case Response Time Estimation of Distributed Embedded Systems”, Proceedings of the 50th Annual Design Automation Conference (DAC), p. 1, 2013.

[86] B. Melhart and S. White, “Issues in Defining, Analyzing, Refining, and Specifying System Dependability Requirements”, Proceedings of the Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000), pp. 334–340, 2000.

[87] Q. Guo, X. Li, L. Huang, J. Gao, and X. Xiao, “Research on Reliability Optimization of Weapon System Based on Heuristic Arithmetic”, Proceedings of the International Conference on System science, Engineering design and Manufacturing informatization, pp. 24–26, Oct. 2011.

[88] S. Sharma and R. Uppal, “RF Coverage Estimation of Cellular Mobile System”, International Journal of Neural Neyworks and Advanced Applications, vol. 3, no. 6, pp. 398–403, 2011.

10 List of Publications

[1] I. Sanduka, T. Lochow, and R. Obermaisser, “Runtime Fault Prediction and Prevention for Emerging Services in System of Systems”, Proceedings of the Posters of Complex Systems Design and Management Conference, 2013.

[2] I. Sanduka and R. Obermaisser, “Model-Based Development of Systems-of-Systems with Real-Time Requirements”, Proceedings of the 12th IEEE International Conference on Industrial Informatics (INDIN 2014), 2014.

[3] I. Sanduka and R. Obermaisser, “Model-Based Development of Systems-of-Systems with Reliability Requirements”, Proceedings of the13th IEEE International Conference on Industrial Informatics (INDIN 2015), 2015.

[4] R. Kalawsky, I. Sanduka, and M. Masin, “Incorporating Architecture Patterns in a SoS Optimization Framework”, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2013.

[5] H. Broodney, R. Kalawsky, and I. Sanduka, “Leveraging Domain Expertise 1in Architectural Exploration”, Proceedings of Complex Systems Design and Management Conference, no. 287716, pp. 1–12, 2015.

[6] T. Lochow, I. Sanduka, and A. Arnold, “Model Based Architecting for evolutionary design of Systems of Systems”, Proceedings of EMEA Systems Engineering Conference 2014 Systems Engineering: Exploring New Horizons, October, 2014.

11 Declaration / Eidesstattliche Erkärung

Hiermit erkläre ich an Eides statt, dass ich die Dissertation selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe

Siegen, 02. July 2015

Place, Date

Imad Sanduka