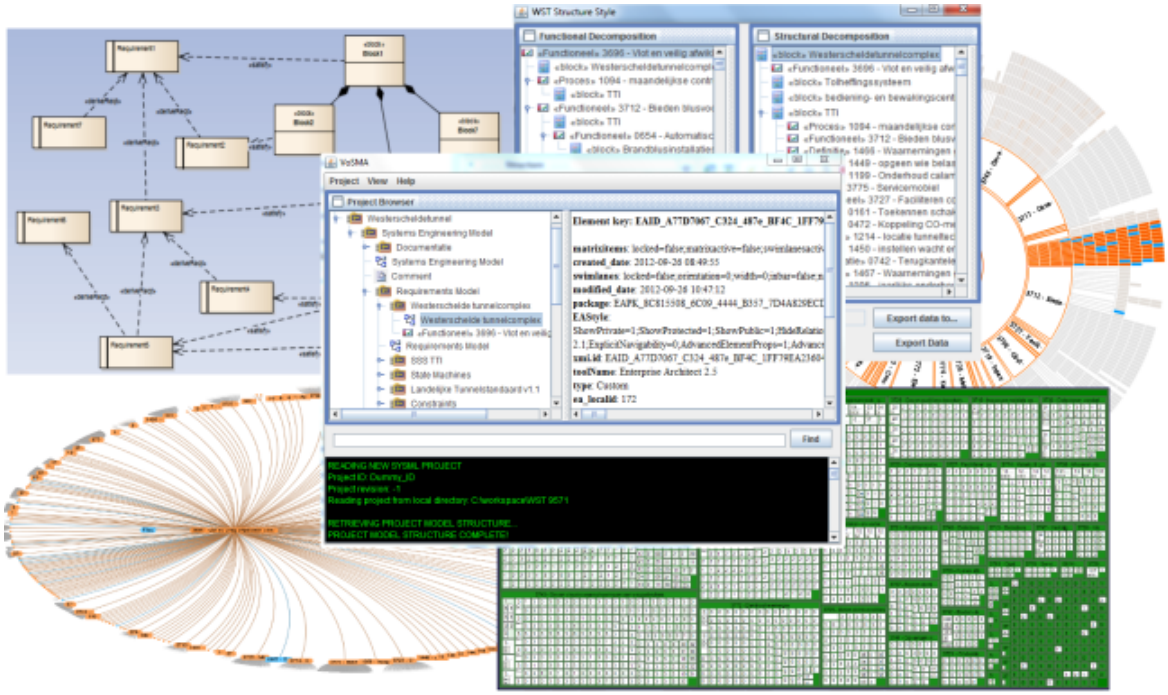
Visualization of SysML Project Meta-Model Architecture and Evolution

Version of February 12, 2013



Junior García

Visualization of SysML Project Meta-Model Architecture and Evolution

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Junior García
born in Castellón de la Plana, Spain

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Soltegro B.V.
Rivium Quadrant 159
Capelle aan den IJssel
the Netherlands
http://www.soltegro.com

Visualization of SysML Project Meta-Model Architecture and Evolution

Author: Junior García
Student id: 1538616
Email: j.garciagarcia@student.tudelft.nl

Abstract

Large and complex projects, such as infrastructure, often require the collaboration of multiple parties and disciplines, where an structured interdisciplinary methodology is necessary: Systems Engineering. This field traditionally relied on a document based approach, but is currently transitioning to a model based variant, with the Systems Modeling Language as one of its main standards. Visual modeling platforms, such as Enterprise Architect, allow to design and construct models in SysML. However, these tools lack proper measurement and visualization functionality to deal with project specific meta-model architectures. To overcome these limitations a software tool was developed using the extract-abstract-present paradigm: VoSMA. Also, multiple measurements were developed applying the Goal-Question-Metric approach, and different visualization methodologies were explored to present this information. Three SysML tunnel projects were analyzed as test case studies to assess the usefulness and correctness of the generated data, and multiple feedback sessions were conducted with experienced systems engineers. The results are very promising and indicate that the data generated may greatly benefit project development. Based on the results of the evaluation and the achieved progress, some suggestions and possible future directions were provided at the end of this study.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Prof. Dr. Ir. R. van Solingen, Faculty EEMCS, TU Delft
Company supervisor:	A. Stehouwer, Soltegro B.V.
Company supervisor:	E. Burgers, Soltegro B.V.
Committee member:	Dr. Z. Al-Ars, Faculty EEMCS, TU Delft

This thesis is lovingly dedicated to my grandmother

Mercedes Bermudo Belloso

1936-2002

Preface

This thesis report represents the culmination of my Master's degree program in Computer Science at the EEMCS faculty of the Delft University of Technology. However, it would have been impossible to complete this work without the people around me, and the least I can do is to express my gratitude in the following lines.

First I would like to mention Rini van Solingen, since he originally proposed to visit Soltegro B.V. to explore the possibilities of a Master's project. He also provided invaluable assistance during the research assignment previous to this work (together with the PhD-students Kevin and Ben), and decided to be a supervisor for this project. Next I want to thank my main university supervisor, Arie van Deursen, for his guidance and feedback. He greatly helped my research by providing multiple useful pointers and suggestions.

Of course I want to mention the people at Soltegro, starting with my general supervisor, André Stehouwer, who kept good track of my progress and ensured I had everything necessary for my work. Eric Burgers, my technical supervisor, for stating a clear goal, and providing complete and extensive answers to my questions. Hans de Man for making this internship possible and granting substantial financial compensation, which significantly helped with the expensive travel costs. And, a especial thanks to all the people who took time from their work to partake in the interview sessions, contributing to this project with their invaluable feedback. Of course I also want to mention the other people at Soltegro for the pleasant work environment!

And a special mention goes to Adry Ferwerda for taking the time to visit our offices to explore a possible collaborative research, even though this option was not pursued.

Last, but certainly not least, those who cannot be thanked enough are my family, who helped me through my whole study in all possible ways. Without their endless support all this would have never happened.

Junior García
Delft, the Netherlands
February 12, 2013

Contents

List of Figures

List of Tables

Glossary

A4DS *A4 Delft-Schiedam.* In the context of this thesis work, the MBSE solution developed by Soltegro in SysML to address this tunnel project.

CSV *Comma-Separated Values*. This file format is widely used to store tabular data (numbers and text). Lines in the text file represent table rows, while commas or semicolons in a line separate the data fields.

DOM *Document Object Model*. Cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.

EA *Enterprise Architect*. A multi-user, Windows-based, visual modeling and design platform based on the OMG UML. The tool is developed and distributed by Sparx Systems.

EAB *Enterprise Architect Branch*. A Model Branch file (*.EAB) provides a convenient reference to an exported model sub-tree. It is a small file that can be named in human readable terms, and can later be used to populate a model repository from scratch.

EAP *Enterprise Architect Project*. The EA project file (*.EAP) usually refers to the file-based Model Repository.

GQM *Goal Question Metric*. Approach to software metrics which defines a measurement model on three levels: conceptual, operational, and quantitative.

GUI *Graphical user interface*. Interface that allows users to interact with electronic devices using images rather than text commands.

INCOSE *International Council on Systems Engineering*. Non-profit membership organization dedicated to the advancement of systems engineering, and to raise the professional stature of systems engineers.

LTS *Landelijke Tunnelstandaard*. Safety standard for Dutch tunnels[1] developed by Rijkswaterstaat and multiple commercial parties. Version is 1.2 was introduced in October 2012, and it is meant as the standard for all future government tunnels in the Netherlands. The technical standard forms an important part, displaying the functional requirement processes, and outlining the design and layout of tunnels.

MBSE *Model Based Systems Engineering*. Formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.

MOF *The Meta-Object Facility*. An OMG standard for model-driven engineering.

NVD *Nijverdal.* In the context of this thesis work, the MBSE solution developed by Soltegro in SysML to address this tunnel project.

OMG *Object Management Group*. Consortium focused on modeling and model-based standards.

Project meta-model In this context, the model structure inferred from the use of a set of rules, frames, and constraints applied while modeling projects.

RE May stand for:

Reliability engineering. Engineering field that deals with the study, evaluation, and life-cycle management of reliability (the ability of a system or component to perform its required functions under stated conditions for a specified period of time).

Requirements engineering. Systems and software engineering process which covers all of the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system.

Revision control Also known as *version control* and *source control*. It involves the management of changes in documents, computer programs, and other collections of information.

SAX *Simple API for XML*. Event-based sequential access parser API developed by the XML-DEV mailing list for XML documents.

SE *Systems Engineering*. Interdisciplinary field of engineering focusing on how complex engineering projects should be designed and managed over their life cycles.

[1] `http://www.rijkswaterstaat.nl/wegen/veiligheid/tunnelveiligheid/landelijke_tunnelstandaard/`, in Dutch.

(Software) metric Measure of some property of a piece of software or its specifications.

SoS *System-of-Systems*. Collection of task-oriented or dedicated systems that pool their resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of its constituent parts.

StAX *Streaming API for XML*. Application programming interface able to read and write XML documents, originating from the Java programming language community.

SVN *Apache Subversion*. Software versioning and revision control system distributed under an open source license.

SysML *Systems Modeling Language*. General-purpose modeling language for systems engineering applications.

UML *Unified Modeling Language*. Standardized general-purpose modeling language in the field of object-oriented software engineering. Created by the OMG.

UML Profile UML customization that uses Stereotypes, Tagged Values, and Constraints.

VCS *Version control system*. Software program to aid in the management of changes in other programs and documents.

VoSMA Custom software tool in Java created for this thesis project, used to identify, check, measure and visualize project-specific meta-models in SysML.

W3C *World Wide Web Consortium*. International community that develops open standards to ensure the long-term growth of the Web.

Working Copy In this context, the set of files on a local machine retrieved from the Version Control Repository. Enterprise Architect uses the working copy files to update the model and Version Control Repositories.

XMI *XML Metadata Interchange*. An OMG standard for exchanging metadata information via XML.

XML *eXtensible Markup Language*. Markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Defined in the XML 1.0 Specification by the WC3.

XSLT *Extensible Stylesheet Language Transformations*. Language for transforming XML documents into other XML documents, XSL Formatting Objects, or other objects (such as HTML or plain text).

WST *Westerschelde Tunnel.* In the context of this thesis work, the MBSE solution developed by Soltegro in SysML to address this tunnel project.

Chapter 1

Introduction

This chapter introduces the basic notations that create the context and motivation of this thesis project. Section 1.1 explains the key role structure plays and why, as with other large and complex activities, it benefits from the Systems Engineering approach. Section 1.2 provides a general overview of Soltegro, the company involved in this project, while section 1.3 outlines the problem statement and defines the main research questions. In section 1.4 the research approach is described, followed by the a general overview of the document's structure in section 1.5.

1.1 Problem Context

According to the 2010 World Bank's Logistics Performance Index[2], the quality of Dutch infrastructure ranks second in the world. Indeed, infrastructure is crucial for the country: not only is the Netherlands the world's fifth largest exporter of goods [15], it is also one of the key entry points and distribution hub of goods in Europe: the Port of Rotterdam (Europort) is the largest port in Europe, Amsterdam Airport Schiphol is one of busiest airports in the world, there are more distribution centers than anywhere else in the continent, and almost every cargo destination can be reached by inland waterways. However, the country also has one of the highest population densities which generates increasing traffic congestions [67], and the Dutch railway network ranks among the densest in the world [75].

Tunnels, locks, bridges, traffic and rail systems play a key role in Dutch infrastructure, tied to strict regulations demanding extremely rigorous requirements with respect to safety and traffic flow. The outcome are large and complex projects, employing a combination of technical systems to guarantee the safety, reliability and availability of these installations. The involvement of a diversity of disciplines performed by multiple parties becomes indispensable, which, beyond technical solutions, demand a greater focus on processes and communication between the different concerned parties. To deal with this situation, a structured interdisciplinary approach is required, able to handle both the technical effort and the project management aspects: Systems Engineering.

[2] `http://info.worldbank.org/etools/tradesurvey/mode1b.asp` (accessed 01-08-2012)

1.2 The Company - Soltgero B.V.

Soltegro[3,4] is a consultancy firm established in 2009 by a group of experienced managers and specialists from the world of complex technical systems in various markets. Beside consultancy, the company also provides secondment, training, and realization of complete projects. Soltegro's main challenge is "the integration of different technical disciplines using a multidisciplinary approach", accomplished through the application of the Systems Engineering methodology. This has resulted in the development of its own approach with respect to design and execution of multidisciplinary projects, which is based on (and satisfies) multiple international standards (including ISO/IEC 15288, IEEE 1220, ISO 12207, CENELEC 50126, CENELEC 50128, CENELEC 50129, and CMMi). The company is also one of INCOSE's sponsors.

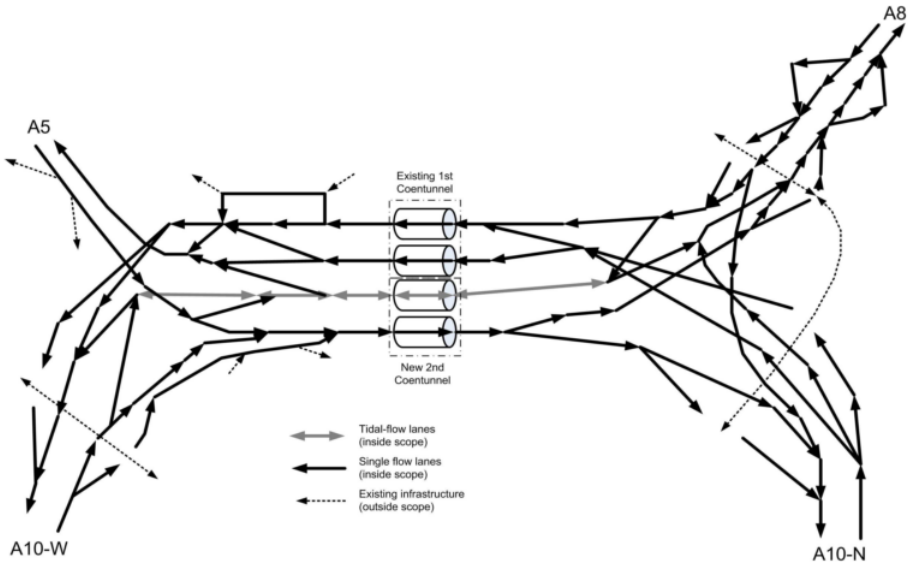


Figure 1.1: An example of a complex system: The road system in the Coentunnel (upper tubes) and Second Coentunnel (lower tubes) [55].

Soltegro also specializes in Reliability Engineering, an engineering field that deals with the study, evaluation, and life-cycle management of reliability (the ability of a system or component to perform its required functions under stated conditions for a specified period of time) [37], and Reliable Software Engineering (RSE, also known as Software Reliability Engineering), concerned with techniques for developing and maintaining software systems whose reliability can be quantitatively evaluated [39]. These approaches are crucial when working on systems which must satisfy stringent requirements in terms of safety, availability, and reliability.

[3] http://www.soltegro.com/

[4] Soltegro factsheet http://www.soltegro.com/fileadmin/user_upload/Soltegro_overview_ENG.pdf (accessed 09-10-2012)

Examples of Soltegro's involvement in infrastructure projects include the Second Coen Tunnel[5] (see figure 1.1), Nijverdal Tunnel[6], Western Scheldt Tunnel[7] (Westerscheldetunnel), and A2 Leidsche Rijn Tunnel[8]. Industrial software engineering projects include the development of new products for companies such as Priva and Oliveira. Soltegro has also been one the main knowledge partners in the first two *Tunnel safety in the Netherlands* congresses[9] (*Tunnelveiligheid in Nederland*, in its original Dutch name) organized in 2011 and 2012.

1.3 Problem Statement and Research Goals

While working on systems engineering solutions, Soltegro engineers employ the Systems Modeling Language (SysML). Models are created following a methodology based on the IEEE J-STD-016-1995[10] standard. However, this methodology may require to be adapted or modified depending on the needs and constraints of a project, leading to different project meta-model architectures (consult appendix A for an overview of meta-model evolution in tunnel projects at Soltegro). While models represent abstractions of phenomena in the real world, meta-models represent yet another abstraction highlighting properties of models. A project's system architecture meta-model may be defined as follows: the model structure inferred from the use of a set of rules, frames, and constraints applied while modeling. Soltegro engineers use the Enterprise Architect (EA) visual modeling platform to work on SysML projects. Although EA provides functionality such as element traceability (the capability to trace element relations through a model), it lacks facilities for project-specific meta-models. Thus, the goal of this research is to provide users with understanding about these meta-model structures, by identifying, analyzing and measuring the corresponding architecture, and effectively presenting this information. It is also desired to depict this data through the project's development, to visualize the project's evolution. To achieve these goals this study has been divided into two main research questions:

R.Q.1 - How can we gain insight into project-specific SysML meta-model architectures?

The objective of the first research question is to provide insight into the meta-model architectures described before. This involves developing a framework to identify and extract the project-specific structure from the SysML model data, analyze it applying the appropriate measurements, and generating the corresponding data.

[5] http://www.rijkswaterstaat.nl/wegen/plannen_en_projecten/a_wegen/a10/tweede_coentunnel_westrandweg/, in Dutch

[6] http://www.rijkswaterstaat.nl/wegen/plannen_en_projecten/n_wegen/n35/combiplannijverdal/, in Dutch

[7] http://www.westerscheldetunnel.nl/, in Dutch

[8] http://www.rijkswaterstaat.nl/wegen/plannen_en_projecten/a_wegen/a2/maarssen_tot_knooppunt_oudenrijn/, in Dutch

[9] http://www.soltegro.nl/fileadmin/user_upload/Brochure_congres_tunnelveiligheid.pdf (accessed 11-10-2012)

[10] http://standards.ieee.org/findstds/standard/J-STD-016-1995.html

R.Q.2 - How can we effectively visualize the meta-model structure, its measurement data, and picture its evolution through development?

The second research question is concerned with visualization of the previously generated data. The objective is to present this information to the users in a meaningful way, so they can gain insight into project's meta-model. The second part involves using this data to depict the meta-model's evolution through development. This aims to outline the effect of changes in the model, and may be used (for example) to perform impact analysis.

1.4 Research Approach

To answer the questions posed in the previous section, this project has been divided into three main parts:

1. **Preliminary research:** Provides context and background, including a short literature study of two relevant subjects to this research project: Systems Engineering and the Systems Modeling Language, and definitions relating the concept of "system". An overview of the current situation is also provided, including other possible available solutions.

2. **Approach implementation:** Divides the proposed solution into phases, which are further refined into multiple research sub-questions. The VoSMA software artifact, which serves as a proof-of-concept framework, is designed and implemented to satisfy the described requirements.

3. **Assessment of correctness and quality:** The last part is concerned with the verification and evaluation of the obtained results according to the established research goals.

Literature consulted for this thesis work comes primarily from academic sources written in English. Whenever multiple sources were available, the "quality" of the material was established as follows: a) journals, b) conferences, c) (non-self) published third-party academic books, d) workshops, e) technical reports, and f) others sources. Information from web pages is also included, but it is often added as footnotes providing clarification or further reading material.

1.5 Thesis Outline

The next chapter introduces some fundamental concepts, such as *system* and *system-of-systems*, which lead to a different approach to problem solving: Systems Thinking. Chapter 3 provides a general overview of the Systems Engineering field, explaining its main characteristics and how the traditional document-based approach needs to be replaced with a model-based approach, requiring a new modeling language. Chapter

4 examines the Systems Modeling Language with a general overview, but paying special attention to a couple of essential aspects to this research: how structure and requirements are modeled, and how model data is exchanged between tools. Chapter 5 describes the current situation and available methodologies, and proposes an approach based on the *extract-abstract-present* paradigm resulting in three main implementation phases: data gathering, knowledge inference and information interpretation, respectively discussed in chapters 6, 7, and 8. Chapter 9 describes how the obtained results were assessed for completeness, correctness and usefulness, while chapter 10 provides an evaluation of these results. Finally, chapter 11 summarizes the findings of this study, lists the contributions of this study, and discusses some general recommendations and possible future directions.

Chapter 2

System, System of Systems, and Systems Thinking

Some elemental concepts concerning systems must be introduced before discussing Systems Engineering and the Systems Modeling Language. Section 2.1 provides a definition of *system*, which will be used to describe the *system of systems* principle in section 2.2, in particular its main characteristics and challenges. Section 2.3 discusses how this architectural paradigm leads to a new systems-based approach: *systems thinking*, and how it differs from traditional analysis when dealing with large and complex projects.

2.1 System

The word *system*, as with many other common words in the English language, has a broad meaning. One typical definition is: *"a set of things working together as parts of a mechanism or an interconnecting network; a complex whole"* [52] (refer to appendix B for the definition of *system* in international System Engineering standards). From this description it can be derived that a system is a collection of different interacting components/parts/elements working together to achieve goals not obtainable by the individual elements alone (figure 2.1 illustrates this concept). It should be noted that the word *elements* is used to describe *parts*, which may refer to physical objects or intangible concepts (i.e. people, software, hardware, documents, policies...). The parts are required to produce system-level *results*, including system-level qualities, properties, characteristics, functions, behavior, and performance [31] [36] [41] [42]. Some defining characteristics of systems include [2]:

- *Presence*: All parts of a system must be present to carry out its purpose optimally. If components can be added or removed without affecting the functionally, then it is a collection, not a system.
- *Arrangement*: Components must be arranged in a specific way to carry out the system's purpose.
- *Purpose*: Each system has its own purpose within larger systems.

- *Stability*: Systems seek to maintain their stability through interactions, feedback and adjustments.
- *Feedback*: A system has feedback within itself and other external systems.

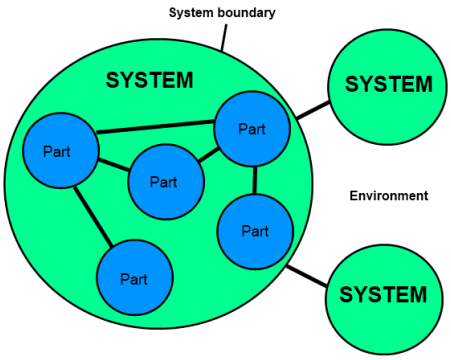


Figure 2.1: Symbolization of the concept *system*. Systems may be composed of various parts, and are delimited by a boundary. Systems may also be connected to other systems in their environment.

The concept of *system* has a fundamental role in contemporary science [1] leading to a *systems approach*, which focuses on systems as a whole rather than its parts taken separately, and where properties are addressed from a holistic[11] point of view.

2.2 System of Systems

Currently many applications are not just large scale and complex, but also characterized by distributed, decentralized networked compositions of heterogeneous and (semi) autonomous elements [30]. In the Systems Engineering community, these are described as *system of systems* (or system-of-systems, SoS), which describes a decentralized architectural paradigm. In this paradigm, systems exist within a broader context, a *super-system*, which is a collection of related systems. Also, a component in a system may be complex enough to be a system on its own, also known as *subsystem*. This leads to a hierarchical sequence (see 2.2 for an example of such a sequence). Some formal definitions of SoS include:

"A configuration of systems in which component systems can be added/removed during use; each provides useful services in its own right; and each is managed for those services. Yet, together they exhibit a synergistic, transcendent capability" [57]

"[SoS] are man-made, created and utilized to provide services in defined environments for the benefit of users and other stakeholders. These systems may be configured with one or

[11] Holistic: *"characterized by the belief that the parts of something are intimately interconnected and explicable only by reference to the whole"* [52].

more of the following: hardware, software, humans, processes (e.g., review process), procedures (e.g., operator instructions), facilities, and naturally occurring entities (e.g., water, organisms, minerals). In practice, they are thought of as products or services. The perception and definition of a particular system, its architecture and its system elements depend on an observer's interests and responsibilities. One person's system- of-interest can be viewed as a system element in another person's system-of-interest. Conversely, it can be viewed as being part of the environment of operation for another person's system-of-interest." [ISO/IEC 15288]

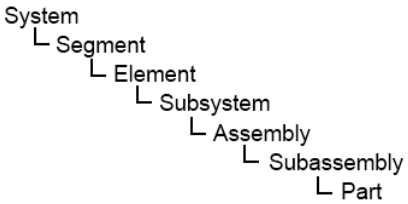


Figure 2.2: Hierarchical sequence of terms used by INCOSE's Systems Engineering Working Group.

From these definitions it can be derived that a *system of systems* is any system composed of various other systems which are themselves autonomous [20][1], where *autonomous* stands for an entity exercises independent action or decision making. However, there are five fundamental properties the whole (often) possesses to be considered a system of systems [40][20][59][8]:

- *Operational independence:* The various component systems must be able to operate in their own right and independently if dissembled from the overall system.
- *Managerial independence:* The various component systems are managed for their own purposes, and do operate independently for their own purposes rather than the purpose of the whole.
- *Geographic distribution of the systems:* *Geographic* must be interpreted as distributed (either local or wide area scale). *Distribution* refers to physical distribution and decoupling of individual system capabilities.
- *Evolutionary development:* The development of SoS is evolutionary, and adapts with functions and purposes added, removed, and modified as technologies evolve with time.
- *Emergence:* Emergent properties of SoS may not be localized to any component system, as the whole is greater than the sum of its parts.

Element	System	System of Systems
Autonomy	Autonomy is ceded by parts in order to grant autonomy to the system.	Autonomy is exercised by constituent systems in order to fulfill the purpose of the SoS .
Belonging	Parts are akin to family members; they did not choose themselves but came from parents. Belonging of parts is in their nature.	Constituent systems choose to belong on a cost/benefits basis; also in order to cause greater fulfillment of their own purposes, and because of belief in the SoS supra purpose.
Connectivity	Prescient design, along with parts, with high connectivity hidden in elements, and minimum connectivity among major subsystems.	Dynamically supplied by constituent systems with every possibility of myriad connections between constituent systems, possibly via a net-centric architecture, to enhance SoS capability.
Diversity	Managed i.e. reduced or minimized by modular hierarchy; parts' diversity encapsulated to create a known discrete module whose nature is to project simplicity into the next level of the hierarchy.	Increased diversity in SoS capability achieved by released autonomy, committed belonging, and open connectivity.
Emergence	Foreseen, both good and bad behavior, and designed in or tested out as appropriate.	Enhanced by deliberately not being foreseen, though its crucial importance is, and by creating an emergence capability climate, that will support early detection and elimination of bad behaviors.

Table 2.1: Differentiating a *system* from a *system of systems*, adapted from [8].

There are some fundamental differences between a system and a system of systems (as presented in table 2.1), but the two concepts have something fundamental in common: being "gathered together". This means that each consists of parts and their associated relationships, where the whole is greater than the sum of the parts, and thus the same in this sense. However it has been noted [20] that the "gathering together" in SoS comes from two opposing forces: a) *legacy*, given by the previous existence of systems which constitutes the SoS, and b) *mystery*, from the uncertain and unknowable environment in which SoS must operate. Given its properties, seven main challenges can be identified when developing system of systems [25]:

- *System elements operate independently:* SoS must be capable of operating on their own.
- *System elements have different life cycles:* Since SoS involve multiple components, we should be aware that these components may have different life cycles.
- *The initial requirements are likely to be ambiguous:* Requirements in the design stage are usually no more explicit than the system component requirements.
- *Complexity is a major issue:* Conflicting or missing interface standards complicate data exchange between components. Adding components also increments system complexity in a non-linear fashion.

- *Management can overshadow engineering:* The development of a SoS is complicated because every component has its own requirements, constraints, schedules, interfaces…
- *Fuzzy boundaries cause confusion:* Unless someone explicitly defines and controls the scope of a SoS, no one controls the definition of external interfaces.
- *SoS engineering is never finished:* After deploying a SoS, we must account for changes in the various components' life cycles.

2.3 Systems Thinking

Ludwig von Bertalanffy's publication of his article "General system theory" in 1956 served as the foundation of a field of study which has shown the importance of system-based approach. Systems theory (or General Systems Theory, GST) [9] is applied through systems analysis, and one of its major tools is systems thinking problem methodology approach [24][51]. Systems Thinking views systems from a broad perspective (including overall structure, patterns and cycles in systems), rather than only specific events. Adopting this methodology can lead to quickly identifying the causes of issues in complex systems, and where to work to address them.

2.3.1 The Systems Thinking Approach

Traditional analysis focuses on separating individual pieces of what is being analyzed. In fact, the word *analysis* itself means "the process of separating something into its constituent elements" [52]). In contrast, systems thinking "focuses on how the thing being studied interacts with the other constituents of the system - a set of elements that interact to produce behavior - of which it is a part" [4]. The goal of this discipline can be defined as "systems thinking is a discipline for seeing wholes. It is a framework for seeing interrelationships rather than things, for seeing patterns of change rather than static *snapshots*" [5].

Models are frequently used to analyze large and complex problems, but these are often oversimplified or incomplete, usually from applying a model to a different system than that for which it was designed [13]. Thus, instead of reacting to specific parts of a system and their components, systems thinking attempts to view problems as parts of an overall system. This is especially beneficial when dealing with dynamically complex systems involving multiple actors, or systems with a great amount of internal and/or external feedback. In short, Systems Thinking provides structured steps to walk a path toward a vision, and to gather all possible processes and interrelationships into an organized structure.

Chapter 3

Systems Engineering

The previous chapter introduced some fundamental concepts, and how they lead to a new system-based approach. However, while *systems thinking* relies upon understanding the holistic properties of complex systems, this chapter discusses an approach focused upon transforming the need for a system into a set of capabilities, requirements, functions or objects; and guides the production of these systems and services to meet this need in an effective manner: *Systems Engineering*. This can be envisaged as "to think" versus "to act" in terms of systems [5] [23]. This chapter presents a general overview of this field, providing some formal definitions, and explaining how this field originated, its management aspect, and the role of systems engineers (sections 3.1 to 3.4 respectively). One of the most important aspects for this research is discussed in section 3.5: the current transition from a document-based to a model-based approach.

3.1 Definition and Objectives

There are multiple definitions to describe Systems Engineering, but the following are provided to illustrate this concept:

"A discipline that concentrates on the design and application of the whole (system) as distinct from the parts. It requires examining a problem in its entirety, taking into account all the facets and variables and relating the social to the technical aspect. The translation of operational requirements into design, development, implementation concepts, and requirements in the life cycle of a system." [18]

"Systems engineering is a methodical, disciplined approach for the design, realization, technical management, operations, and retirement of a system." [42]

Multiple definitions can also be found in international standards (refer to appendix B for other formal definitions in SE standards):

"Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems." [25]

"An interdisciplinary approach that encompasses the entire technical effort, and evolves into and verifies an integrated and life cycle balanced set of system people, products, and process solutions that satisfy customer needs." [EIA Standard IS-632, Systems Engineering, December 1994]

"An interdisciplinary, collaborative approach that derives, evolves, and verifies a life-cycle balanced system solution which satisfies customer expectations and meets public acceptability." [IEEE P1220, Standard for Application and Management of the Systems Engineering Process, Final Draft, 26 September 1994]

"The process by which a customer's needs are satisfied through the conceptualization, design, modeling, testing, implementation, and operation of a working system." [57]

From these definitions it can be concluded that the function of Systems Engineering is to *guide the engineering of complex systems* [31], where *to guide* may be defined as "to show the way, lead, manage, or direct, usually based on the superior experience in pursuing a given course". In short, Systems Engineering can be defined as a holistic, integrative interdisciplinary engineering management process focusing on how complex engineering projects should be designed and managed over their life cycles. Thus, Systems Engineering is about looking at the *big picture*, not just by meeting the requirements, but also following the right design.

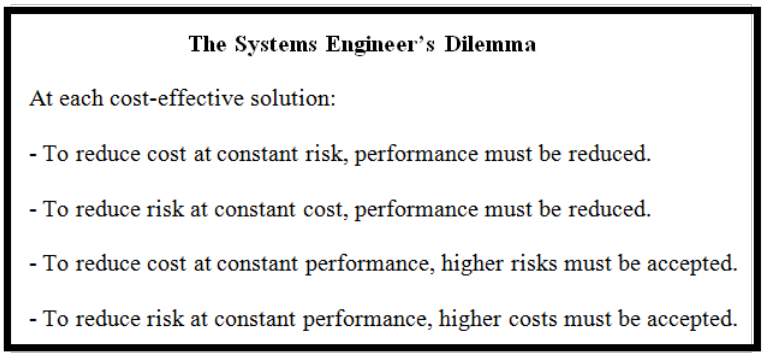


Figure 3.1: The System Engineer's dilemma. It should be noted that time in a schedule is often a critical resource, and thus schedule behaves like a kind of cost.

The objective of Systems Engineering has also been described as "*to see to it that the system is designed, built, and operated so that it accomplishes its purpose in the most cost-effective way possible, considering performance, cost, schedule, and risk*" [41]. In this context, *cost-effective* refers to a balance between *effectiveness* (quantitative measure of the degree to which the system's purpose is achieved) and *cost* (foregone value of the resources needed to design, build, and operate the system). This leads to the Systems Engineer's Dilemma [42], as seen in picture 3.1.

3.2 Origins

It is difficult to pinpoint the origins of Systems Engineering to a particular date, since the discipline has been involved in some form in the execution of complex projects following a system specification through history. However, the term itself can be traced back to the Bell Laboratories in the 1940s, and textbooks first recognized it as a separated field in the 1950s [28]. Moreover, Systems Engineering as a separated activity can be traced to the effects of World War II, which triggered a tremendous advancement in technology and the development of complex systems (e.g. jet fighters, military radar, ballistic missiles, proximity fuse, the atomic bomb...). The need for an increased level of organization and efficiency to combine multiple technical disciplines, and a new approach to program planning to meet tight schedules, resulted in the Systems Engineering field as we know it today. During the Cold War (1950s-1980s), technological advancement was still greatly influenced by military requirements, but it is the development of solid-state electronics which probably had the biggest impact. This led to the ongoing Information (or Digital) Age, with the development of the digital computer and associated software technology, resulting in increased automation and greater system complexity. These are of particular interest to Systems Engineering. In the summer of 1990 a professional society for systems engineering, the National Council on Systems Engineering (NCOSE), was founded by representatives from a number of U.S. corporations and organizations. In 1995, due to the growing involvement of systems engineers outside of the U.S., the name of the organization was changed to International Council on Systems Engineering[12] (INCOSE). INCOSE is a non-profit membership organization dedicated to the advancement of systems engineering and to raise the professional stature of systems engineers. In 2002 the international standard ISO/IE 15288 was introduced, which formally recognized systems engineering as a preferred instrument, and establishes a common framework for describing the life cycle of systems created by humans, defining a set of processes and associated terminology within that framework. In short, the origins of SE can be attributed to three basic factors [31] [12]: a) *advancing technology* which leads to new opportunities as well as new challenges, b) *competitive pressures* on the system development process leading to trade-offs, and c) *specialization* of systems leading to the emergence of interfaces.

3.3 Systems Engineering Process and Management

Systems Engineering is often divided into two main sub-disciplines [64] [36]:

- *Systems Engineering Process*, actual technical knowledge domain, including identification and quantification of system goals, creation of alternative system design concepts, performance of design trades...
- *Systems Engineering Management*, which integrates three main activities (as seen in figure 3.2): development phasing, Systems Engineering Process, and life cycle integration.

[12] `http://www.incose.org/` (accessed 01-06-2012)

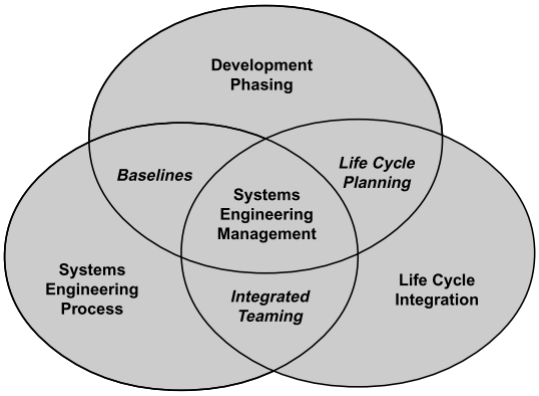


Figure 3.2: The three activities of Systems Engineering management, as depicted in [36].

Development phase: The development phase involves a design process where more detailed systems descriptions or designs are progressively produced:

- *System concept* describes what the system should do and how it would operate.
- *Functional baseline* where all the functions the system are identified and specified.
- *Allocated baseline* in which subsystems are defined and the functions are allocated to the various subsystems.
- *Product baseline* describing the detailed design.

Systems Engineering Process: The Systems Engineering Process (SEP) is "*a comprehensive, iterative and recursive problem solving process, applied sequentially top-down by integrated teams*" [36]. Its purpose is to transform a set of needs and requirements into a system product and process descriptions, achieved by generating information for decision makers, and providing input for the next level of development. This process is applied sequentially (as shown in figure 3.3) where the steps are as follows:

- *Process Inputs*: Inputs consist primarily of the customer's needs, objectives, requirements and project constraints.
- *Requirements Analysis*: Used to develop functional and performance requirements (what the system must do and how well it must perform).
- *Functional Analysis/Allocation*: Decompose higher-level functions into a description of the product or item in terms of what it does logically and in terms of the performance required.
- *Requirements Loop*: Iterative process of revisiting requirements analysis as a result of functional analysis.

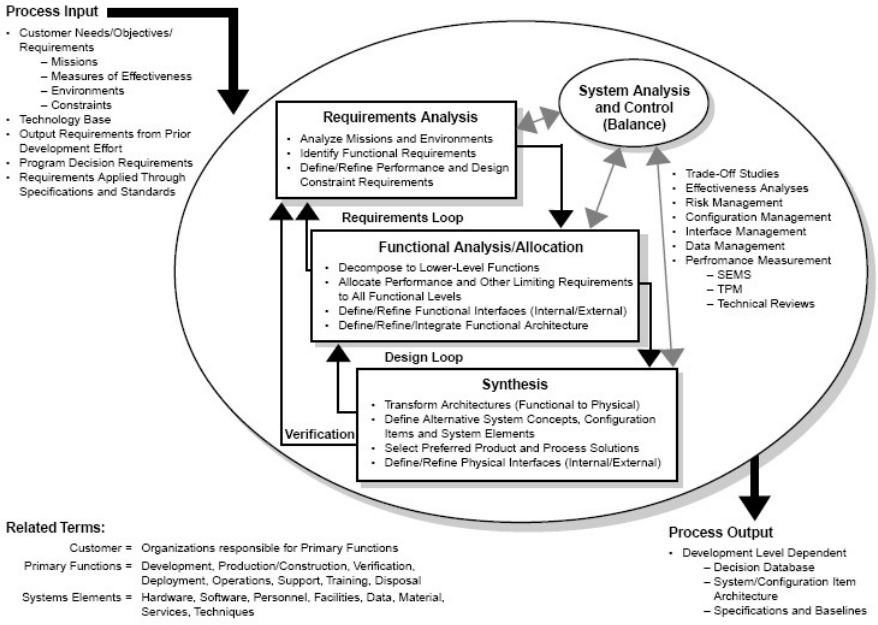


Figure 3.3: Systems Engineering Process, as originally depicted in [36].

- *Design Synthesis*: Process of defining the product or item in terms of the physical and software elements.
- *Design Loop*: Revisiting the functional architecture to verify that the physical design synthesized can perform the required functions at required levels of performance.
- *Verification*: Solution is compared to the requirements for each application of the process.
- *Systems Analysis and Control*: Used to measure progress, evaluate and select alternatives, and document data and decisions.
- *Process Output*: Includes any data that describes or controls the product configuration or the processes necessary to develop the product.

Life cycle integration: Every system has a life cycle which includes the development, production, usage, and retirement stages. Systems Engineering encompasses the entire life cycle for the system, and its goal can be defined as:

"The purpose and outcomes shall be defined for each stage of the life cycle. The life cycle processes and activities are selected, tailored as appropriate, and employed in a stage to fulfill the purpose and outcomes of that stage." [ISO/IEC 15288]

3.4 Systems Engineers

Systems Engineering has been described as both and *art* and a *science* [56], where systems engineers must display *technical leadership* (the art), focusing on a system's technical design and technical integrity throughout its life-cycle; and handle *systems management* (the science), focusing on rigorously and efficiently managing the development and operation of complex systems. Many personal behavior characteristics have been attributed to effective systems engineers [31] [56], both innate, or learned and honed (refer to figure 3.4 for an example).

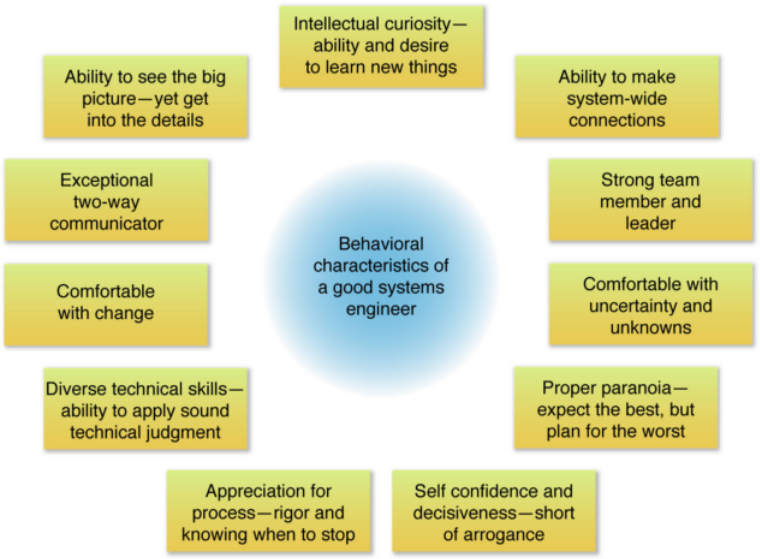


Figure 3.4: Characteristics of good systems engineers as seen in [56], with decreasing priority from top to bottom.

Systems engineers (sometimes referred as technical manager or chief engineer), need to find a safe and balanced design while dealing with multiple opposing interests and constraints. Systems engineers must possess the skill to focus efforts to optimize the overall design, instead of favoring (sub)systems at the expense of another. Responsibilities of a systems engineer in a project include: ensuring that the system fulfills the technical needs and requirements defined, monitoring, coordination and oversight of technical teams, and evaluation of the technical aspects of the project to ensure the proper function of all (sub)systems from concept to product. Among many other tasks, the lead system engineer of a project is usually charged with a leading role in the development of the systems architecture, the definition and allocation of requirements, evaluation of design trade-offs, and oversight in the validation and verification of activities. The systems engineer also must produce many of the project documents, including requirements-, specification-, verification-, validation-, and certification-documents. These include the Systems Engineering Management Plan (SEMP), which provides the framework and

guidance for all engineering activities within the overall project. The lead systems engineer of a project must ensure that the system fulfills the technical needs and requirements defined, while directing, communicating, monitoring and coordination tasks between technical teams. Systems engineers can fulfill multiple roles in a project, and up to twelve different roles have been identified and analyzed [60].

3.5 Document-Based versus Model-Based Approach

Traditionally, large and complex projects have relied on a *document-based systems engineering approach*, characterized by the generation of textual specifications and design documents, which are then exchanged between the project's stakeholders (refer to appendix C for a list of stakeholders in SE). Some fundamental concepts of this approach include [21]:

1. *Specification tree*: Depicts in a hierarchical manner the specifications for a particular system, its subsystems, and its hardware and software components.
2. *Systems engineering management plan*: Documents the systems engineering process employed on the project, and how the engineering disciplines work together to develop the documentation needed to satisfy the requirements in the specification tree.
3. *Operation document*: Defines how the system is used to support the required mission or objective.
4. *Document-based requirements traceability*: Maintained by identifying the part of the system or subsystem that satisfies the requirement, and/or the verification procedures used to verify the requirement, and then reflecting this in the requirements database.

This approach has some fundamental limitations. Completeness, consistency, and relationships between requirements, design, engineering analysis, and test information is spread across multiple documents. This results in difficulty to access information and to understand a particular aspect of the system, which leads to problems with requirements traceability and impact assessments when changes occur, especially for an evolving or variant system design.

Model-based systems engineering (MBSE) attempts to overcome these limitations [50] by emphasizing a system architecture model as the primary work artifact, combining traditional SE best practices with rigorous visual modeling techniques. MBSE is starting to be more prevalent in SE, becoming part of a long-term trend toward model-centric approaches, although it has been the standard practice in disciplines such as mechanical engineering and electrical engineering for many years. A formal definition provided by INCOSE is as follows:

Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities

beginning in the conceptual design phase and continuing throughout development and later life cycle phases. [29]

Some of the fundamental characteristics of MBSE include:

1. *System model*: Includes system specification, design, analysis, and verification information, and consists of elements that represent requirements, design elements, test cases, design rationale, and their interrelationships. Primarily used to design a system that satisfies system requirements and to allocate these requirements to the system's components.

2. *Model repository*: Stores model elements and captures specification, design, analysis, and verification information.

3. *Model-based requirements traceability*: The system model maintains rigorous traceability between requirements, design, analysis, and testing.

Take figure 3.5 as an example, which shows the traditional *V-Model* approach. In this methodology each stage of refinement results in a set of documents, which serve as input for the next level of system definition. Models used at each level of definition are independent, often relying on different techniques and tools. Thus, document validity is essentially established by isolated review of the document's content, and traceability is limited to the requirements as stated by the previous product.

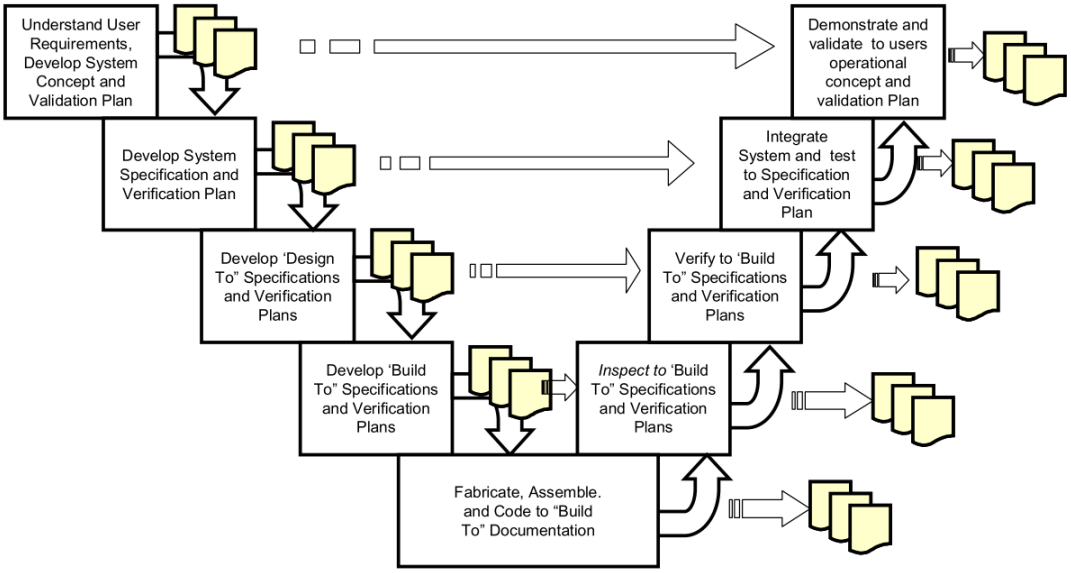


Figure 3.5: The Systems Engineering process using the V-Model, taken from [38].

Now see figure 3.6 for an example of a model-based and model-centric approach, which involves the same tasks as in the previous methodology. However, in this case these

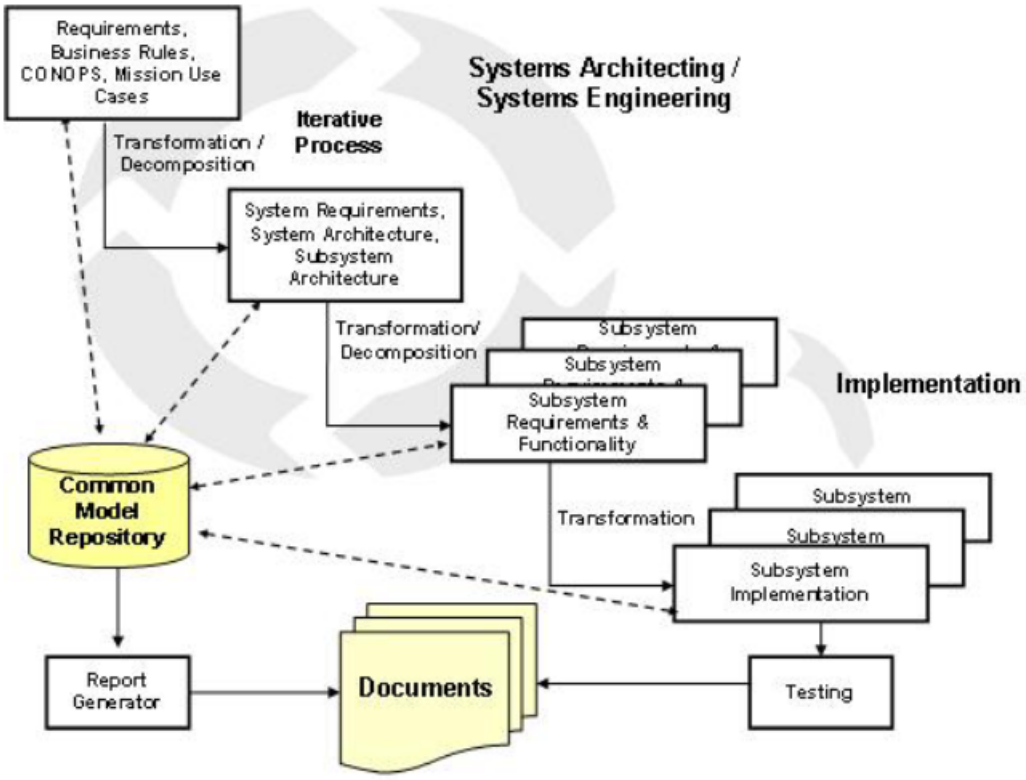


Figure 3.6: The Systems Engineering Process using a model-centric approach, taken from [38].

tasks contribute to a central model repository (a relational database which captures system elements and relationships). Traceability is provided through these relationships, and documents are generated from the model data. MBSE provides multiple methodologies [17] to address many of the limitations of a document-based approach, which define a more rigorous means for capturing and integrating system requirements, design, analysis, and verification information, and facilitating the maintenance, assessment, and communication of this information across the system's life cycle. It is the INCOSE's intention [29] to facilitate the transition to MBSE, shifting the emphasis from controlling the documentation about the system, to controlling the model of the system. However, lack of tool interoperability and absence of convergent MBSE standards has been a significant inhibitor to widespread deployment of MBSE. Still, systems modeling standards are commencing to emerge which should have a significant impact on the application and practice of MBSE, including OMG's Systems Modeling Language, discussed in the next chapter.

Chapter 4

The Systems Modeling Language

The previous chapter described how the Systems Engineering field is currently transitioning from a traditional document-based approach to a model-based approach. There are some emerging systems modeling standards, with the Systems Modeling Language as a prime example. This chapter presents a general introduction to SysML, describing its origins, providing a general overview, and describing the diagram types used (sections 4.1 to 4.3 respectively). The most relevant aspects to this research are presented in section 4.4 (how structure is modeled), and section 4.5 (how requirements are modeled). Finally, section 4.6 analyzes the current situation when exchanging data between modeling tools.

4.1 Background

As described before, Systems Engineering processes tend to be document-centric, employing a variety of techniques which are often inconsistent and imprecise. During the 1990s software engineers sought a general-purpose modeling language to specify software-intensive systems, which derived in the Unified Modeling Language (UML) (see figure 4.1 for a visualization of the evolution of modeling languages). UML is now the standard modeling language in the software community. UML is also capable to address the systems engineering needs through its wide range of notations, and it is adaptable though extensions known as *UML profiles*. Some examples of SE projects using adapted UML profiles include *MARTE* [48] and *System on a Chip* [74]. Still, systems engineers seek a general purpose modeling language to specify complex systems-of-systems that include non-software components. UML cannot satisfy this need without modifications because of its software bias, while lacking the semantics to model requirements and parametric constraints (which are crucial to support requirements engineering and performance analysis, two essential Systems Engineering activities).

The Systems Modeling Language initiative originated in January 2001 from the decision of the INCOSE's Model Driven Systems Design workgroup to customize the UML for Systems Engineering applications. This decision led to a collaborative effort between the OMG[13] (Object Management Group), which maintains the UML

[13] http://www.omg.org/

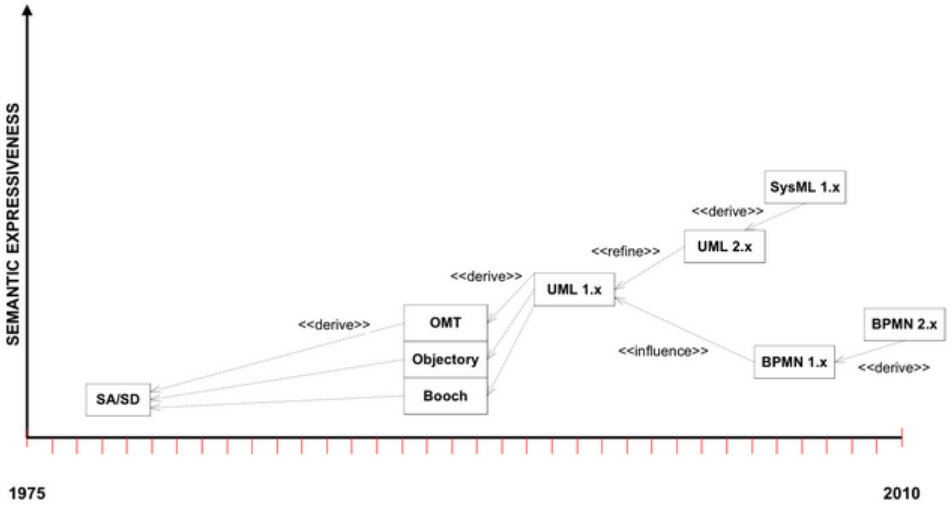


Figure 4.1: SysML and visual modeling language evolution, from [68].

specification, and INCOSE, resulting in the establishment of the OMG Systems Engineering Domain Special Interest Group[14] (SE DSIG) in July 2001. The SE DSIG, aided by INCOSE and the ISO AP 233 workgroup, developed the requirements for the modeling language. In March 2003, these requirements were issued by the OMG as part of the UML for Systems Engineering Request for Proposal (SE RFP). The goals of this language were defined as follows:

"A standard modelling language for Systems Engineering to analyze, specify, design, and verify complex systems, is intended to enhance systems quality, improve the ability to exchange systems engineering information amongst tools, and help bridge the semantic gap between systems, software, and other engineering disciplines". [19]

In 2003, in response to these developments, an informal association of industry leaders and tool vendors organized in SysML Partners [69] to initiate an open source SysML specification project. Their first open source SysML specification drafts were distributed in 2004, and the SysML 1.0a was submitted to the OMG for technology adoption in November 2005. This led to a series of competing proposals, which ended with the SysML Merge Team, adopted in July 2006 as OMG SysML. It is worth noting that OMG SysML™ is trademarked and maintained by the OMG, but since it is derived from open source SysML, an open source license is included for distribution and use.

[14] `http://syseng.omg.org/`

4.2 Language Overview

SysML is defined as an extension of the OMG UML 2.0 Superstructure Specification [44] which provides a standard modeling language to support the specification, analysis, design, verification and validation of a broad range of complex systems which are not necessarily software related. SysML is based on the minimal subset of UML that satisfies the needs of systems engineers, adapting UML only when it is required. Figure 4.2 shows the relationship between the UML and SysML languages. The subset of modeling constructs that SysML reuses from UML is called "UML4SysML". The new modeling constructs defined for SysML, which have no counterparts in UML or replace UML constructs, are called "SysML extensions to UML". Other constructs that are not required to implement SysML are defined as "UML not required by SysML". It should be noted that UML does not address how to trace the requirements of a system from informal specifications down to the individual design elements and test cases, a crucial activity in Systems Engineering. Instead SysML provides support for representing requirements and relating them to the model of a system, the actual design, and the test procedures.

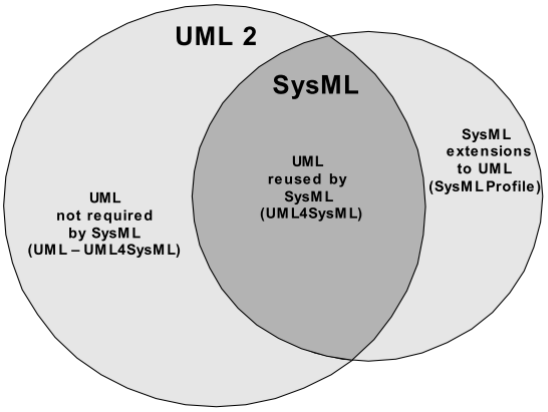


Figure 4.2: Overview of SysML/UML interrelationship, as originally appeared in [49].

4.2.1 Design Principles

SysML was created according to the following fundamental design principles [47] [49]:

1. *Requirements-driven*, intended to satisfy the requirements of the UML for SE RFP.
2. *UML reuse*, by reusing UML wherever practical to satisfy the requirements of the RFP, and when modifications are required, they are done in a manner that strives to minimize changes to the underlying language.
3. *UML extensions*, by extending UML as needed to satisfy the requirements of the RFP.
4. *Partitioning*, where the package is the basic unit of partitioning in this specification. The packages partition the model elements into logical groupings that minimize circular dependencies among them.
5. *Layering*, where packages are specified as an extension layer to the UML metamodel.
6. *Interoperability*, by inheriting the XMI interchange capability from UML.

Cris Kobryn, chair of the SysML Partners, observed [68] that 80%+ of the time the SysML Partners discussed SysML language features in four diagrams: Requirement,

Activity, Block, and Parametric diagrams, used to specify, respectively, system requirements, behavior, structure, and parametric relationships. He coined the term *Four Pillars* of SysML, as depicted in figure 4.3, to refer to these four essential diagrams [14]. From these four concepts, two are essential for our research: structure and requirements, which will be further discussed in the following sections.

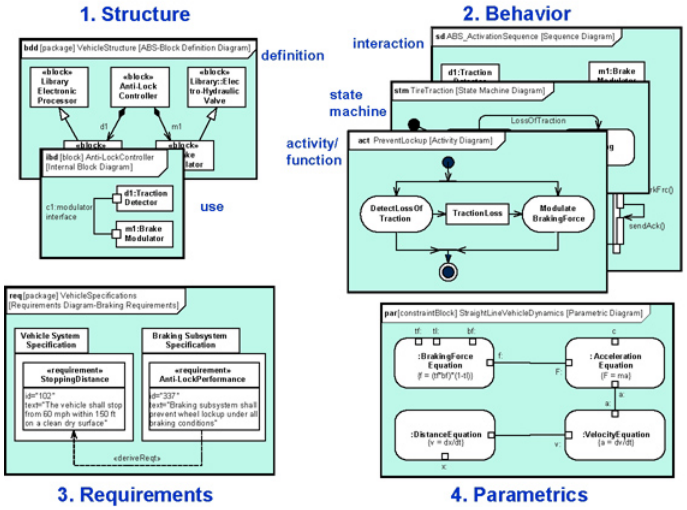


Figure 4.3: The *Four Pillars* of SysML, from [72].

4.3 Diagram Overview

Figure 4.4 shows all the diagram types supported by SysML (refer to appendix E for a comparison between SysML diagrams and their UML counterparts). Every diagram graphically represents a particular aspect of the system model. The diagram types in SysML are as follows:

- *Activity diagram (act):* Represents behavior in terms of the ordering of actions based on the availability of inputs, outputs, and control, and how the actions transform the inputs to outputs.
- *Sequence diagram (sd):* Represents behavior in terms of a sequence of messages exchanged between parts.
- *State machine diagram (stm):* Represents behavior of an entity in terms of its transitions between states triggered by events.
- *Use case diagram (uc):* Represents functionality in terms of how a system or other entity is used by external entities to accomplish a set of goals.
- *Block definition diagram (bdd):* Represents structural elements called blocks, and their composition and classification.

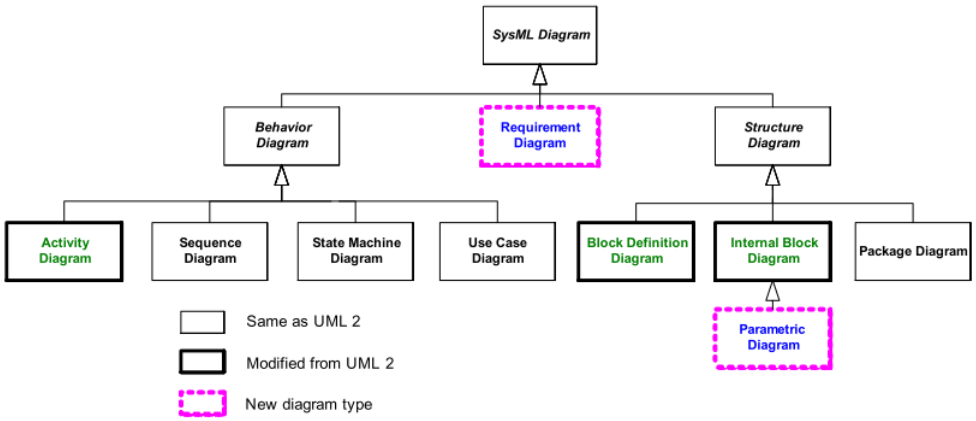


Figure 4.4: SysML diagram taxonomy, as originally appeared in [49].

- *Internal block diagram (ibd):* Represents interconnection and interfaces between the parts of a block.

- *Parametric diagram (par):* Represents constraints on property values used to support engineering analysis.

- *Package diagram (pkg):* Represents the model structure.

- *Requirement diagram (req):* Represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability.

Table 4.1 shows the valid permutations of model elements in the different diagram types.

SysML Diagram Type	**Model Element Allowed**
Activity diagram	activity control operator
Internal block diagram	block
Package diagram	package, model, model library, profile, view
Parametric diagram	block, constraint block
Block definition diagram	block, constraint block, package, model, model library
State machine	state machine
Use case diagram	package, model, model library
Requirement diagram	package, model, model library, requirement
Sequence diagram	interaction

Table 4.1: Allowed permutations of model elements in SysML diagrams.

4.4 Structure

In SysML packages are used to organize the model, system structure is represented in block definition diagrams, and internal block diagrams describe the internals of a block such as parts, ports, and connectors. SysML also involves modeling *blocks* (instead of modeling classes as in UML) to suit the vocabulary of systems engineers. Blocks extend the UML Structured Class, and are a general purpose hierarchical structuring mechanism that abstracts away much of the software-specific detail implicit in UML structured classes. A block may represent software, hardware, data, process, personnel, facilities and any other system element [54]. Blocks are shown in diagrams as UML classes stereotyped «block».

4.5 Requirements

Requirements for a system are a collection of needs expressed by stakeholders regarding some constraints under which the system must operate [65]. These requirements may come from different sources (e.g. the party requiring the system, regulations and norms, consumer preferences...), and are realized as functionality and constraints which must be satisfied by the delivered application or system. Requirements form a significant part of the contractual agreement between acquirer and supplier, and must be in a form that is understandable to both non-technical customers and technical developers. Systems engineers must ensure that these requirements are: expressed in clear and unambiguous terms, consistent, feasible, validated according to the needs of the stakeholders, and verified to ensure that they are satisfied by the system design.

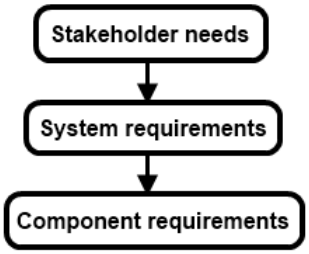


Figure 4.5: Requirements flow, where stakeholder needs flow down as system requirements, which in turn flow down as component requirements in SysML.

It is a common practice to group similar requirements into a *specification*. Stakeholder requirements are gathered in the *user requirement* specification, which is usually written using natural language. Systems requirements are derived from this specification, and include a detailed description of how the system should work using (semi) formal methods and languages. The process by which requirements for systems and software products are gathered, analyzed, documented and managed throughout the development life cycle is called Requirements Engineering (RE) [16].

One of the principal extensions of SysML is the support for requirements [26], where each requirement specifies a capability or condition that must (or should) be satisfied, a function that a system must perform, or a performance condition a system must achieve. Component requirements are modeled in SysML to satisfy these system requirements using the stereotype «requirement», which specifies the textual *shall* statement (picture 4.5 illustrates the process of requirement modeling).

4.6 Model Interchange Between Tools

Data can be exchanged between SysML modeling tools trough manual, file-, interaction-, and repository-based mechanisms. However, to reduce the cost and improve the quality of the data exchange, a standardized approach is often preferred [29].

4.6.1 ISO AP 233

OMG's SysML specification is intended to be compatible with the ISO AP233 Systems Engineering Data Exchange standard (formally ISO 10303-233). This standard is still evolving and is intended to describe, represent, and exchange industrial data in a computer interpretable format. When implemented, the standard will allow to support the whole system development life cycle ranging from requirements definition to system verification and validation in different fields, including: Engineering Analysis, Algorithm Design, Planning Tools, Testing Tools, Software Design, Mechanical Computer Aided Design, and Electrical Computer Aided Engineering. With this standard, OMG will try to ensure that models can be exchanged using the AP233 data exchange protocol, and data generated by another tool stored in AP233 can be visualized in any other OMG SysML tool.

4.6.2 XML Metadata Interchange

The XML Metadata Interchange (XMI) [46] is an OMG standard for exchanging metadata information via eXtensible Markup Language (XML). XMI integrates three industry standards:

1. XML: eXtensible Markup Language, a W3C standard.
2. UML: Unified Modeling Language, an OMG modeling standard.
3. MOF: Meta Object Facility, an OMG language for specifying metamodels.

XMI is used for any metadata whose metamodel can be expressed in Meta-Object Facility, and is commonly applied to serialize UML models for interchange. Since SysML is an extension of the UML metamodel, SysML models can also be exchanged using an XMI schema. However it should be noted that certain limitations exist, especially when exchanging diagrammatic information.

4.6.3 Interchange Problems

We should be aware of the distinction between data interchange and diagram interchange. XMI and AP 233 are used to exchange model data, but do not include diagram layout information, such as where symbols appear in a diagram. OMG provides the *Diagram Interchange* standard [45], though it is not widely used. It is a long term goal of the OMG to address this challenge in the future through the use of MOF-based models.

Chapter 5

Current Situation and Proposed Approach

The previous chapter presented a general overview of the Systems Modeling Language, paying particular attention to how models are structured, how requirements are modeled an traced, and how data can be exchanged between tools. In this chapter, section 5.1 further analyzes the current situation at Soltegro. In sections 5.2 and 5.3 multiple alternatives are evaluated. However, it is concluded that the development of a new framework is necessary. Thus, in section 5.4 a software tool is designed to provide answer to the main research questions, and multiple research sub-questions are defined for each of the implementation phases.

5.1 Current Situation

Figure 5.1 shows an overview of the current situation. Soltegro systems engineers create and model SysML projects using the Enterprise Architect[15] (EA) visual software platform. According to the engineers themselves, this tool was chosen because, in their opinion, it is one of the best visual modeling tools available, particularly highlighting its support for SysML, scalability, and license price. Figure 5.2 presents an example of EA's interface. Important components to note are:

A. Toolbox: Allows the user to add various elements to the model.

B. Diagram viewer: Allows the user to inspect and modify diagrams.

C. Project browser: Allows the user to navigate through the project space. It displays packages, diagrams, elements and element features in a tree like structure, reflecting the arrangement of elements and packages within the model.

D. Traceability window: Shows element traceability through the model, allowing the user to explore the relationship chain of which any element is a component.

[15] `http://www.sparxsystems.com/products/ea/` (accessed 01-06-2012)

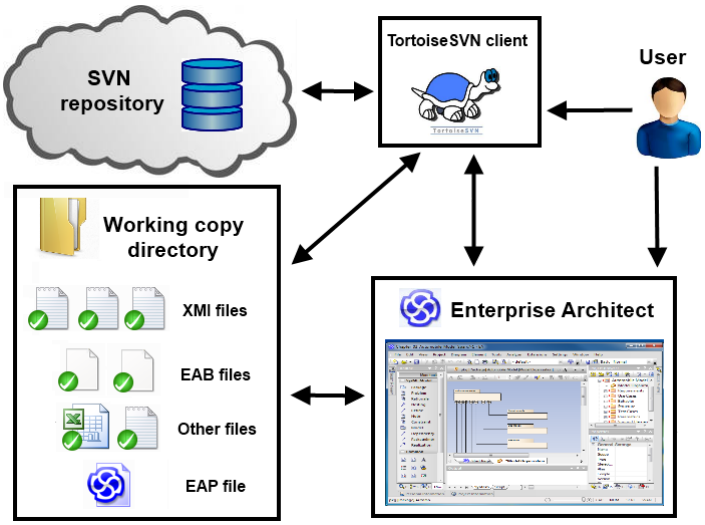


Figure 5.1: Overview of the current situation at Soltegro.

All Enterprise Architect models are stored in databases, which are kept in a single Enterprise Architect Project file (with an EAP extension). This EAP file uses a Microsoft Jet database engine[16], and may be browsed using programs such as Microsoft Access 97, 2000 or 2003, or any other reporting tool that can work with JET databases (see figure 5.3 for an example of the contents of this file).

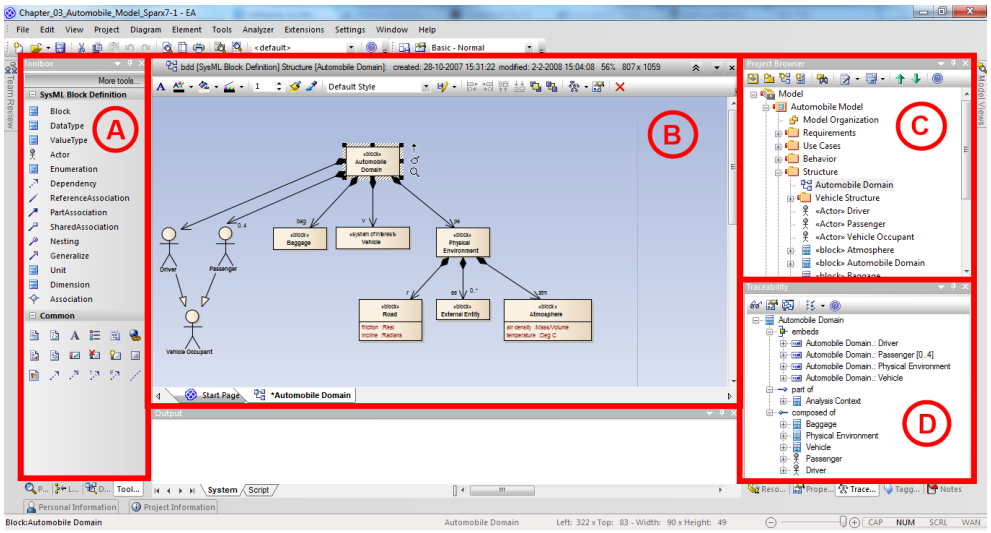


Figure 5.2: Example of Enterprise Architect's interface. In the figure we can identify the following components: A) Toolbox, B) Diagram Viewer, C) Project Browser, and D) Traceability Window.

[16] http://support.microsoft.com/kb/275561/ (accessed 01-06-2012)

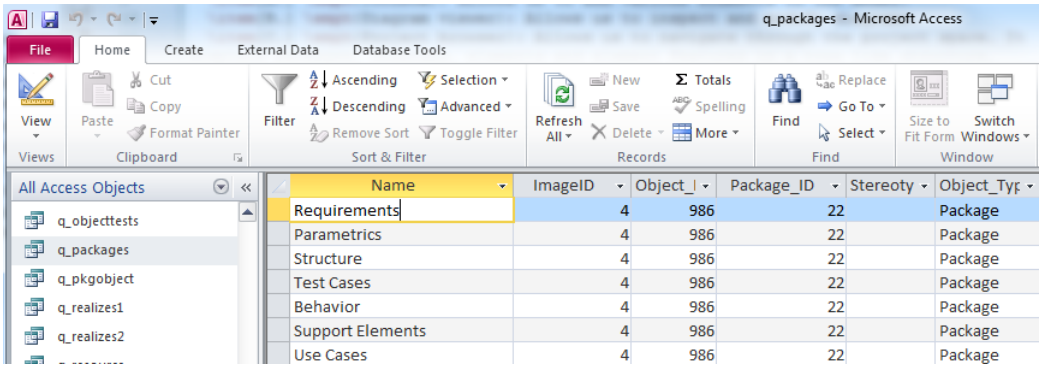


Figure 5.3: Browsing an EAP file with Microsoft Access 2010.

However, applying version control to the database as a whole would require to use the *file-locking* mechanism. In this revision control variant, concurrent access to files in a central repository is prevented using locks. This is often necessary when dealing with files in binary format (non line-based text files, such as artwork and sound), where it is not possible to merge changes. Still, the file-locking mechanism means only a single user may work on a model at a time, which especially hinders collaborative work. To overcome this limitation, EA exports the packages in a model to individual XMI files which are kept under revision control (and not the EAP file). Be aware that these files are still treated as binary files, and thus disallowing merge operations. However, dividing the model into smaller parts allows multiple users to work on different parts of the model simultaneously. Figure 5.4 provides an example of the files stored in an EA project working copy directory.

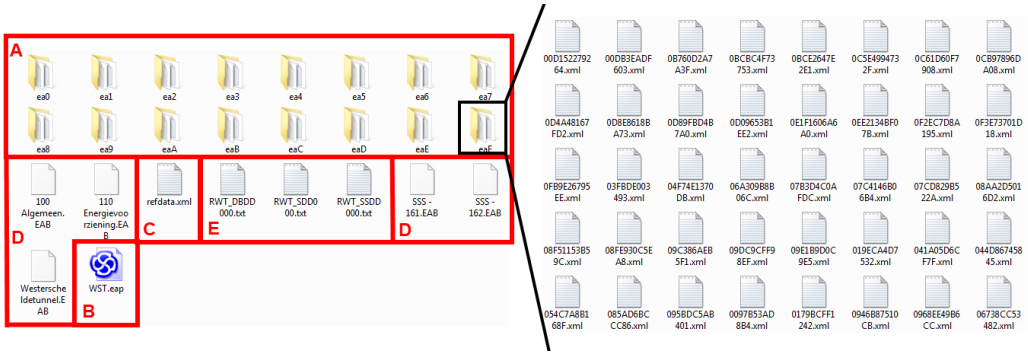


Figure 5.4: Example of EA SysML project working copy directory. On the left, we can identify the following: A) multiple folders containing the XMI files, B) the EAP file, C) the reference data file (contains data such as EA scripts), and D) EAB files, and E) resource files (such as text files and spreadsheets). On the right, the collection of XML files in XMI format in such a folder is shown. Notice that file name does not reflect package name.

SysML model data is kept under revision control at Soltegro using an Apache Subversion[17] (SVN) repository. A user may work on the revision of a project from the SVN repository by performing a *check-out* command on a local directory using the TortoiseSVN client[18]. From EA, a model may be imported selecting the root XMI package file. However, since every package is exported to an individual XMI file, a large collection of files would need to be placed under revision control for complex projects, where it may become difficult to find the corresponding root package. To alleviate this, EA uses Model Branch files to retrieve information about the root package file and import a model branch. When a user performs a *check-out* operation on a package, EA commands the revision control system to check-out the corresponding XMI file. The latest revision of the file is put into the user's working copy directory overwriting any previous revision. With a *check-in* operation, EA exports a package as an XMI file overwriting any existing copy, and then the revision control system performs a commit on the new file.

5.2 Existing UML/SysML Measurement Tools

Since it is required to measure SysML models, this section reviews a couple of promising projects which may provide the required data: SDMetrics and the EmpAnADa project.

5.2.1 SDMetrics

While reviewing possible UML/SysML measurement techniques, we were able to evaluate the SDMetrics[19] software tool. Juergen Wuest, the tool's author, provided us with a free academic license (excludes commercial use). SDMetrics is an object-oriented design quality measurement tool that analyzes the structure of UML and SysML models (refer to figure 5.5 for a couple of examples showing the tool's interface). The tool has two main features:

1. *Object-oriented measures* used to measure design, coupling and complexity.

2. *Design rule checking* aimed at detecting incomplete, incorrect, redundant, or inconsistent design and style problems.

SDMetrics provides metamodel extensions and metrics support for OMG's SysML v1.2 through *SysML 1.2 design quality metrics and rules*[20], a collection of XMI transformation files. Note however, that these serve as a demonstration of how SDMetrics can be adapted to deal with UML 2 profile extensions. The metrics provided assess system quality in MBSE, including: assessment of the completeness, correctness, consistency of the model; adherence to modeling conventions; estimating design and development effort; and monitoring of design and development progress.

[17] `http://subversion.apache.org/`
[18] `http://tortoisesvn.net/`
[19] `http://www.sdmetrics.com/`
[20] `http://www.sdmetrics.com/PF_SysML.html` (accessed 01-08-2012)

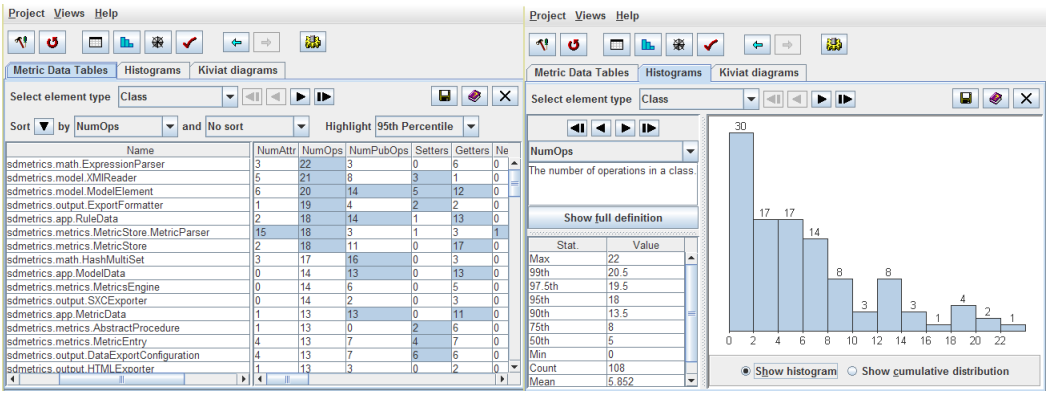


Figure 5.5: Examples of SDMetrics' interface. On the left, the data presented in table format per element (note the ability to highlight value beyond an specific threshold). On the right, the data as an histogram per class.

It should be kept in mind that SDMetrics is unable to read XMI files generated by EA, since SDMetrics' default XMI transformations expect all model elements in the XMI file to have and XMI ID, while EA does not provide XMI IDs for all the model elements. To solve this, *XMI 1.2 Transformation File for Enterprise Architect*[21] is available from the tool's site. SDMetrics also has a free Java implementation featuring its core functionality as open source (AGPLV3 license for non-commercial use). This distribution provides full XMI import, measurement, and rule checking capabilities, also including the XMI parser, metrics and rule engine.

5.2.2 EmpAnADa project

EmpAnADa[22] (Empirical Analysis of Architecture and Design Quality) is a project of the Eindhoven University of Technology (as part of the System Architecture and Networking group) led by M.R.V. Chaudron and C.F.J. Lange. The project is concerned with the development of techniques to improve the quality of UML (and nowadays SysML) models. Techniques derived from this project include rules, metrics, visualization techniques, and modeling conventions. These tools include the following:

- **MetricView**: Analysis and visualization of model quality and model evolution.
- **DICT Toolset**: Design Implementation Conformance Tools. Techniques to assess the conformance and differences between the UML model of a system and its implementation.
- **SquAT**: Sequence Diagram Analysis Tool. Assessment of the conformance of UML sequence diagrams to predefined rules.

[21] `http://www.sdmetrics.com/PF_EA.html` (accessed 01-08-2012)
[22] `http://www.win.tue.nl/empanada/` (accessed 01-10-2012)

- **SAAT**: Software Architecture Analysis Tool. Metrics that combine information from different UML diagram types.

The EmpAnADa project also includes research in the visualization of UML models using metric and architectural information [71] [32].

Collaboration with Adry Ferwerda, one of Chaudron's students, was proposed. His research project focuses on the evaluation of software development using a model driven approach. Even though it did not involve SysML models, there were overlapping interests, such as identifying regions in the model with low quality and visualizing how changes in one element affect other elements in the project. However, Adry expressed that these points had a low priority and were not planned to be addressed before the end of this research project. Thus, further contact was not pursued, but future endeavors could review the measurements proposed and/or developed to address these points in a (non-SysML) model, and how they impact user insight into the model's quality (evolution).

5.2.3 Evaluation

A possible solution would involve using a tool from (or similar to) those previously described, which will be tasked with reading, interpreting and measuring the model data from the XMI files. However these tools only analyze one file at a time, and since EA doesn't export all the data about every package (often relies on the use of *stubs*, see section 6.1.1), reading individual files would result in incomplete information. Instead, an automatized method would need to be devised to read all the XMI files, or combine these files into one single file, or enforce exporting models without stubs. Even so, project meta-models would still require custom rules which take into account their specific structures.

5.3 Extending Enterprise Architect Functionality

There are three main ways to add extra user-custom functionality to Enterprise Architect: scripts, custom MDG technology, and add-ins.

Scripts: EA has a built-in *Script Editor* to write and run custom scripts using the JavaScript[23], JScript[24] or VBScript[25] languages. Scripts are often used to perform time consuming and repetitive GUI tasks. Although scripts can be saved to the file system, they are stored in the model, and thus, are only visible in the model in which they were created.

[23] http://msdn.microsoft.com/en-us/library/ms970435
[24] http://msdn.microsoft.com/en-us/library/hbxc2t98
[25] http://msdn.microsoft.com/en-us/library/t0aew7h6

MDG Technology: Starting from version 8.0, the built-in wizard in EA allows a user to create Model-Driven Generation (MDG) technology files. These files extend (or limit, depending on the user's needs) EA's functionality, providing additional toolboxes, UML profiles, patterns, templates and other modeling resources (see the Sparx website[26] for a list of third-party examples). Note that support for SysML in EA is provided as an extension through MDG technology, as shown in figure 5.6. Tutorials showing how to develop MDG technology for EA are available on the web[27].

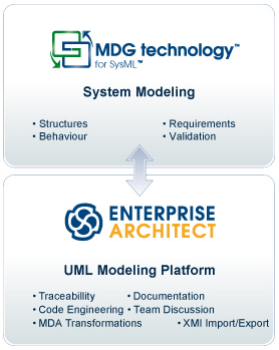


Figure 5.6: SysML support in EA through MDG technology.

Add-ins: Add-ins allow to add functionality to EA and extend the Automation Interface (which provides access to the internal EA models). Add-ins can be used to add (sub) menus and access all user-interface events (such as mouse clicks and file changes). Add-ins require installation, but do not need to be configured. The Sparx website provides a tutorial [28] to create an add-in using C#.

Evaluation Choosing one of these methods would allow us to extend EA, a tool engineers are familiar and comfortable with. EA also provides all the model data necessary and methods to access it. However, choosing this route would make the framework EA-dependent, meaning it would be bound to the tool to perform the necessary measurements. This could prove detrimental if, for example, another modeling tool is used in the future. Also, every time a model needs to be analyzed, the corresponding EAP file should be reconstructed (as only XMI and EAB files are stored in the repository). Besides the problems described above with respect to distribution and availability of these extensions, it is also desired to take model evolution into account, thus requiring reading data from the repository (only available from XMI files).

5.4 VoSMA Software Tool

A third option encompasses developing a custom framework, such as a stand-alone application. The main functions of this tool are to read the relevant model data from the XMI files. Another variant of this option involves reading the data from the .EAP database instead. Although querying information from a database is a simpler process than developing the framework described above, this option was not pursued since, as explained before, the EAP file is not kept under revision control, has to be generated through EA, and

[26] `http://www.sparxsystems.com.au/products/3rdparty.html` (accessed 01-11-2012)

[27] `http://www.tigerteam.dk/2011/how-to-develop-mdgs-for-enterprise-architect-part-1/` (accessed 01-11-2012)

[28] `http://community.sparxsystems.com/tutorials/tool-integration/create-your-first-c-enterprise-architect-add-10-minutes` (accessed 01-11-2012)

Project name	Revision	Total XMI file size	EAP file size
Nijverdal	5000	82,0 MB	566 MB
Nijverdal	11043	94,0 MB	572 MB
Westerschelde	5000	76,7 MB	133 MB
Westerschelde	9571	119 MB	224 MB

Table 5.1: XMI and EAP file size comparison.

it is much larger than the XMI file collection (refer to table 5.1). Thus, reading the model data from the XMI file collection is the preferred solution. The software is tasked with the analysis of this data according to the defined criteria, and generation of the required output. Advantages of this approach include accessibility and control over the whole model (all the model data is available from the XMI files), tool independence (other modeling tools also use XMI files to exchange model data), custom metric rule definition (thus providing the ability to describe and implement our own measurement criteria), and custom output data generation. Disadvantages involve the need to develop and implement a system to read, parse, store, interpret, reconstruct and traverse SysML model data. Still, we consider that the advantages of this variant heavily outweigh the disadvantages. It is expected that, when implemented, the framework will provide simple and flexible (yet powerful) means to create new custom rules for analysis, measurement and visualization.

Figure 5.7: Proposed approach using the *extract-abstract-present* paradigm.

To this end the *VoSMA* software tool (originally the acronym for *Visualization of SysML Meta-model Architecture*) was designed and implemented. The tool provides the framework responsible for the analysis, measurement and generation of the output data required to answer the main research goals. The tool's design follows the *extract-abstract-present* paradigm [73], resulting, as shown in figure 5.7, into three main steps: extraction (data gathering), abstraction (knowledge inference), and presentation (information interpretation). The goal is to develop a tool capable of reading SysML model data from a XMI file collection, extract the corresponding project meta-model, measure its elements, and export the structural and metric data to output files. These files serve as input for a third-party visualization tool, tasked with the presentation of this data to the user, as shown in figure 5.8.

A set of research sub-questions have been defined for each step of the proposed approach, which are further analyzed and addressed in the following chapters.

Operating System	Windows 7 Home Premium SP1
System Type	64-bit
Processor	Intel Core2 Duo @ 2.13GHz
Memory	4.00 GB
Java Development Kit	7 update 7

Table 5.2: Development machine specifications.

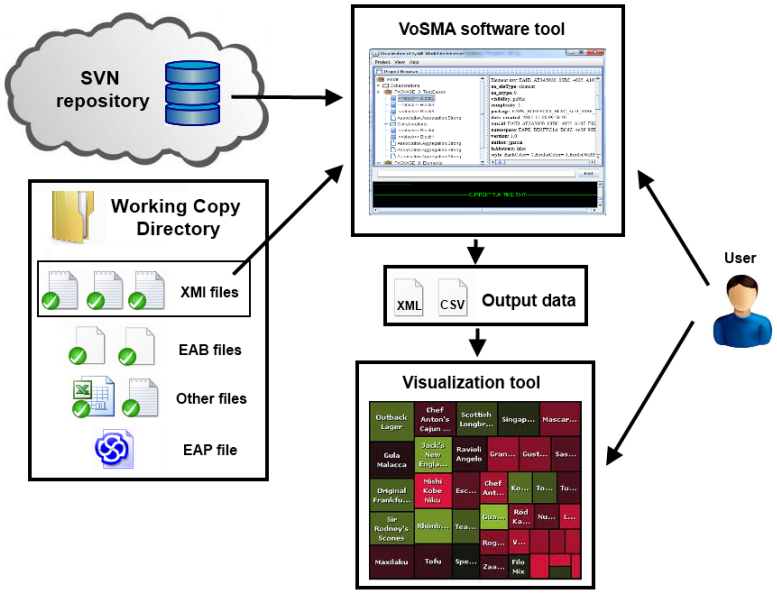


Figure 5.8: Overview of the proposed approach using the VoSMA tool. Note that model data is read from the XMI file collection in the working directory or SVN repository.

5.4.1 Extraction - Data Gathering

The first phase is concerned with the identification of the available data sources and the recollection of the necessary data to produce the required input. This phase is further described in chapter 6, which provides answer to the following five research sub-questions:

RSQ.1a - *What are the data sources available at our disposal?*

RSQ.1b - *What data do we need to analyze for our research?*

RSQ.1c - *How can we read the necessary data?*

RSQ.1d - *How can we effectively store the required data?*

RSQ.1e - *How can we provide element traceability?*

5.4.2 Abstraction - Knowledge Inference

The second phase involves the extraction and analysis of the project meta-model, which is measured using the appropriate metrics, leading to a set of output data. This phase is further described in chapter 7, which provides answer to the following three research sub-questions:

RSQ.2a - *How can we reconstruct the project meta-model structure?*

RSQ.2b - *What measurements should be performed on the project meta-model architecture?*

RSQ.2c - *What output data do we need to generate?*

5.4.3 Presentation - Information Interpretation

The final phase mainly involves presenting the output data to the users. This phase is further described in chapter 8, which provides answer to the following two research sub-questions:

RSQ.3a - *How can we efficiently present the generated data to systems engineers?*

RSQ.3b - *How can we represent the project meta-model architecture evolution through development?*

However, it is also required to evaluate the satisfaction level of the presented results, as one of the main research objectives is to aid engineers to gain insight into the project meta-model. This is discussed in chapter 9.

RSQ.3c - *How satisfied are systems engineers with the presented data?*

Chapter 6

Data Gathering

The previous chapter examined the current situation, where multiple alternatives where proposed and evaluated. After concluding that a new framework had to be developed, a three step approach based on the *extract-abstract-present* paradigm was proposed. Each step was refined with a set of research sub-questions, and a software artifact was introduced to provide the necessary answers.

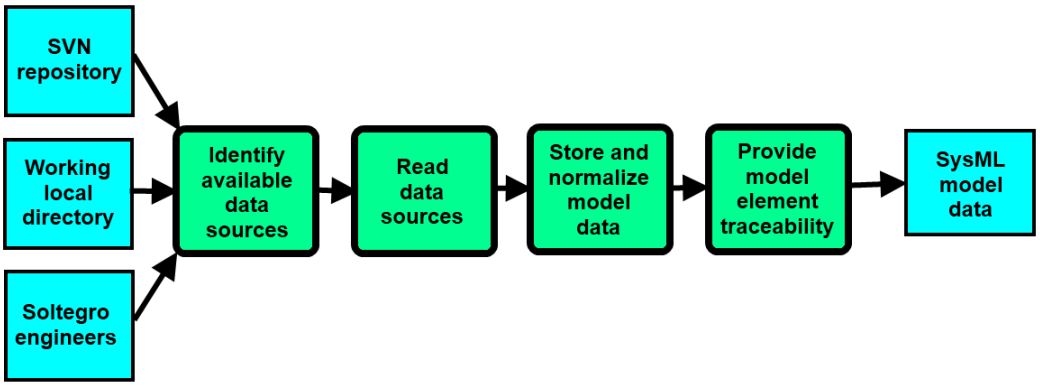


Figure 6.1: Overview of the *data gathering* phase.

This chapter analyzes the *data gathering* phase (see figure 6.1 for an overview). Section 6.1 identifies, analyzes and filters the available data sources. Section 6.2 explains how the required data is obtained and read, while section 6.3 describes how this data is stored and normalized. Lastly, section 6.4 presents means to provide element traceability. The result of this phase is the complete SysML model data, and the means to transverse the model.

6.1 Identifying Available Data Sources

This section provides answer to two research sub-questions: **RSQ.1a -** *What are the data sources available at our disposal?*, and **RSQ.1b -** *What data do we need to analyze for our research?* Three main data sources were identified in our environment:

- **EA working directory**: Contains all the data used by EA, including the model data. Section 5.4 explained why retrieving the model data from the XMI file collection is the preferred choice (XMI format discussed in section 6.1.1). Most element data is required, but diagram and visual information can be left out.
- **The SVN repository**: Provides the various model revisions and revision history metadata through change-log (format discussed in section 6.1.3).
- **Soltegro systems engineers**: Provide needs and constrains for this project. They also determine the appropriate measurement and visualization preferences.

6.1.1 XMI file format

This section examines EA's proprietary native XMI format. As explained earlier, every XMI file holds data about a single package in the model, including all information about every element in its sub-packages (refer to figure 6.2 for an example). However, after version 4.5, EA only saves *stub* information about any nested packages. When using version control, this method ensures that information in a nested package is not inadvertently over-written by a top level package (and thus, only the top level package is modified when checking out). The option to save nested packages as stubs is enabled (and recommended) by default in EA.

```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3">
    <XMI.header>
    <XMI.content>
    <XMI.difference/>
    <XMI.extensions xmi.extender="Enterprise Architect 2.5">
</XMI>
```

Figure 6.2: XMI file example.

EA's XMI file format is as follows:

1. **Declaration**, which consists of:
   a) An **XML version processing instruction**: *xml version="1.0"*.
   b) An optional **encoding declaration** that specifies the character set, which follows the ISO-10646 (also called extended Unicode) standard: *encoding="windows-1252"*.
2. An **schema XMI** element (<*XMI*>), which consists of:
   a) An **XMI version processing instruction**: *xmi.version="1.1"*.
   b) An optional **encoding declaration** that specifies the character set: *xmlns:UML="omg.org/UML1.3"*.

This XMI schema contains four nodes:

a) **Header** (<*XMI.header*>): The header is the same across all XMI files (see figure 6.3), and contains the name of the tool that generated the file, and the exporter version, which has remained unchanged since EA version 2.5.

```
<XMI.header>
    <XMI.documentation>
        <XMI.exporter>Enterprise Architect</XMI.exporter>
        <XMI.exporterVersion>2.5</XMI.exporterVersion>
    </XMI.documentation>
</XMI.header>
```

Figure 6.3: XMI header.

b) **Content** (<*XMI.content*>): The main source of element data (see figure 6.4 for an example). It is important to notice that the data over the associated package can always be found in the following path:

"XMI.content/UML:Model/UML:Namespace.ownedElement/UML:Package"

Almost every element in a model possesses an unique identifier, also known as UID (*xmi.id* attribute). The element's UID and name (if any) can always be found in the attributes. Data about every other element in the package can be found in the "UML:Namespace.ownedElement" node.

```
<XMI.content>
    <UML:Model name="EA Model" xmi.id="MX_EAID_F9E5AC5D_3078_4f6b_AE7F_00D152279264">
        <UML:Namespace.ownedElement>
            <UML:Class name="EARootClass" xmi.id="EAID_11111111_5487_4080_A7F4_41526CB0AA00" isRoot="true" isLeaf="false
            <UML:Package name="Eisen aan interne systeemgegevens" xmi.id="EAPK_F9E5AC5D_3078_4f6b_AE7F_00D152279264" isR
                <UML:ModelElement.taggedValue>
                    <UML:TaggedValue tag="documentation" value="In dit hoofdstuk zijn de eisen gesteld aan de gegevensop
                    <UML:TaggedValue tag="parent" value="EAPK_5998C8E7_E913_4b67_8556_0BCBC4F73753"/>
                    <UML:TaggedValue tag="created" value="2012-05-07 15:25:06"/>
```

Figure 6.4: XMI content node example.

c) **Differences** (<*XMI.difference*>): Holds XML elements representing differences to a base model.

d) **Extensions** (<*XMI.extensions*>): Contains data to transfer that does not conform to the metamodel(s) in the header. Extensions can be used or ignored by other modeling tools. In EA it is mainly used for stubs (declared as an element, but not used for interactions) and NoteLink (the connector of a Note) elements.

Information about every element in a package is stored in a node with the name of its SysML class type. This node is assigned multiple attributes (such as the element's name and UID), optionally followed by a list of associated tagged values, owned elements and other related data.

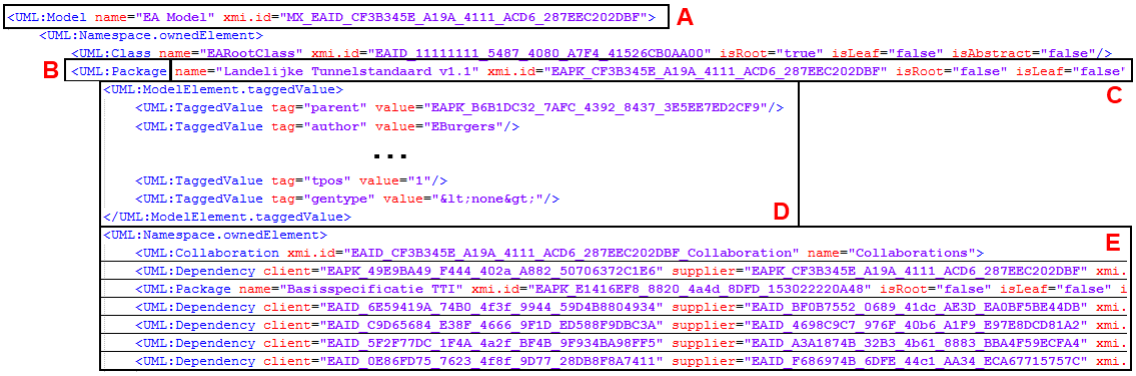


Figure 6.5: Example of an XMI element node, showing A) the owner node, B) the node type, C) attributes, D) tagged values, and E) owned elements.

6.1.2 Model Branch File format

Reading a model from a SysML project can be done in two ways: a) read the whole collection of XMI and retrieve the corresponding model root(s), or b) use a branch file which points the model root of the branch. The model is then reconstructed by recursively reading the owned packages, which point to their corresponding files. The first option reads the whole project and finds all the models, while the second only reads what is necessary to reconstruct the desired model.

```
<BranchVCSpec>
    <PkgGUID>{90305318-5035-4112-ADF2-739634124A7C}</PkgGUID>
    <XMLPath>ea7\739634124A7C.xml</XMLPath>
    <Config>
        <GUID>devzing</GUID>
        <LocalPath>%devzing%</LocalPath>
        <Type>2</Type>
    </Config>
</BranchVCSpec>
```

Figure 6.6: Contents of the Westerscheldetunnel.EAB file. Note that the address marked in red points to the relative path of the model's root XMI file package.

EA's Model Branch File (.EAB file extension) is a text file used to record information version control configuration settings. The file contains the relative path to the XMI package file used as root of the model to be imported (as shown in figure 6.6).

6.1.3 The SVN change-log data

The SVN repository revision metadata can be accessed through the *change-log*, which stores the following information:

- *Revision number*: The incremental number assigned when the revision was committed.
- *Author*: The user who committed the revision.
- *Date*: The date when the revision was committed.
- *Action Performed*: There are four actions possible per file when committing a revision: M (modified), A (added), D (deleted), or R (replaced).
- *Path*: Files and folders that were changed with the revision.
- *Message*: Full log message for the committed revision (if any).

This data will be mainly used to visualize the model evolution through different revisions.

6.2 Reading Data Sources

This section addresses research sub-question **RSQ.1c -** *How can we read the necessary data?*, describing the tools used to extract, filter and/or generate the required data.

SVN repository reader: This module is responsible for reading the XMI files and change-log data from the repository. The module was built using the SVNKit[29] Java Subversion library. Other possible alternatives included JavaHL[30] and SvnClientAdapter[31], but were discarded due to lack of documentation, support and/or working examples.

Streaming API for XML (StAX): In the Java programming language community, there are traditionally three options to read data from an XML file:

- *Event-based APIs (streaming)*: SAX[32] (Simple API for XML) is a prime example, where an event-based sequential access parser only operates on portions of the XML document at a time (push model). However, once started, it goes to the end of the document and the caller must be ready to handle all of the events in one shot.
- *Tree-based APIs (document object model)*: DOM[33] (Document Object Model) implementation relies on this technique, where a hierarchy-based parser recreates an object model of the entire XML document in memory (the model built is usually larger than the original XML document).
- *String manipulation*: Sometimes implemented due to performance or memory constraints, but often hardly scalable or reusable.

[29] http://svnkit.com/
[30] http://subclipse.tigris.org/wiki/JavaHL
[31] http://subclipse.tigris.org/svnClientAdapter
[32] http://docs.oracle.com/javase/1.4.2/docs/api/javax/xml/parsers/SAXParser.html
[33] http://docs.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/package-summary.html

Table 6.1 provides an overview of XML parser APIs for Java. *Streaming pull parsing* refers to model in which the client only gets (pulls) XML data when it explicitly asks for it (used by StAX). In the *streaming push parsing* variant the parser sends the data whether or not the client is ready to use it at that time (used by SAX). Several advantages of the pull method include: capable of reading multiple documents with a single thread, elements unnecessary to the client can be ignored, and the client controls the application thread (methods are called on the parser when needed). Our project requires reading a writing XML files, may use the 'forward only' option, and an efficient memory usage is preferred. XPath and XML modification capabilities are not necessary. From these requirements it appears the StAX API is the prime candidate for this project.

Feature	StAX	SAX	DOM	TrAX
API Type	Pull, streaming	Push, streaming	In memory tree	XSLT Rule
Ease of Use	High	Medium	High	Medium
XPath Capability	No	No	Yes	Yes
CPU and Memory Efficiency	Good	Good	Varies	Varies
Forward Only	Yes	Yes	No	No
Read XML	Yes	Yes	Yes	Yes
Write XML	Yes	No	Yes	Yes
Create, Read, Update, Delete	No	No	Yes	No

Table 6.1: XML parser API feature comparison[34]

StAX is defined in the JSR 173 specification[35]. VoSMA uses the Sun Java Streaming XML Parser[36] (SJSXP) implementation. The core StAX API falls into two categories: *cursor API,* and *event iterator API*. The cursor variant mirrors the SAX API: it represents a cursor which traverses through an XML document from beginning to end. This cursor may point to one element at a time, and always moves forward (never backwards). The cursor API methods return XML information as strings, which minimizes object allocation requirements. On the other hand, the event iterator variant somewhat resembles the DOM API: it represents an XML document stream as a set of discrete event objects pulled by the application and provided by the parser in the order in which they are read. The event iterator API is more flexible and extensible, but the cursor API is more CPU and memory efficient, while using smaller code. This variant was chosen as advantages of the iterator API are not required by our solution, while efficiency and memory usage are important aspects.

SysML element reader: This module is tasked with the filtering and interpretation of the data from the StAX parser. The module tries to identify individual elements, assigning all the corresponding data. Unnecessary data is filtered out (though given the properties of StAX, all the XML nodes are traversed). Element data is stored in-memory (discussed in the next section).

[34] `http://docs.oracle.com/javase/tutorial/jaxp/stax/why.html`
[35] `http://stax.codehaus.org/`
[36] `http://sjsxp.java.net/`

6.3 Model Data Storage

This section addresses research sub-question **RSQ.1d -** *How can we effectively store the required data?*. Every element in a SysML model has a unique identifier and a list of associated data, consisting of string pairs: a data identifier and a data value (e.g. name = block, author = Eric, complexity = Medium). The Java HashMap[37] data structure is used to hold all the element data in the model. HashMap is a table implementation of the *Map* interface, which relies on the principle of *hashing*, storing data as pair with the *put(key, value)* method, and retrieving information using the *get(key)* method. Note that HashMap accepts *null* values, but it is not synchronized, and thus requires external synchronization if accessed from multiple threads.

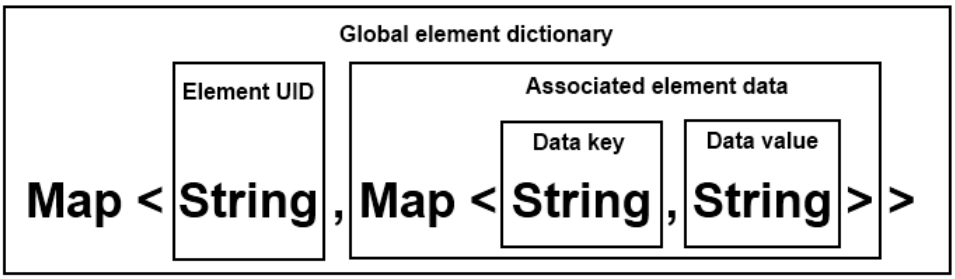


Figure 6.7: Global element dictionary structure. Note that the Map interface declared is implemented as a HashMap

The HashMap structure used to store every model elements is called the *elementDictionary*, which keeps all the element UIDs as key, and where the values hold a HashMap with the associated element data. Figure 6.7 shows this concept. It is important to notice that in this dictionary stubs are, whenever available, replaced with full element data.

6.4 Model Element Traceability

This section provides answer to research sub-question **RSQ.1e -** *How can we provide element traceability?* In SysML, diagram elements can be essentially divided into blocks and connectors. Connectors associate two elements (although sometimes connectors may appear to branch and connect multiple components, they are actually different connectors). When analyzing models, it is important to provide efficient element traceability, which identifies the correlation between entities (i.e. how an element is related to other elements in the model).

Although all the model data is available from *elementDictionary*, it becomes a cumbersome task to find information when dealing with very large dictionaries from complex projects. Take figure 6.8 as an example. Three model elements can be

[37] `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/HashMap.html`

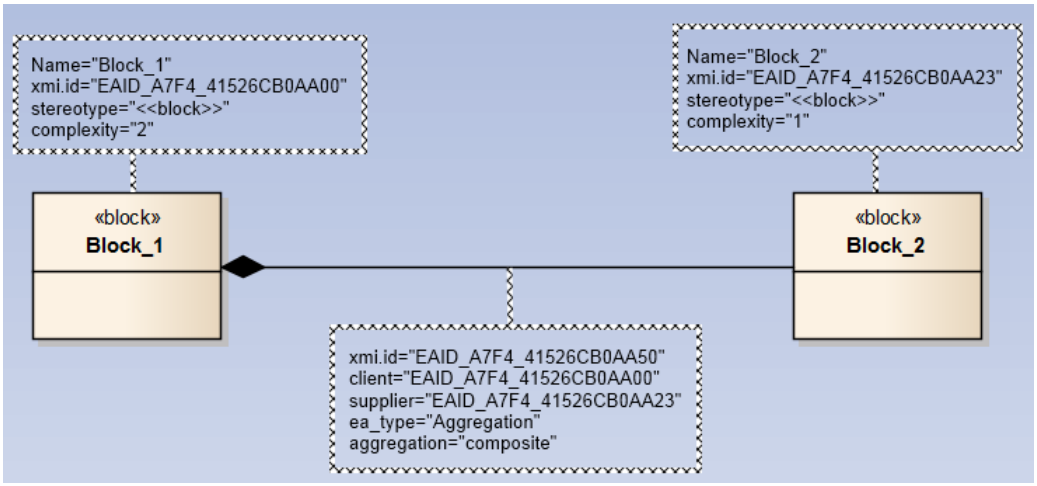


Figure 6.8: Element traceability example. Note how the connector is a separate entity, and no connection information is stored in the elements' data.

distinguished: two blocks (*Block_1* and *Block_2*) and a composite aggregation connector, meaning that *Block_2* is part of *Block_1*. Imagine that given Block_1 it is required to identify all of its parts. To achieve this, it would be necessary to iterate through every entry in the dictionary, check if it is a connector of the composite aggregation type, and check wherever the client's UID matches the elements' UID (meaning the supplier is one of its parts).

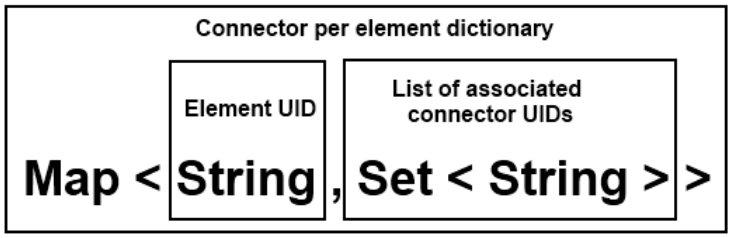


Figure 6.9: Structure of *associationDictionary*. The Map interface is implemented as a HashMap, and the Set interface is implemented as a HashSet

To alleviate this situation, a second (auxiliary) dictionary was implemented: *associationDictionary*. This structure does not contain any new information but rather speeds up searches. This approach is similar to adding an index to a SQL database table[38], where indexes are created on columns which are accessed frequently, so information can be accessed quickly. As with SQL indexes, the auxiliary dictionary improves search speed, but the more indexes used, the more memory space is required (however, this dictionary only holds UIDs as string values, and only for elements with connectors). Another

[38] http://www.w3schools.com/sql/sql_create_index.asp

potential drawback involves the modification of the source model data while in use, since it would require to update all the auxiliary dictionaries as well, which is not the case in the current implementation (source data is static read-only and never modified).

In Java, the HashMap *get()* and *containsKey()* methods result in a lookup computational complexity[39] of $\Omega(1)$ and O(n). However, the worst case is approached when the hash function does not disperse the elements properly among the buckets, the initial capacity is too high, or the load factor too low. Since this is not the case with this implementation, it can be assumed that these functions have a lookup of approximately O(1). Thus, finding the UIDs of all parts of an element would result in O(n) on the *elementDictionary* (iterating through the whole dictionary), where *n* is the total number of elements in the model. With the auxiliary dictionary, it would take O(1) on the *associationDictionary*, plus O(n), where *n* represents the number of connections of this element, resulting in a much faster implementation. Thus, it is recommended to use auxiliary dictionaries on values that are often accessed.

[39] `http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html`

Chapter 7

Knowledge Inference

The previous chapter discussed the first phase of the proposed implementation approach, where multiple data sources were identified and read. After describing the means to normalize and store the data, element traceability was provided. This chapter examines the second phase: *knowledge inference*, and all related research sub-questions. The main goal is to analyze and process the input data, extract and measure the corresponding project meta-model, and generate the output necessary for the next chapter.

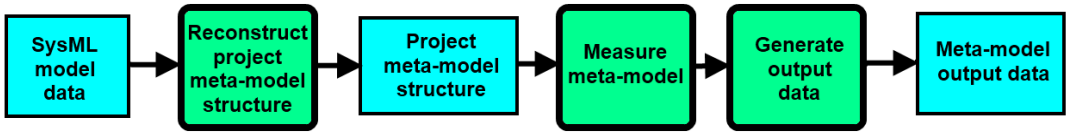


Figure 7.1: Overview of the *knowledge inference* phase.

Figure 7.1 shows an overview of this phase. Section 7.1 analyses the project meta-model and all involved elements and relations, from which the meta-model's architecture can be reconstructed. The next step is to measure the elements in this structure, based on the criteria expressed by the Soltegro engineers. Section 7.2 describes how the Goal-Question-Metric approach was applied to generated goal oriented measurements. The corresponding output data is generated in section 7.3.

7.1 Reconstructing the Project Meta-Model

This section provides answer to research sub-question **RSQ.2a -** *How can we reconstruct the project meta-model structure?* As explained in section 1.3, tunnel project models in SysML are often based on the IEEE J-STD-016-1995 standard at Soltegro. Note however that the structure described may be subject to modifications due to project constraints. Figure 7.2 shows a basic example of a model following this approach. Basically, the methodology works with levels, and it is applied as follows: a *top-requirement* is defined and placed in level 1. This element is further refined in a set of requirements, which are placed in level

2, which in turn are further refined (or copied) in level 3 and so on. Also, a *top-system* (a block), is defined, placed in level 1, and connected to the top-requirement. This element is made out of parts or sub-systems, which are placed in level 2, which in turn may contain sub-parts/systems in level 3 and so on. These (sub)systems are connected to one or more requirements in the same level through a *«satisfy»* relation. Thus, the following elements can be identified in the model (see table 7.1 for an overview):

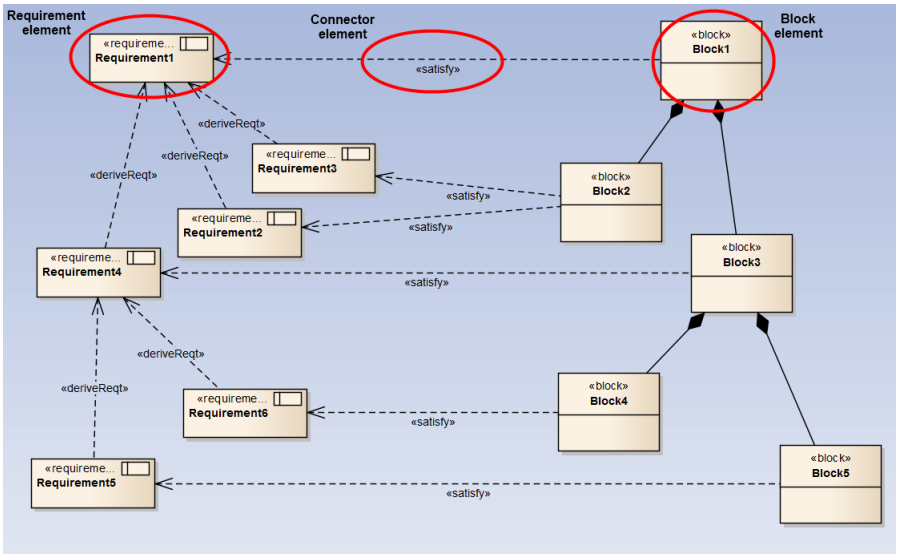


Figure 7.2: SysML model example following the modeling methodology specified. Note that the model consists of three types of elements: block, requirements and connectors.

- **Blocks**: the block is a very general modeling concept and the prime modular unit of structure. It is used to model a wide variety of entities with a structure, such as systems, hardware, software, physical objects, and abstract entities. It may also define a type of system, system component, or item that flows through the system, as well as conceptual entities or logical abstractions. Each block describes a set of uniquely identifiable instances that share the block's definition.
- **Requirements**: As discussed in section 4.5, requirements from stakeholders are specified as system requirements. These system and physical requirements are modeled as component requirements in SysML.
- **Connectors**:
  - **Derive**: connects a derived requirement and a source requirement. This association often shows relationships between requirements at different levels of the specification hierarchy. It is also used to represent a relationship between requirements at the peer level of the hierarchy, but at different levels of abstraction (e.g. more detailed requirements may be related to the original requirements through a derive relationship).

- **Copy**: relates a copy of a requirement to its original to support reuse of requirements. A requirement exists in one namespace or containment hierarchy and has specific meaning in its containing context. To support reuse of the requirement, the copied requirement is a requirement whose text property is a read-only copy of the text property of the source requirement, but with a different identifier.
- **Composite aggregation**: Part properties, or just *parts*, are used to model the block composition hierarchy. This type of hierarchical composition (whole-part relationship) of blocks is often used in the breakdown of equipment or bill of materials.
- **Satisfy**: used to assert that a model element satisfies a particular requirement (actual proof that this assertion is correct is achieved with the verify relationship).

In SysML, the *derive*, *copy* and *satisfy* relationships are treated as *dependencies*, indicating that changes in one end may result in corresponding changes on the other end.

Element name	Stereotype	SysML Type	Association Flow
Block	*«block»*	Block	-
Requirement	*«requirement»*	ClassifierRole	-
Derive	*«deriveReqt»*	Dependency	Requirement –> Requirement (next level)
Copy	*«copy»*	Dependency	Requirement –> Requirement (next level)
Satisfy	*«satisfy»*	Dependency	Block –> Requirement (same level)
Composite aggregation	-	Association	Block –> Block (next level)

Table 7.1: Overview of relevant SysML model elements. Note that the multiplicity of the relations shown under the 'Association Flow' column is 0..*. However, elements do not have multiple incoming associations of the same kind.

There are two important points which are not enforced by the modeling tool, but rather derived from the modeling methodology: a) the possible connection permutations and association multiplicity as stated in table 7.1, and b) only one requirement and one block are allowed in level 1 (the top-requirement and top-system respectively).

Modeling levels and element hierarchies: Given the nature of the modeling methodology, the model can be divided in two ways: into two element hierarchies, and into multiple levels (see figure 7.3). The two hierarchical structures are:

- Requirement hierarchy: Represents the *functional decomposition* of concepts that must be fulfilled (focuses on *what*).
- Block hierarchy: Represents the physical *structural (system) decomposition* into its interconnected components (focuses on *how*).

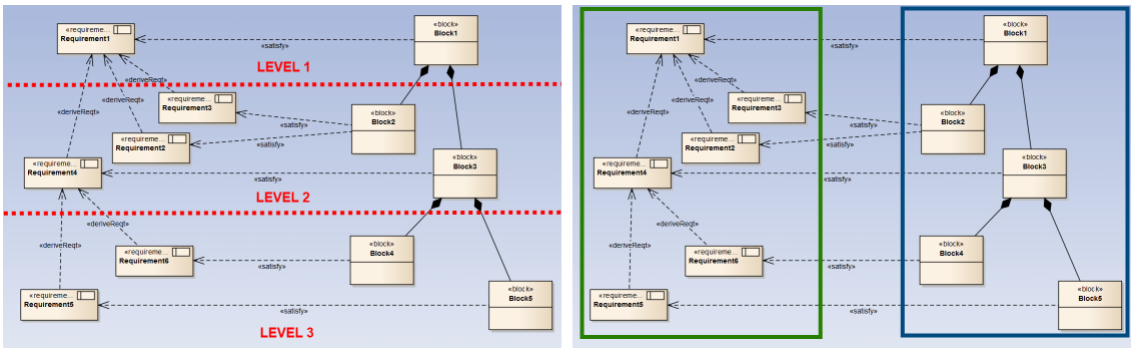


Figure 7.3: Model levels and hierarchies. The figure on the right shows the three levels in the model. The figure on the left shows two different element hierarchies: on the left the requirements hierarchy, on the right the block hierarchy (or block decomposition).

7.2 Measuring the Meta-Model Architecture

This section provides answer to **RSQ.2b -** *What measurements should be performed on the project meta-model architecture?* There are multiple measurements that can be performed on the model described in the previous section. The first step was to consult engineers about the values they considered important when measuring their models. The following were named:

- *Block complexity level*: Block elements can be assigned a *complexity* value by the engineer, which (often) can be set to one of the following three levels: 'Easy', 'Medium', and 'Difficult'. This value represents the estimated complexity to realize the system.
- *Requirement difficulty level*: Requirement elements can be assigned a *difficulty* value, which (normally) can be set to one of the following three levels: 'Low', 'Medium', and 'High'. The value represents the estimated difficulty to fulfill this requirement.
- *Number of «satisfy» associations*: The number of connectors with a *«satisfy»* stereotype associated with an element. Given the modeling methodology described before, every requirement is fulfilled by one system, but a system may fulfill multiple requirements. Thus, in the case of blocks this value should be above 0, in the case of requirements it should always be 1.
- *Fan-in and fan-out values*: In the hierarchies, the *fan-in* value represents the number of incoming connections from the previous level, while *fan-out* represents the number of outgoing connections to the next level. Note that block fan-in value is always 1 (except for the top-block, which is 0).

Note that both the *complexity* and *difficulty* values are not derived from a measurement or specification, but rather represent an engineer's personal estimate. Also, note that when

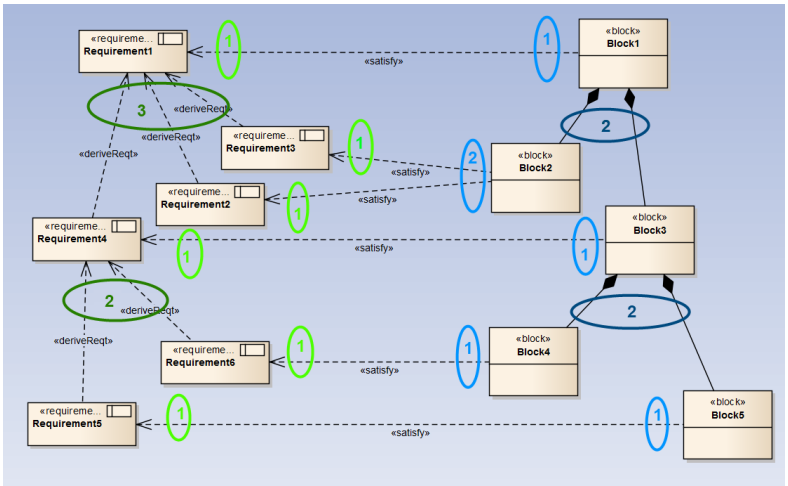


Figure 7.4: Example of *fan-out* and number of *satisfy* relations values. In dark blue: block *fan-out*, in dark green: requirement *fan-out*, in light blue: *satisfy* associations per block, and in light green: *satisfy* associations per requirement.

estimating the total value of a structure only *leaf* elements count (thus, elements with a *fan-out* value of 0). Take figure 7.2 as an example, where *Block3* in level 2 is decomposed in *Block4* and *Block5* in level 3. This means that Block3 is made of the combination of Block4 and Block5. Still, given the nature of the value, a block may have a lower complexity level than one of its descendants in the hierarchy.

These values can lead to multiple possible measurements for every element, as shown in table 7.2. However, presenting all this data to the engineers would provide no real insight or understanding about model's quality or hot-spot problems. Instead, measurement should only be performed towards an explicit stated purpose, called *goal-oriented measurement*. To achieve this, a framework methodology that follows this principle is applied: the Goal Question Metric approach.

7.2.1 Goal Question Metric Approach

The *Goal Question Metric (GQM)* approach is a mechanism for defining and interpreting operational and measurable software [6][7][66]. Its main principle is that measurement should be goal-oriented, and its main result is the specification of a measurement model with the following three levels:

1. *Conceptual level (GOAL):* A goal is defined for an object of measurement (e.g. product, process, resource) with respect to various models of quality.

2. *Operational level (QUESTION):* A set of questions is used to characterize the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint.

Metric Symbol	Type	Description
BLCK_LVL	integer	The block's level in the hierarchy
BLCK_CMPX_LVL	string	Block complexity level
BLCK_CMPX_VAL	integer	Block complexity value
BLCK_CMPX_VAL_TOT	integer	The complexity value of the block hierarchy
BLCK_CMPX_VAL_LVL	integer	The complexity value of the block level
BLCK_NR_TOT	integer	Total number of blocks in the hierarchy
BLCK_NR_LVL	integer	Total number of blocks in the hierarchy level
BLCK_SAT	integer	Number of associated requirements per block
REQT_LVL	integer	The requirement's level in the hierarchy
REQT_CMPX_LVL	string	Requirement difficulty level
REQT_CMPX_VAL	integer	Requirement difficulty value
REQT_CMPX_VAL_TOT	integer	The difficulty value of the requirement hierarchy
REQT_CMPX_VAL_LVL	integer	The difficulty value of the requirement level
REQT_NR_TOT	integer	Total number of requirements in the hierarchy
REQT_NR_LVL	integer	Total number of requirements in the hierarchy level
REQT_SAT	integer	Number of blocks associated to a requirement

Table 7.2: Overview of possible model measurements. Note that *block* element measurements are prefixed with 'BLCK_', while *requirement* element measurements are prefixed with 'REQT_'.

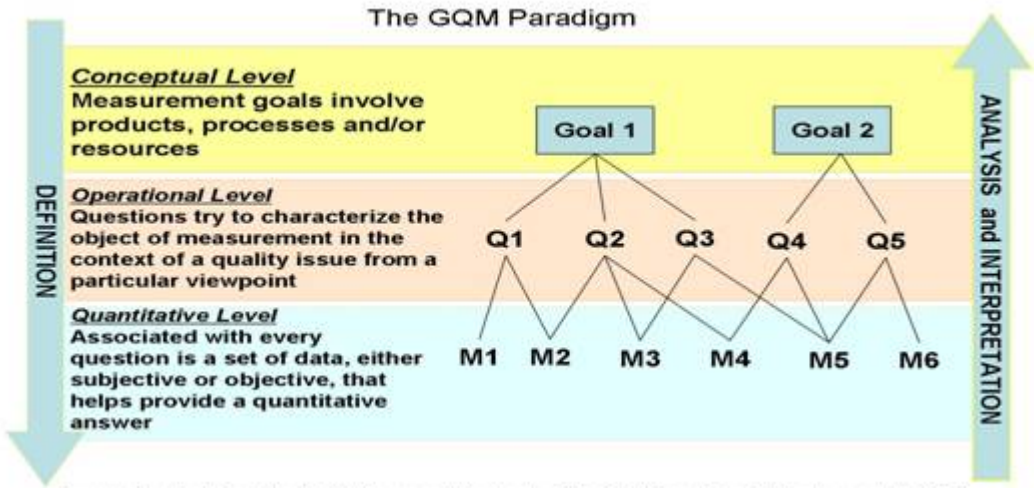


Figure 7.5: Goal-Question-Metric paradigm[40].

3. *Quantitative level (METRIC):* A set of data is associated with every question in order to answer it in a quantitative way. The data can be *objective*, if it depends only on the object that is being measured and not on the viewpoint from which they are taken (e.g. number of versions of a document, staff hours spent on a task, size of a program...), or *subjective*, if it depends on both the object that is being measured and the viewpoint from which they are taken (readability of a text, level of user satisfaction...).

[40] `https://goldpractice.thecsiac.com/practices/gqm/` (accessed 01-12-2012)

7.2.2 Goal

The GQM template [6] was applied to define the goal of measurements (see table 7.3). The next step involves creating a set of questions around the stated measurement goal.

Goal	Purpose	**To aid**
	Viewpoint	**Soltegro systems engineers**
	Issue	**to gain insight into the quality**
	Object(process)	**of project meta-model structures in SysML**

Table 7.3: Metric goal definition using the GQM template.

7.2.3 Questions

The following questions were defined with the aid of Soltegro engineers when asked to define the properties of a project meta-model (in terms of systems and requirements) that would provide insight into its quality.

*** MQ1 - What is the complexity value assigned to the system?**

Context: Systems are assigned a *complexity level* value by engineers. Elements with a high value are expected to require a higher amount of work. It may also be an indication that further decomposition is required.

Metric involved: Block complexity ($\text{BLCK}_{\text{COMPLEXITY}}$)

*** MQ2 - What is the requirement density per system?**

Context: Block elements are associated to one or more functional requirements. Changes in systems associated to many requirements may affect a high number of components.

Metric involved: Requirement density per system ($\text{BLCK}_{\text{REQT\_DENSITY}}$)

*** MQ3 - What is the requirement's total assigned difficulty?**

Context: Requirement are assigned a *difficulty level* value. A high difficulty may represent a lot work is necessary to fulfill this requirement, or it may indicate that further decomposition is required.

Metric involved: Requirement difficulty ($\text{REQT}_{\text{DIFFICULTY}}$)

*** MQ4 - What is the system fragmentation per requirement?**

Context: Each requirement is satisfied by one system. This system may be further decomposed into multiple systems (in the next level). A high system fragmentation may indicate that changes in the requirement could propagate through many systems.

Metric involved: Block fragmentation per requirement ($\text{REQT}_{\text{FRAGMENTATION}}$)

7.2.4 Metrics

A set of metrics follows, designed to provide quantitative answers to the previous questions. Note that the first two measurements are meant for block elements, while the last two are meant for requirement elements.

*** Block complexity ($\text{BLCK}_{\text{COMPLEXITY}}$):** Represents the total 'complexity' value of a block element. It is calculated by first identifying all the block's 'leaves' in the structure. Next, for every block, the complexity value is read and weighted against a value assigned by the user.

Interpretation: A high value indicates that engineers expect a system have a high complexity, and thus it may require a high amount of work to complete, or (if applicable) further decomposition is required.

*** Requirement density per system ($\text{BLCK}_{\text{REQT\_DENSITY}}$):** Represents the number of requirements assigned to a system. It is calculated by identifying the number of requirements elements associated to a block with an outgoing *«satisfy»* connector.

Interpretation: A high value indicates a large number of requirements associated to a single system. Changes on this system may potentially affect many requirements.

*** Requirement difficulty ($\text{REQT}_{\text{DIFFICULTY}}$):** Represents the total 'difficulty' value of a requirement element. It is calculated by first identifying all the requirement's 'leaves' in the structure. Next, for every requirement, the difficulty value is read and weighted against a value assigned by the user.

Interpretation: A high value represents that engineers expect a high amount of work to fulfill this requirement, or (if applicable) further decomposition is required.

*** Block fragmentation per requirement ($\text{REQT}_{\text{FRAGMENTATION}}$):** Represents the number of systems associated to a requirement. It is calculated by identifying all the associated block elements with an incoming *«satisfy»* connector. From these, all the leaf blocks are identified (duplicate entries are removed).

Interpretation: A higher value represents a higher system fragmentation, and thus high values may denote that a change in a requirement may affect a high number of systems.

7.3 Generated Output Data

Finally, this section is meant to provide answer to research sub-question **RSQ.2c -** *What output data do we need to generate?* There are three key elements we wish to visualize:

- *Element identification*: The ability to identify each element, through a label or identifier.
- *Project-meta model structure*: The data should reflect the structure of the model, and the relation between the elements. In the case of the WST, there are two tree structures.
- *Metric values*: Each element (block and requirement) has two metric values associated.

Every element hierarchy analyzed is exported to a separate file (a distinction is made between the block- and requirement-hierarchy). Since the framework relies on third party tools, the format of the file depends on the input accepted by the tool.

Chapter 8

Information Interpretation

The previous chapter examined the second phase of the proposed implementation approach: knowledge inference. The first goal was to identify and reconstruct the project meta-model structure. However, to present the user with the appropriate data, it was chosen to apply the goal-oriented GQM methodology. After stating the goal, and deriving a set of related questions, and defining the corresponding metrics, the generated data was exported to a collection of output files for visualization with a third-party tool.

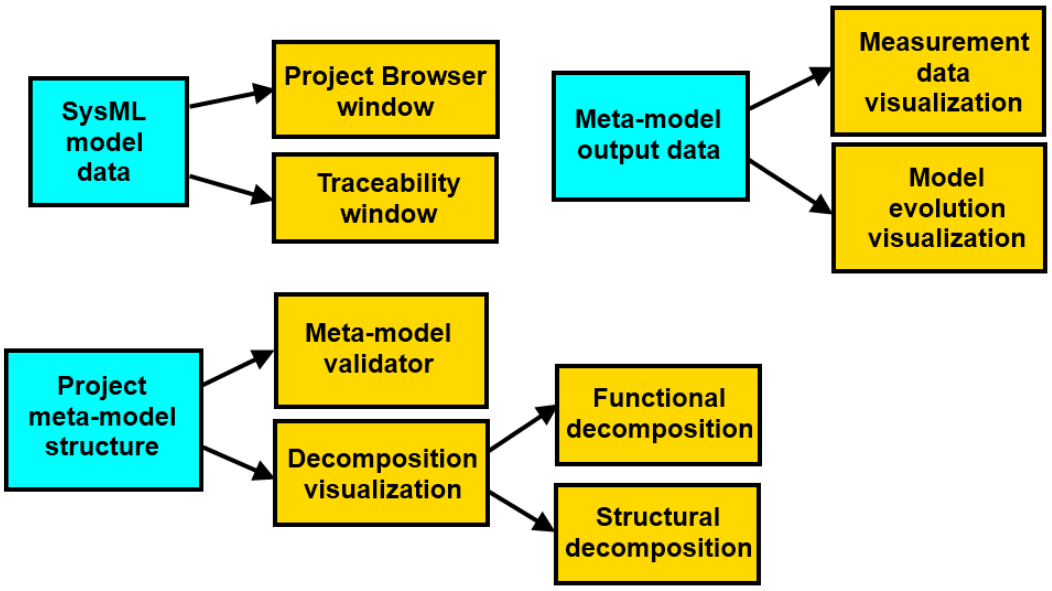


Figure 8.1: Overview of the *information interpretation* phase.

This chapter discusses the third and last phase of the implementation approach: information interpretation (see figure 8.1 for an overview). Section 8.1 describes how the SysML model data was visualized within the VoSMA tool to provide functionality resembling the EA visual platform. Section 8.2 examines how this technique was further

expanded to present the appropriate project meta-model architecture, visualizing the corresponding hierarchical decomposition structures, and providing the tool with a model checker functionality. However, presenting the data using these layout is limited, thus in section 8.3 multiple visualization methodologies are presented relying on third party tools, by combining the tree-map and heat-map displays. Lastly, section 8.4 explores and discusses various alternatives to provide model evolution visualization in a future project.

8.1 SysML Model Data Visualization

The next three sections will attempt to provide answer to research sub-question **RSQ.3a** - *How can we efficiently present the generated data to systems engineers?* Tree structures are used in VoSMA to reconstruct, analyze, and visualize the hierarchical nature of the SysML model data. To this end, the tool implements the Java JTree[41] class. These structures are used to recreate two windows familiar to EA users: *Project Browser* and *Traceability*. These two windows can be used with any SysML model.

8.1.1 *Project Browser* window

In Enterprise Architect, the Project Browser window (discussed in section 5.1) allows users to navigate through a SysML project model, displaying packages, diagram, elements and element properties. As shown in figure 8.2, VoSMA implements such a functionality, which resembles the EA variant.

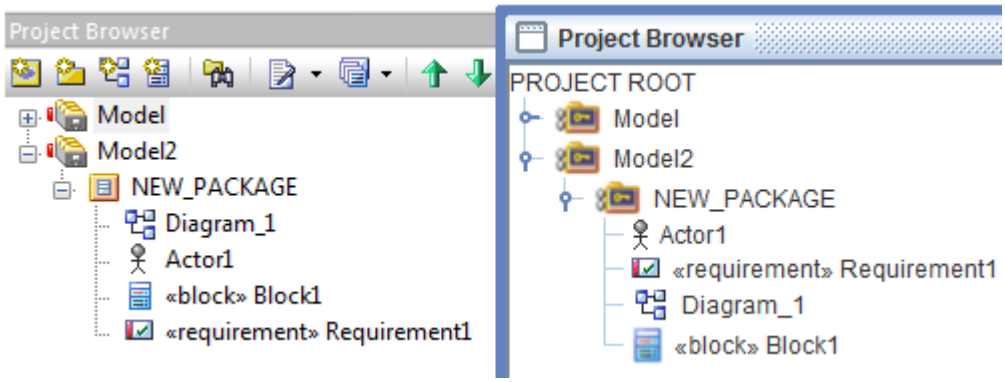


Figure 8.2: *Project Browser* window. On the left, the EA variant, on the right the VoSMA counterpart.

8.1.2 *Traceability* window

As discussed in section 5.1, the Traceability window shows the composition of the current element with respect to other model elements. This information is derived from

[41] http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/JTree.html

relationships with child or related classes. Figure 8.2 shows how the VoSMA implements similar functionality, mainly used to test the correctness of the model data read, and the association between elements.

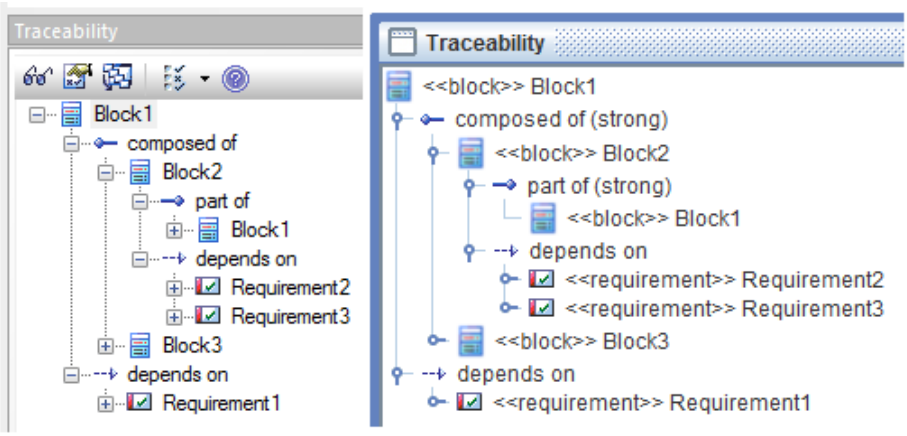


Figure 8.3: *Traceability* window. On the left, the EA variant, on the right the VoSMA counterpart.

8.2 Project Meta-Model Architecture Visualization

In chapter 7 the relevant model structure was extracted and reconstructed. As explained, this structure can be dived into two different element hierarchies, and thus it was chosen to use two different tree structures. However, there are two ways to populate these structures:

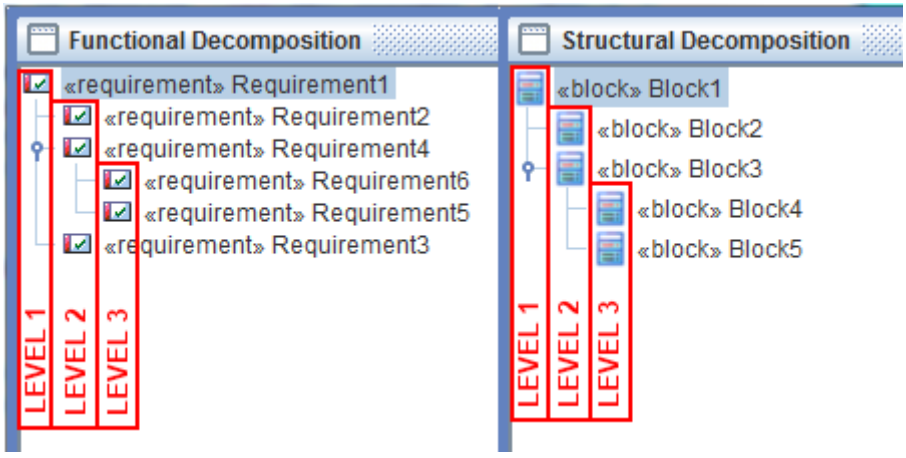


Figure 8.4: Decomposition visualization in VoSMA using the "plain" layout.

- *"Plain" visualization*: This layout shows both hierarchies next to each other (see figure 8.4). Each hierarchy is a tree structure with the top element at the root. This

element contains elements from level 2 as its children, and so on. This layout is meant to provide the user with a clear view of each decomposition, but no relations with the other hierarchy is shown.

- *"Rich" visualization*: As before, this layout also shows both hierarchies next to each other (see figure 8.5). However, this time every element is also connected to the associated elements from the other hierarchy, which provides some sense of "connectivity" between both structures. However, this results into a more cluttered display.

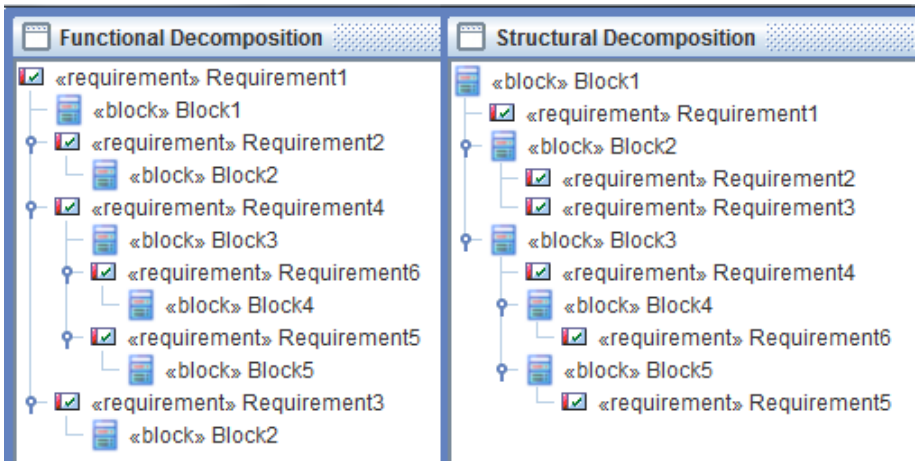


Figure 8.5: Decomposition visualization in VoSMA using the "rich" layout.

8.2.1 *Project Meta-Model Validator* window

As explained before, the project meta-model structure is not enforced by the modeling tool. Engineers expressed their wish to be able to check if this structure was consistent with the rules defined. The *Project Meta-model Validator* window is designed to show errors present in this structure sorted by error type (see figure 8.6). Another variant of this functionality based on this principle checks all the elements in the project according to these rules. The objective, for example, is to find elements which should have been assigned to the structure, but have been left out.

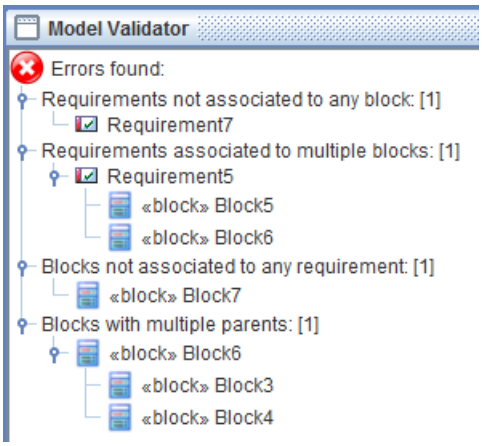


Figure 8.6: Example of the *Project Meta-Model Validator* window. Errors are sorted by type.

8.3 Measurement Data Visualization

Visualization methodologies may present information through the modification of a finite set of element properties [53]:

1. *x*, *y*, and *z* position of elements.
2. *width*, *height*, and *depth* of elements.
3. *color*, *shape*, and *texture* of elements,
4. *labels* of elements.

As described in section 7.3, this project aims to present a minimal set of data to the user. A 2-D solution will suffice, where the two metric values may be presented trough the elements size and box color. Labels are used to identify elements, and element position represents the element's place within the model's hierarchical structure. The visualization of tree structures is further explored in the following sections.

8.3.1 Tree Visualization

The previous sections discussed the hierarchical nature of SysML model data, and how tree structures can be employed to analyze and present project-specific model information. Tree visualization (or hierarchy visualization) can be defined as: *"a branch of information visualization dedicated to the graphical representation of connected, acyclic graphs – trees"* [58]. Most techniques require *rooted trees*, consisting of a *root node* (designated top element), *internal nodes*, and *leaves* (bottom elements). There are multiple tree visualization techniques [27] [70], (see figure 8.7 for an example). Visit treevis.net[42] for an extensive overview of different tree visualization layouts.

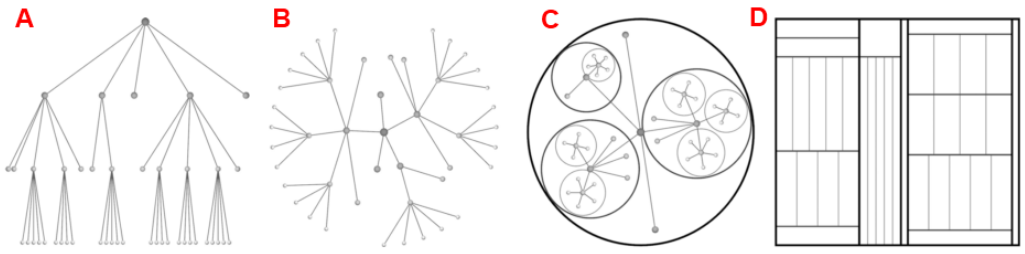


Figure 8.7: Common tree visualization techniques: a) rooted tree, b) radial tree, c) balloon tree, and d) tree map layout. As originally appeared in [27].

[42] http://www.treevis.net

Tree-maps Tree maps can be defined as: *"a representation designed for human visualization of complex traditional tree structures: arbitrary trees are shown with a 2-d space-filling representation"* [62]. Thus, this enclosure method layout representation is used to display (tree-structured) hierarchical data nodes as a set of nested rectangles [11]. In a tree, each branch is represented as a rectangle, which is further partitioned in smaller rectangles representing sub-branches. Leaf node rectangles' size are proportional to a specified dimension on the data. There are multiple tree map visualization variants [63], but figure 8.8 shows the four main types.

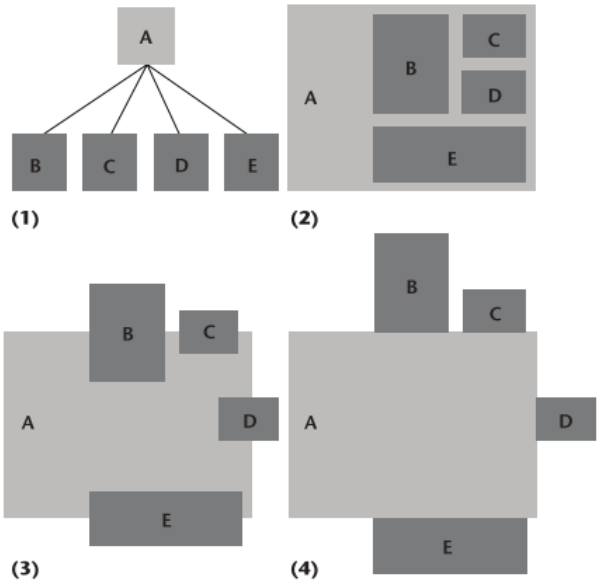


Figure 8.8: Edge representation types in tree maps: (1) explicit, node- link, (2) implicit, inclusion, (3) implicit, overlap, and (4) implicit, adjacency. As originally appeared in [58].

The implicit-inclusion technique is ideal for displaying large trees (because its space-filling layout technique), and viewers can easily compare values (node space is proportional to value size) [62]. One of its main disadvantages however, is the difficulty for viewers to identify the hierarchical relationship between nodes (because its enclosure to display the structure [27]), and the different levels.

8.3.2 Heat-map

In this context the term 'heat-map' is used to refer to a *choropleth map*, rather than the 2D display of values in a data matrix, or representation of cluster data [77]. Coropleth maps rely on a system of color coding patterned in proportion to the measurement of the variables being displayed on the map, and are often used to represent how values vary between areas [10] [3]. Figure 8.9 shows an example displaying the world's population density.

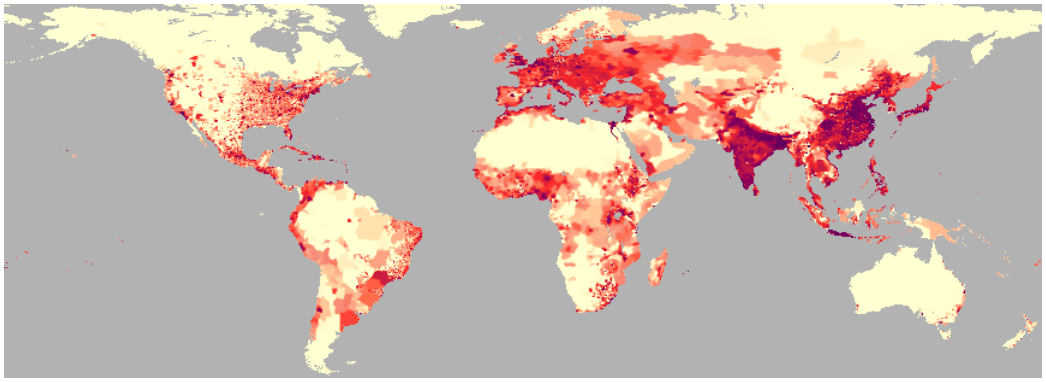
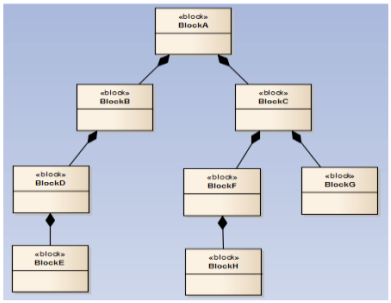


Figure 8.9: Example of a choropleth map, showing the world population density, from Gridded Population of the World[43]. A darker shading represents a higher population density in the area.

8.3.3 Visualization with third-party tools

The metric and structural data generated in the previous chapter can be visualized employing a third party tool or software library. Some of the free services include Treemap-gviz[44], Google's visualization API[45], and Microsoft's Treemapper[46]. As proof-of-concept, two of these tools are evaluated: Treeviz[47] and MS Treemapper. The next sections will showcase both tools using the structure displayed in figure 8.10. Note that all the elements in the following examples have been assigned the same metric values, which result in all (leaf) boxes having the same color and size. However, other layouts proved useful to display the project's structure (such as the Sunburst Tree layout, refer to appendix I).



Node	Children
BlockA	BlockB, BlockC
BlockB	BlockD
BlockC	BlockF, BlockG
BlockD	BlockE
BlockE	BlockH
BlockF, BlockG, BlockH	-

Figure 8.10: Model structure as a tree. On the left, the element structure in SysML. On the right, its tree representation, where elements have been modeled as nodes and connectors as edges.

[43] http://sedac.ciesin.columbia.edu/gpw-v1/globldem.doc.html

[44] http://code.google.com/p/treemap-gviz/ (accessed 01-12-2012)

[45] https://developers.google.com/chart/interactive/docs/gallery/treemap/ (accessed 01-12-2012)

[46] http://research.microsoft.com/en-us/downloads/3f3ed95e-26d8-4616-a06c-b609df29756f/ (accessed 01-12-2012)

[47] http://www.randelshofer.ch/treeviz/

Treeviz project: Treeviz project aims to achieve fast and interactive visualization of large tree data structures. A demo application (version 1.1) was used to visualize the output data as proof-of-concept. The application is capable of visualizing the data with five types of tree layouts (Hyperbolic, Sunburst, Icicle, Sunray, and Iceray), and two types of tree-map layouts (Circular and Rectangular). Every component may be assigned a label, additional data (displayed on mouse-over), and two metric values (box color and size). After reviewing the various layouts, we came to the conclusion that tree-map variants are the best choice when dealing with large projects with a large number of elements. Especially the rectangular layout which uses the whole screen.

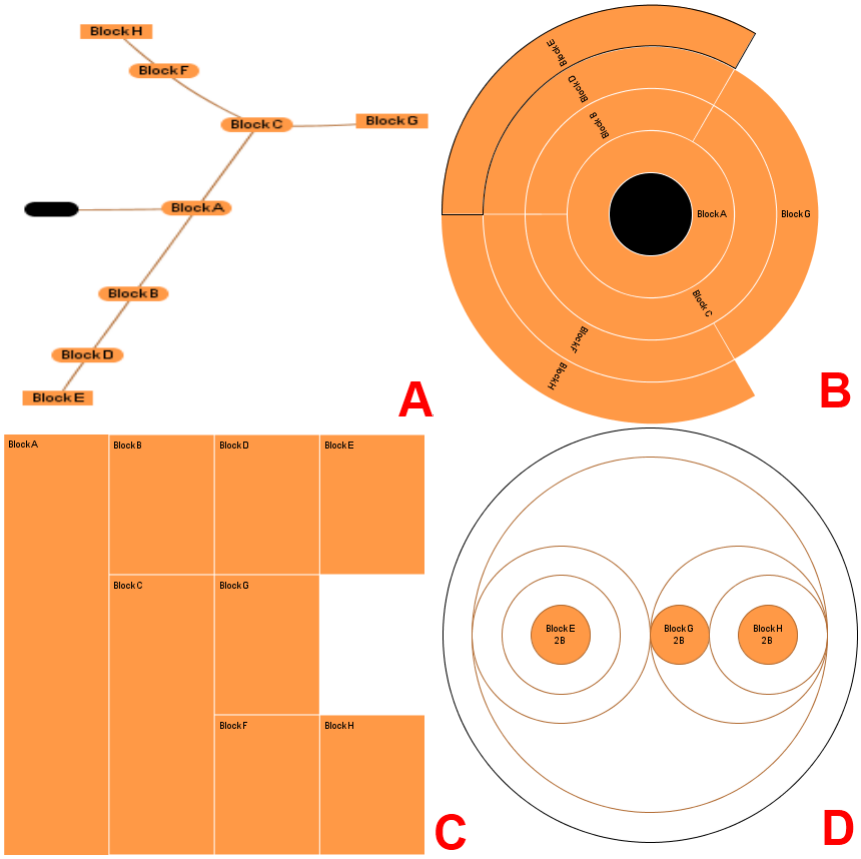


Figure 8.11: Treeviz visualization example. The tree structure from figure 8.10 is presented using four layout types: a) Hyperbolic Tree, b) Sunburst Tree, c) Icicle Tree, and d) Circular Treemap.

Microsoft Research Treemapper: Microsoft Research Treemapper allows (rectangular) tree-map visualization of hierarchical data. For this project, the standalone application (version 1.0.1.34) was chosen, but the tool is also available as a MS Excel add-in. Elements are represented as rectangular boxes and may be assigned labels with their identifier. Two metric values can be represented, by adjusting the box's size and color.

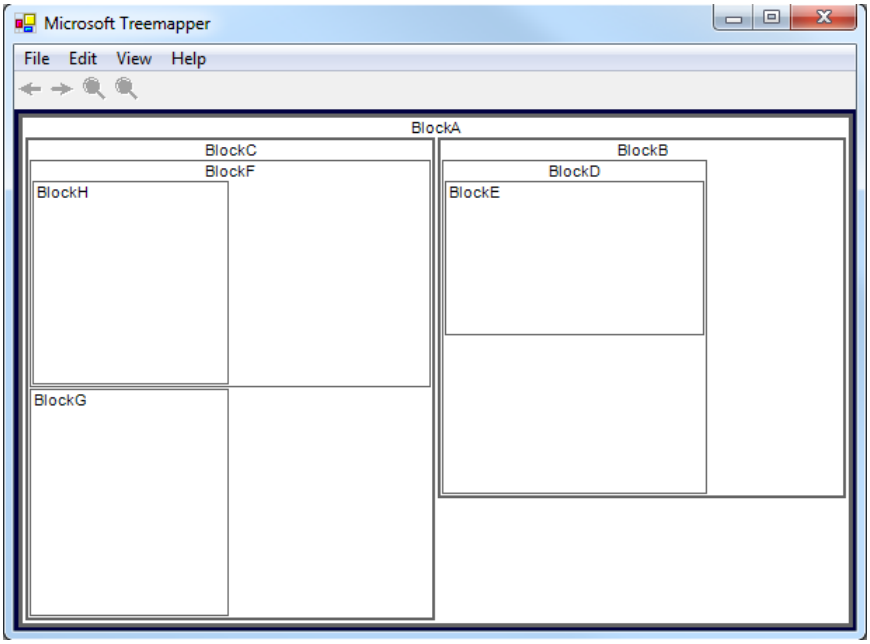


Figure 8.12: MS Treemapper visualization example. The tool uses the structure from 8.10 as input.

With respect to rectangular tree-maps, there are a couple of differences between the Treeviz and Tremapper tools (see figures 8.12 and 8.13). There are two layouts in Treeviz: 'Show full depth', where all leaf elements are shown, and 'Show current depth only', where clicking on a element will show its direct children (but not its descendants). MS Treemapper displays all elements on screen, nested within their corresponding parent's box. Non-leaf elements can be assigned a weight, represented by the extra space in a box (excluding boxes representing its children). Treemapper also provides the ability to zoom-in on elements. Another difference is the information shown on mouse-over: Treeviz can display detailed information, while Treemapper only shows the element's ID or path.

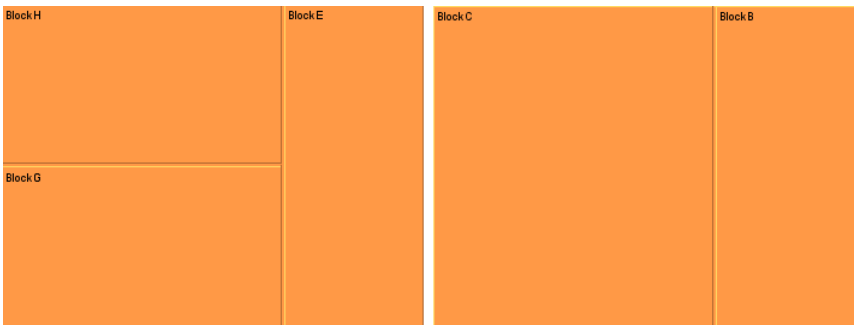


Figure 8.13: Rectangular tree-map in Treeviz. The tool uses the structure from 8.10 as input. On the left, the 'Show full depth' is selected. On the right, the layout after selecting 'Show current depth only' and clicking on *BlockA*.

8.3.4 Other visualization techniques

It should be kept in mind that the techniques presented above are intended to visualize hierarchical data as tree-structures. However, it is expected that not all project meta-models can rely on these structures, and thus a couple of alternatives are explored.

Class Blueprint: This technique aims to *"provide a better understanding of a class"* [34] by visualizing its internal structure. Classes are mapped to a *template* class blueprint, consisting of five layers: initialization, interface, implementation, accessor, and attributes (see figure 8.14 for an example).

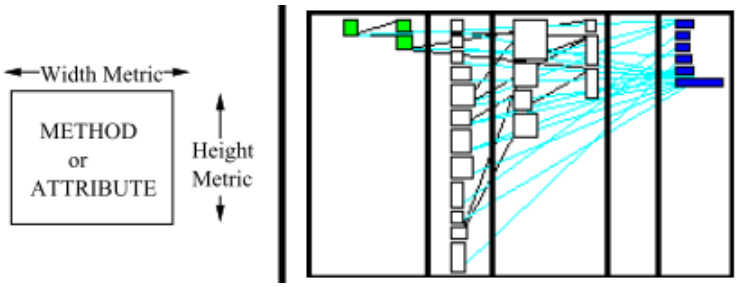


Figure 8.14: Visualization with Class Blueprint, as it originally appeared in [34]. On the left, how up to two metrics can be visualized. Note that color is used to identify element type, rather than display a metric value. On the right, a blueprint visualization of a class.

Polymetric Views: This technique aims to *"understand the structure and detect problems of a software system in the initial phases of a reverse engineering process"* [35]. It relies on two-dimensional displays to visualize object-oriented software, where nodes represent (abstraction) entities, and edges the relationships between them. This is coupled with metric information (see figure 8.15 for an example).

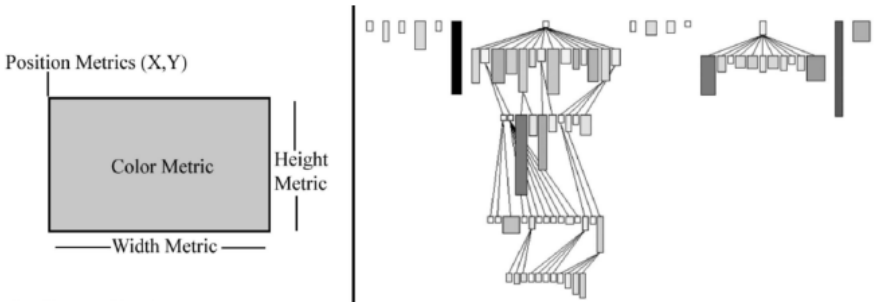


Figure 8.15: Visualization with Polymetric Views, as it originally appeared in [35]. On the left, how up to five metrics can be visualized. On the right, visualization as a tree structure, with edges as relationships.

8.4 Meta-Model Evolution Visualization

This section provides answer to research sub-question **RSQ.3b** - *How can we represent the project meta-model architecture evolution through development?* Visualization of the evolution of project meta-model structures is one of the key goals of this project. However, time constrains prevented the implementation of a final solution at this point. Instead, a short study is performed to present some possible future pointers. The desired goal is to map the structural evolution through the project's development, and to identify possible hot-spots, which, for example, may benefit from early refactoring. Other application is measuring how much the structure changes when an element is modified. Few changes through the model may indicate a robust structure. There are two main approaches [22] to model visualization evolution:

1. *Version-centered approaches*: Concerned with answering of when something happened in the history, using version as a representation granularity. This often involves representing time on an axis and placing different versions along it to show where changes occurred.

2. *History-centered approaches*: Concerned with the question of what and where something happened (rather than when), using an ordered set of versions as representation granularity.

Be aware that in this project there are two kinds of versions: model version (as labeled by the repository) and element version (which represents the how many times an element has been modified). Since it is easier to visualize how the model or particular elements evolve, rather than the evolution of changes and their meaning, the following version-centered examples (from research into software code evolution) are presented:

Two versions comparison: This variant forms the base of any evolution analysis. Two versions of a model or entity are measured, where the goal is to find different types of changes between them.

Evolution Matrix: This methodology [33] compares the evolution of multiple entities of properties (see figure 8.16 for an example). In our case, each box may represent a version of a SysML element (a block or requirement for example), and each line holds multiple versions of that element. Size and color of the box represent measurement or properties of that element. From the matrix different patterns can be detected (growth, shrinking, stabilization...).

Evolution as Cities: Another interesting possibility is visualizing models as cities [76]. In this 3D layout entities can be represented as buildings located in (nested) districts, which may represent levels or packages, for example. Metric data can be expressed adapting dimensions, position, color, color saturation, and transparency of entities. Figure 8.17 shows an example of this concept using the *CodeCity* tool.

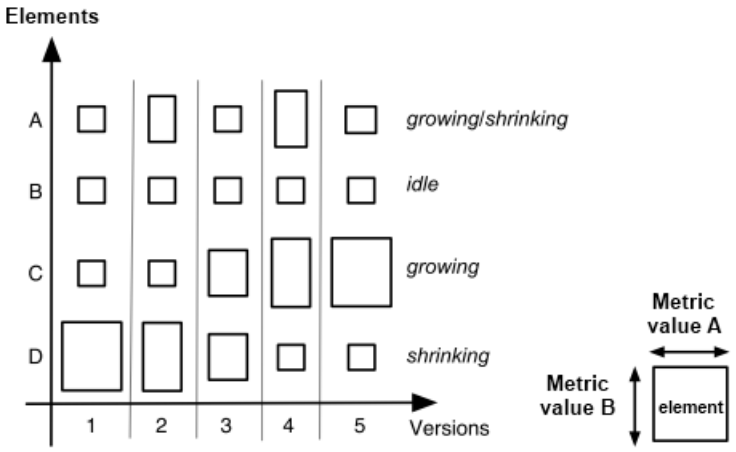


Figure 8.16: Evolution Matrix concept, adapted from [22]. The horizontal position is given by the element's revision number, and the vertical position is given by the element's name. The size is determined by a combination of two measurements. A third measurement could be visualized by assigning colors to the blocks.

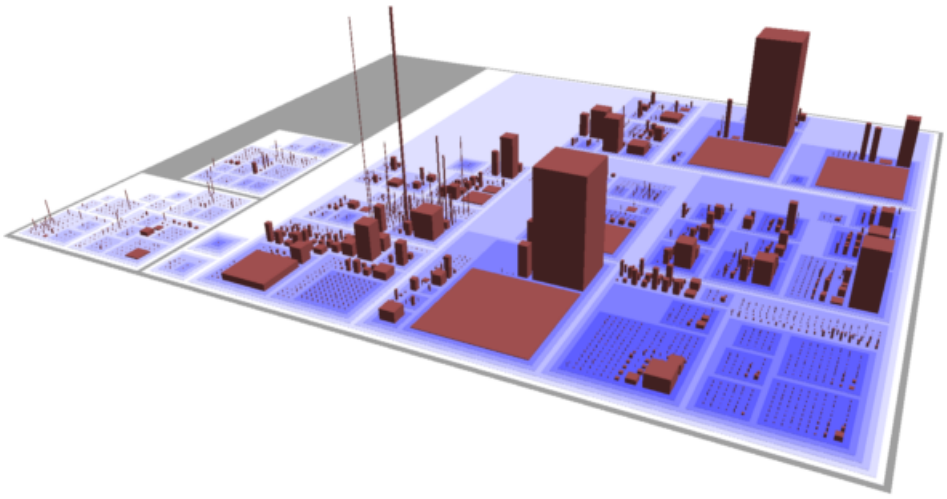


Figure 8.17: Snapshot from the *CodeCity* tool, as it originally appeared in [76].

Chapter 9

Assessment of Correctness and Usefulness

Previous chapters discussed the implementation approach followed to answer the main research questions, where VoSMA was developed to aid with this task. This chapter provides answer to the last research sub-question: **RSQ.3c** - *How satisfied are systems engineers with the presented data?*, by evaluating the obtained results. In section 9.1 the correctness and completeness of the data read is assessed. In section 9.2 three tunnel projects in SysML are analyzed and evaluated as test cases, while the usefulness is assessed through interviews with the potential users in section 9.3. Finally, possible threats to validity are examined in section 9.4.

9.1 Completeness and Correctness Assessment

A series of test cases were created in EA to assess the completeness and correctness of the SysML model data read by VoSMA (see appendix F for a couple of examples). These test cases were mainly designed to check that all the relevant element data was correctly read, and that the corresponding package structures could be recreated. A second series of test cases was implemented to check element relationships and different structures (such as block decomposition). The third series was concerned with reading project meta-model structures such as those as described in section 7.1. After these simple tests, it was decided to use the tool, to analyze a basic (yet complete) SysML project: the example [48] discussed in "A practical guide to SysML" [21], the book used as SysML reference at Soltegro. The project is a good starting point when learning SysML, simple enough to not overwhelm the user, yet complete, since it features most of the language's capabilities. This project was mainly used to test the ability to navigate a project, provide element traceability, and reconstruct the various element hierarchies. However, it is necessary to assess the system's performance with real life scenarios, which contain a higher number of packages, diagrams

[48] Available from `http://www.elsevierdirect.com/v2/companion.jsp?ISBN=9780123852069`, accessed 01-11-2012

and elements. Three tunnel projects in SysML were chosen to this end: Nijverdal, Westerschelde, and A4 Delft-Schiedam.

9.2 Test Cases

VoSMA was tested using the latest versions of the following projects in SysML as test cases:

9.2.1 Test case: Combiplan Nijverdal (NVD) project

Rijkswaterstaat[49] (RWS), part of the Dutch Ministry of Infrastructure and Environment, and responsible for the execution of the public works and water management, partnered with ProRail[50], a government organization tasked with care of maintenance and extensions of the national railway network infrastructure, to execute the Combieplan Nijverdal project[51], scheduled to be completed in 2014. Currently, a busy and often congested state road (the N35) runs through the middle of the city of Nijverdal. To alleviate this situation, the road will be relocated to a new 6 kilometer trajectory to the north, together with an adjacent railroad.

Figure 9.1: Section of the Nijverdal Combitunnel, showing the railroad (left), and two-lane road (right)

A part of the new road and railroad will come to lie in a tunnel: the 1.5 kilometer long *Salland Twente* (combi)tunnel. The main objective of this project is to enhance the quality of life and safety in the city by reducing traffic congestion, pollution and noise. Soltegro relied on the MBSE methodology to design the tunnel using SysML, while integrating the RWS' Landelijk Tunnelstandard using models.

Evaluation The NVD SysML project was proposed as test case for the VoSMA software tool. However, this project has a modified version of the meta-model described in section 7.1. Instead, the NVD project is based on the "Functional Design" approach, leading to a

[49] http://www.rws.nl/wegen/plannen_en_projecten/n_wegen/n35/combiplannijverdal/
[50] http://www.prorail.nl/Publiek/Infraprojecten/Overijssel/CombiplanNijverdal/
[51] http://www.tunnelplan.nl/

functional analysis which produces the functional decomposition (do not confuse with structural decomposition in SysML). Before analyzing this project with VoSMA, the NVD project meta-model should be defined, the appropriate measurements be discussed and developed. Even so, all the elements in the project can be correctly read using VoSMA, and after the proper meta-model is defined, it is expected to easily visualize the model's architecture. Still, given the nature of this project (many interfaces between elements), tree structures may not be appropriate to visualize the structure.

9.2.2 Test case: Western Scheldt Tunnel (WST) Complex project

The Westerschelde[52] (Western Scheldt) tunnel has a length of 6.6 kilometers, the longest tunnel for highway traffic in the Netherlands. Its location under the Western Scheldt estuary between Ellewoutsdijk and Terneuzen makes it a key connection point between Zeeland, Flanders and West-Brabant. Opened on 14 March 2003, it consists of two excavated tubes, each with room for two driving lanes (see figure 9.2).

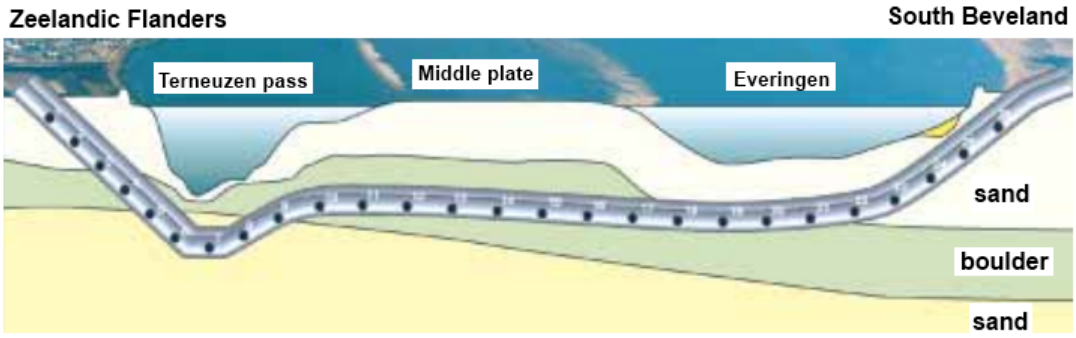


Figure 9.2: Section of the Westerschelde tunnel, adapted from [43].

The tunnel is part of the *Westerscheldetunnelcomplex* (Western Scheldt Tunnel Complex), consisting of access roads in South Beveland and Zeeland, the toll points, the tunnel and associated (tunnel) buildings. The current maintenance agreement for the *Westerscheldetunnelcomplex* ends in 2013. Because of this, a new contract is necessary (from 2013 to 2033, afterwards the complex will be transferred to the State). The new contract deals with the maintenance, refurbishment and other adaptation works, and following European directives. To take part in this process, Soltegro has modeled their own MBSE solution based on SysML models in Enterprise Architect.

Evaluation The WST SysML project served as the main test case (refer to appendix I for an extensive evaluation using VoSMA). Both element hierarchies were identified, reconstructed, analyzed and measured. Multiple visualization layouts were used to present this information.

[52] `http://www.westerscheldetunnel.nl/`, in Dutch

9.2.3 Test case: A4 Delft-Schiedam (A4DS) project

Currently the A13 highway is the main connection between the cities of The Hague and Rotterdam. However, the increasing traffic flow on this road leads to frequent traffic jams which increase air pollution, noise, and traffic hazards in the surrounding areas. RWS began in 2012 extending the A4 [53] highway to connect the cities of Schiedam and Delft. The project is also known as *A4 Midden Delfland*[54]. Part of this project is a 2.0 kilometer land tunnel (Tunnel Schiedam), to be completed in 2015. Soltegro is currently developing a solution based on MBSE for this project.

Figure 9.3: Artist's impression of the A4DS project. Orange represents the half-sunken location, red the sunken location, and green the land tunnel.

Evaluation As with the NVD project, the whole SysML model data can be read correctly. However, the A4DS project is currently in the early stages of development. Parts of this projects resemble the methodology examined in 7.1 (the land tunnel also integrates the LTS), and thus some element's structural decompositions can be identified. Still, the Model Validator functionality may be used to identify requirements in the model which have not been assigned to a block, for example. No project-specific measurements were developed for this project.

9.3 Usefulness Assessment

Short feedback sessions were conducted at Soltegro with engineers experienced with the EA platform and SysML, following a presentation given at a company meeting on 27-11-2012, which provided a general overview of the research project. These sessions seek to achieve three main goals: a) to evaluate the project according to user experience and expectations, b) to raise awareness about this project within Soltegro by providing engineers with the opportunity to know more about the software and its capabilities, and c) to generate pointers and suggestions for future work. See appendix G for an overview of the interviews conducted. Note that the software was not evaluated as a final product, but rather as a showcase for possible functionality.

The main outcomes from these sessions were as follows: users mainly appreciate the extra functionality provided by VoSMA, which cannot be found in EA at this point, specially

[53] `http://www.rws.nl/wegen/plannen_en_projecten/a_wegen/A4/delft_schiedam/`, in Dutch
[54] `http://www.a4middendelfland.nl/`, inDutch

to analyze project meta-model structures (which are hard to track in EA). The architecture visualization was greatly appreciated, as it helped to quickly identify points of concern in the model. Suggestions included using other layouts to perform change impact analysis, which can help other members of the team (or less technical users) easily picture how changes in a system or requirement may affect the project. In short, users were very optimistic about the project, providing multiple ideas for future functionality.

9.4 Threats to Validity

This section discusses some of the possible factors which could affect the results presented through this study. These mainly involve: the correctness of the data read, the changing working practices at Soltegro, the chosen visual structures and visualization layouts, and the lack of a final product.

SysML model data is correctly read as far as it could be assessed with the test cases. However not enough cases were made to test that information from every element in the model was read correctly. Instead it was chosen to focus on the elements and data relevant to this research project.

It should be noted that the situation described in section 5.1 has started to change at Soltegro. Currently, a local EAP file allows users to edit a model without being connected via a shared model repository. This file periodically updates their copy from a shared repository. However, the new situation involves a dedicated database management system to host the model. This is the recommended approach for larger teams, as all team members can view and edit the current-state model, without the need for a separate synchronization. Thus, the version control system is not a mechanism for distributing model information, but rather to help manage revisions. It should be noted that XMI files are still used in the repository (but not in the 'local working copy'). Even if XMI files were not used in the future (which is not expected), VoSMA could still be easily adapted to query information from the EAP file.

In this study it was chosen to use tree structures to reconstruct the project meta-model structure. Since the SysML model data is hierarchical in nature, it was easy to adapt this to fit the needs of this research. However, its limitations have become apparent. Two tree structures are necessary (one for each decomposition), were it may become cumbersome to identify the relation between elements from each tree. Although the structural decomposition can be represented perfectly as a tree, as each block can only have one parent, this is not the case for the functional decomposition, where an element may have multiple parents. These elements do not count multiple times towards the measurements developed in section 7.2, but are represented as a separate entity in most layouts (thus the same element can appear multiple times). To prevent this, some alternatives were discussed in section 8.3.4. Since there are multiple project meta-models, the question remains if the presented visualization implementations do fit other projects. Depending on their architecture, or the wishes of engineers, other layouts or visual structures could prove more efficient.

Given the nature of this study, the tool has yet to be employed in a real life working environment. Because of this, multiple crucial questions (beyond this study) remain unanswered: *does it perform the right measurements?*, *can it improve the model's development?*, *can it help with change impact analysis?*...Feedback sessions with experienced EA and SysML users did show promising results, but actual application during model development would be required before addressing its added value in a project.

Chapter 10

Discussion

This chapter summarizes the findings through this thesis study (section 10.1), while revisiting the main research questions and the obtained results. Section 10.2 discusses some of the main strengths and limitations of the chosen approach.

10.1 Summary of Findings

The problem was introduced in chapter 1, where the main research questions were posed. However, context information was necessary before these could be addressed. Basic concepts relating systems were introduced in chapter 2, and a general overview of the Systems Engineering field was provided in chapter 3. After discussing the current transition from the traditional document based approach to MBSE, a general overview of the Systems Modeling language was provided in chapter 4. An evaluation the current situation followed in chapter 5, concluding that a new framework was necessary. To this end the VoSMA software tool (designed following the *extract-abstract-present* paradigm) was implemented to answer the main research questions:

R.Q.1 - How can we gain insight into project-specific SysML meta-model architectures?

The first research question resolves around specific project meta-model identification, analysis and measurement. However, the first step was to retrieve the necessary SysML model data. Chapter 6 described what data sources were chosen, what data was read, how the model is stored, and how element traceability is provided (essential to transverse the model). The first research question can now be answered in chapter 7: the mapping of components and requirements is achieved by recreating the relevant project meta-model architecture form the model data. The GQM approach was applied to set a measurement goal, define a set of questions, and finally create the corresponding metrics to perform the appropriate measurements.

R.Q.2 - How can we effectively visualize the meta-model structure, its measurement data, and picture its evolution through development?

The second research question is answered in chapter 8, which explored multiple alternatives to present the measurement and structural data to the user. The chosen solution combines the heat-map and tree-map layouts using third-party tools. Though not implemented, the chapter also discussed possible methodologies to provide model evolution visualization.

Furthermore, chapter 9 presented an evaluation of the implementation based on test cases and feedback sessions with potential users. The VoSMA tool allowed to analyze and visualize the latest revision of the WST project (see appendix I). Other projects were not analyzed because the corresponding meta-model had to be defined (NVD), or the project was in the early stages of development (A4DS). Feedback sessions showed a lot of interest by potential users in this project, and many possible future directions for this project were proposed (see appendix G).

10.2 Approach Implementation Discussion

This section provides a short reflection and discussion of the implemented solution, divided into three categories: configuration effort, usability, and scalability.

Configuration effort: The configuration effort of VoSMA is low, as the software does not need to be installed (but requires a JVM), and no further configuration is necessary (beside selecting the correct meta-model structure type to be analyzed). The software can be adapted to be platform-independent (see section 11.2). Still, the current implemented solution means a user needs to use three tools: visual modeling tool to work on SysML models (EA), a tool to analyze and measure project-specific models (VoSMA) and a tool to visualize this data (Treeviz/MS Treemapper/other). These third-party tools may require additional configuration and/or installation.

Usability: One important issue to note is that VoSMA was intended as a proof-of-concept, meant to show the possibilities of this project and to provide a solid framework for future work, rather than to deliver a final product. Thus, the tool prioritizes functionality above usability at this point. The current user interface was made using Java Swing[55] to somewhat resemble the *look and feel* of the Enterprise Architect platform, a tool engineers are familiar with. It also implements visualization and functionality used by EA to navigate the model and trace elements. Still, using the tool means an additional step for the engineer, which has to model using EA, and analyze it with VoSMA. The current implementation of the visualization means having to rely a third tool to visualize this data, which means yet another step in the process. The first step can be removed if VoSMA (or

[55] `http://docs.oracle.com/javase/tutorial/uiswing/start/about.html`

its functionality) is integrated as an EA plug-in. The second step can be avoided by integrating the visualization into VoSMA (discussed in section 11.2).

Expected scalability: The latest version of the tool (20-12-12) is offered as an executable JAR file of around 3,19 MB (where 3,12 MB necessary for the SVNKit library). Loading a project is almost as fast as opening an EAP file in EA. In terms of memory, the tool requires around 25 MB to run, which increases depending on the number of elements read in a model. In the case of the latest version (9571) of the WST project with 130.639 elements this results in a considerable memory usage of around 290 MB. However this could be alleviated in by only storing relevant element data. Thus, reconstructing the model from the root (leaving out elements not present in the current model) and removing internal EA data fields.

Should the structure ever become too large to handle, using an external database could provide an easy and fast replacement, trading memory for disk space (similar to the solution implemented in EA with the EAP files). However, it is not expected to become a problem, since the three test cases discussed in the previous chapter (up to 250K elements) were easily read by the software.

In terms of visualization, the current solutions may be insufficient in a project with a structure consisting of a very large number of elements. In this case, blocks (elements) shown by Treemapper and Treeviz will become very small, and the labels will be unreadable. Also, even though it is not the practice, if a project consists of a very large number of levels, it may be very hard to identify them with these tools. This can be alleviated by providing functionality to zoom in parts of the layout, or visualizing per level. Since not all the projects may rely on (or be represented by) a tree structure, it is recommended to look at layout techniques such as BlueClass [34].

Chapter 11

Contributions and Future Work

This last chapter concludes this thesis work. Section 11.1 lists the contributions made by this study, while some recommendations and possible future work directions are presented in section 11.2.

11.1 Contributions

The contributions made by this thesis project are as follows:

- *General overview of Systems Engineering and the Systems Modeling Language*: Chapters 2 through 4 of this thesis consist of a short literature study intended to provide context to this project.
- *The VoSMA software tool*: The tool showcases the possibilities of this project, and is meant to serve as a framework and foundation for future work. It features some functionality found in EA, and is capable of identifying measuring, and visualize specific SysML project meta-model structures.
- *SysML project-specific measurements*: Beyond the currently available standard UML and SysML metrics, some metrics were developed suited to measure the project meta-model used at Soltegro using the GQM approach.
- *SysML project-specific visualization*: Multiple visualization layouts were implemented to present both structural and measurement data to the users.
- *Three case studies and multiple feedback sessions*: By performing three test case studies and conducting multiple interviews with experienced EA users, we demonstrated that the system may be capable of providing insight into the structure and quality of project-specific models.

11.2 Recommendations and Future Work

The following recommendations are based on the results obtained up to this point:

Immediate work:

- *Model visualization*: Depending on the project meta-model structure being analyzed, visualization relying on tree structures may prove insufficient. If this is the case, section 8.3 already provided possible alternatives. Visualization relying on third-party tools provides a quick and easy way to display models, as it only requires to change the format of the data exported. However, these tools can hardly be modified or adapted. If users decide they require a specialized display of data, or the current available solutions do not fit their needs, it could be possible to expand VoSMA to also display model data. This could be accomplished using, for example, a third-party library. The advantages of this approach include removing an extra step (no second tool is necessary), removing the need to use external files (no need to export data), and the flexibility to customize the visualization layout. Disadvantages mainly involve having to invest time expanding the tool, and making it heavier (as it also must include visualization). Another option involves developing a solution with Moose[56], the free and open source platform for data analysis. Features include: importing and parsing data, modeling, measuring, querying, mining, and building interactive and visual analysis tools.

- *Other project meta-model structures*: In this study three projects were presented (NVD, WST, and A4DS). However, only the meta-model structure of the WST could be properly analyzed. Other projects could also be read, given that the corresponding meta-model structures and metrics are defined.

- *Model evolution*: One of the long term goals of this project is to provide insight into the evolution of these specific models. The software architecture was built with model evolution in mind, and section 8.4 provided some future pointers. It is recommended to start with a basic *two versions comparison* solution, which can be later expanded to a variation of the *evolution matrix*.

- *More metric rules*: Other metric rules could be designed to analyze different aspects of the model. Beside quantifying the 'quality' of an element, engineers also expressed a need to measure the 'change impact' value (how much will changes in this element impact the model). Other metrics discussed included a 'degree of change' in elements through development (how often, when and how a component has been modified).

- *User friendliness*: The graphical user interface (GUI) was built to provide a *proof-of-concept* of the possibilities of this project. However the tool is meant to be used in the future on a regular basis by engineers at Soltegro. Depending of the user's wishes, a modification of revamp of the GUI would be a necessary step before the software can be used in a regular basis.

[56] `http://www.moosetechnology.org/`

Other possibilities:

- *Expanding functionality*: The *Project Browser* and *Traceability* windows provide similar functionality to their EA counterparts. However, they could be improved with the ability to filter, search, and sort elements. The tree structures should also be populated dynamically (wherever the user requests more information, or expands a node), instead of a static structure with all the elements, which is often unnecessary. Other functionality include similar windows, and an improved search function.

- *Reading models generated by other SysML modeling tools*: The software could be adapted with the capability of reading SysML models by other visual modeling platforms (as long as models can be exported to XMI files). There are two main alternatives: a) write new reading modules for every tool format, or b) use XSLT transformation stylesheets[57] in combination with a XSLT processor (see figure 11.1 for an example). Both are equally feasible, however, the second provides more flexibility and does not require further knowledge of the VoSMA structure.

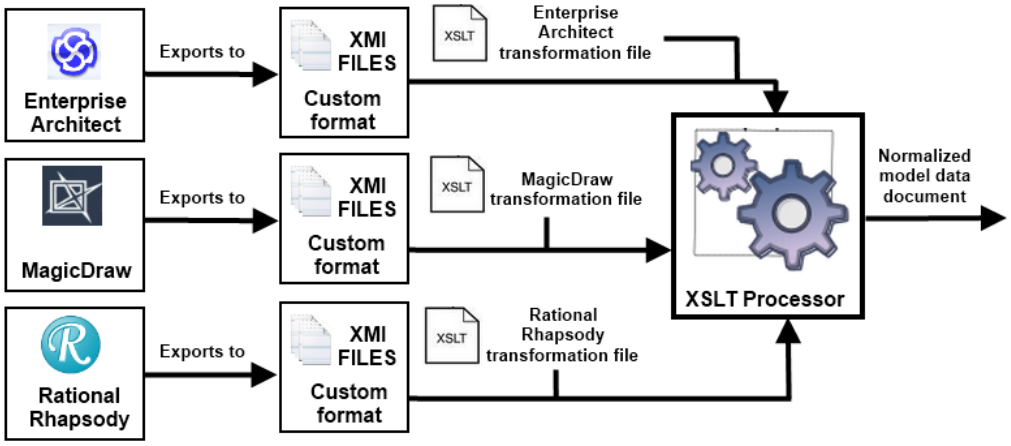


Figure 11.1: Reading SysML model data from different visual modeling platforms. Multiple XSLT transformations are defined which convert custom XMI formats to a normalized data document.

- *Support for the new OMG XMI standard*: As explained in section 4.6, the OMG intends to standardize the XMI format used by visual modeling SysML tools. This standard should be adopted as soon as available, and EA custom format, of older formats, should be supported through the use of XSLT transformation files.

- *Cross platform support*: The VoSMA tool was written using the Java programming language, and thus, thanks to the Java Virtual Machine[58] (JVM) provides cross platform functionality. XMI files contain text based data serialized in a XML

[57] http://www.w3.org/TR/xslt
[58] http://docs.oracle.com/javase/specs/jvms/se7/html/index.html

markup format, and are thus system independent. The software could be easily be adapted to run on different operating systems if necessary.

- *Web client*: Another possibility involves integrating VoSMA into a web client. Since the tool is written in Java, it could be easily be modified to be executed as a Java applet[59], which runs in a Web browser using the JVM or Sun's AppletViewer[60]. This will allow users to access the tool remotely, and since Java's bytecode is platform independent, Java applets can be executed by browsers for different platforms.

[59] http://docs.oracle.com/javase/7/docs/api/java/applet/Applet.html
[60] http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/appletviewer.html

Bibliography

[1] R. L. Ackoff. Towards a System of Systems Concepts. *Management Science*, 17(11):661–671, July 1971.

[2] V. Anderson and L. Johnson. *Systems Thinking Basics: From Concepts to Causal Loops*. Pegasus Communications Inc, Waltham, Massachusetts, 1997.

[3] G. L. Andrienko and N. V. Andrienko. Interactive maps for visual data exploration. *International Journal of Geographical Information Science*, 13(4):355–374(20), June 1999.

[4] D. Aronson. *Overview of Systems Thinking*. Thinking Page, http://www.thinking.net/Systems_Thinking/OverviewSTarticle.pdf (accessed 01-12-2012), 1996.

[5] A.T. Bahill and B. Gissing. Re-evaluating systems engineering concepts using systems thinking. *IEEE Trans Syst Man Cybernet Part C Appl Rev*, 28(4):516–527, 1998.

[6] V. R. Basili. Applying the Goal/Question/Metric Paradigm in the Experience Factory. *Software Quality Assurance and Measurement: Worldwide Perspective*, Internatio:21–44, 1995.

[7] P. Berander and P. Jönsson. A goal question metric based approach for efficient measurement framework definition. In *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06*, page 316, New York, New York, USA, 2006. ACM Press.

[8] J. Boardman and B. Sauser. System of Systems - the meaning of of. In *2006 IEEE/SMC International Conference on System of Systems Engineering*, number April, pages 118–123. IEEE, 2006.

[9] K.E. Boulding. General systems theory : The skeleton of science. *Management science*, 2(3):197–208, 1956.

[10] C. A. Brewer and L. Pickle. Evaluation of Methods for Classifying Epidemiological Data on Choropleth Maps in Series. *Annals of the Association of American Geographers*, 92(4):662–681, December 2002.

[11] M. Bruls, K. Huizing, and J.J. Wijk. Squarified treemaps. In *Joint Eurographics and IEEE TCVG Symposium on Visualization*, 2000.

[12] C. N. Calvano and P. John. Systems engineering in an age of complexity. *Systems Engineering*, 7(1):25–34, 2004.

[13] P. Checkland. Systems thinking. In *Systems thinking, systems practice: includes a 30-year retrospective*, pages 1–11. Wiley, 1999.

[14] D. Cook. *Pulling The Strings*, volume 7. University of Nebraska Press, November 2008.

[15] Dutch Ministry of Economic Affairs. Holland Compared 2012, 2012.

[16] K.E. Emam and N.H. Madhavji. A field study of requirements engineering practices in information systems development. *In Proceedings of the Second IEEE International Symposium on Requirements Engineering. IEEE Computer*, (March), 1995.

[17] J.A. Estefan. Survey of model-based systems engineering (MBSE) methodologies. *Incose MBSE Focus Group*, pages 1–47, 2007.

[18] Federal Aviation Administration. Federal Aviation Administration Safety Management System Manual, 2004.

[19] G. Finance. SysML Modelling Language Explained, 2010.

[20] D. Fisher. *An emergent perspective on interoperation in systems of systems*. Number March. Software Engineering Institute, 2006.

[21] S. Friedenthal, A. Moore, and R. Steiner. A Practical Guide to SysML: The Systems Modeling Language. *Morgan Kaufmann Publishers Inc.*, 2008.

[22] T. Gîrba and S. Ducasse. Modeling history to analyze software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, (18):207–236, 2006.

[23] P. Godfrey. How Systems Thinking contributes to Systems Engineering. In *What Is Systems Thinking?*, number March, pages 1–6. INCOSE UK - Z Guide, 2010.

[24] W.T. Haaf, H. Bikker, and D.J. Adriaanse. *Fundamentals of business engineering and management*. VSSD, 2011.

[25] C. Haskins. *Systems engineering handbook*. International Council on Systems Engineering, 2010.

[26] M. Hause. The SysML Modelling Language. *Fifteenth European Systems Engineering Conference*, (September), 2006.

[27] D. Holten. Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE transactions on visualization and computer graphics*, 12(5):741–8, 2006.

[28] E.C. Honour. Understanding the value of systems engineering. *Proceedings of the INCOSE International Symposium*, 2004.

[29] International Council on Systems Engineering (INCOSE). Systems engineering vision 2020. (September), 2007.

[30] M. Jamshidi. *System of Systems Engineering: Innovations for the 21st Century*. John Wiley & Sons, 2009.

[31] Alexander Kossiakoff, William N. Sweet, Samuel J. Seymour, and Steven M. Biemer. *Systems Engineering Principles and Practice*. John Wiley & Sons, Inc., Hoboken, NJ, USA, April 2011.

[32] C. Lange, M. Wijns, and M. Chaudron. A Visualization Framework for Task-Oriented Modeling Using UML. *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 289a–289a, 2007.

[33] M. Lanza. The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques. In *Proceedings of the 4th international workshop on Principles of software evolution - IWPSE '01*, page 37, New York, New York, USA, 2002. ACM Press.

[34] M. Lanza and S. Ducasse. The Class Blueprint A Visualization of the Internal Structure of Classes. *IEEE Transactions on Software Engineering*, 2001.

[35] M. Lanza and S. Ducasse. Polymetric views - A lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, September 2003.

[36] J. Leonard. *Systems Engineering Fundamentals : Supplementary Text*. Number January. DIANE Publishing, 1999.

[37] E.E. Lewis. *Introduction to Reliability Engineering*. John Wiley and Sons Ltd, 1995.

[38] P. Logan and D. Harvey. Documents as Information Artefacts in a Model Based Systems Engineering Methodology. *5th Asia-Pacific Conference on Systems Engineering (APCOSE 2011)*, (Apcose), 2011.

[39] M.R. Lyu. Software reliability engineering: A roadmap. *Proceeding FOSE '07 2007 Future of Software Engineering*, pages 153–170, 2007.

[40] M.W. Maier. *Architecting principles for systems-of-systems*. 1999.

[41] NASA. NASA Systems engineering handbook. *INCOSE. Version*, (June), 1995.

[42] NASA. NASA Systems Engineering Handbook. *NASA Special Publication*, 2007.

[43] N.V. Westerscheldetunnel. De Westerscheldetunnel: megaproject met grensverleggende boortechniek. (April), 2001.

[44] Object Management Group. Unified modeling language: Superstructure v2.0. (August), 2005.

[45] Object Management Group. Diagram Interchange version 1.0. (April), 2006.

[46] Object Management Group. MOF 2.0/XMI Mapping, Version 2.1.1. (December), 2007.

[47] Object Management Group. OMG Systems Modeling Language v1.1. *OMG Available Specification*, (November), 2008.

[48] Object Management Group. UML Profile for MARTE : Modeling and Analysis of Real-Time Embedded Systems (version 1.0). 15(November), 2009.

[49] Object Management Group. OMG Systems Modeling Language v1.3. *OMG Available Specification*, (June), 2012.

[50] Ingmar Ogren. On principles for model-based systems engineering. *Systems Engineering Journal*, 3(1), 2000.

[51] J. Orr. What is "Systems thinking"? In *Machine Design*, page 405. Addison-Wesley, 2010.

[52] Oxford University Press. Oxford Dictionaries. (April), 2010.

[53] T. Panas, R. Lincke, and W. Löwe. Online-configuration of software visualizations with Vizz3D. In *Proceedings of the 2005 ACM symposium on Software visualization - SoftVis '05*, number i, page 173, New York, New York, USA, 2005. ACM Press.

[54] D. Rosenburg and S. Mancarella. *Embedded Systems Development using SysML*, volume 17. Sparx Systtems Pty Ltd, January 2009.

[55] M.F. Ruland, T.M.C. Daverveld, B.J.W. Duijnhoven, J.A. Gelder, H. Krouwel, and J.M. Teeuw. An integrated functional design approach for safety related tunnel processes. 2012.

[56] M. Ryschkewitsch, D. Schaible, and W. Larson. The art and science of systems engineering. *NASA Systems Research*, pages 1–22, 2009.

[57] T. Saunders, C. Croom, W. Austin, and J. Brock. System-of-systems engineering for air force capability development. *US Air Force Scientific Research SAB-TR-05-04*, (July), 2005.

[58] H.J. Schulz. Treevis. net: A Tree Visualization Reference. *IEE Computer Graphics and Applications*, 31(6):11–15, 2011.

[59] S. Selberg and M.A. Austin. Toward an Evolutionary System of Systems Architecture. *Proceedings of the 8th International Symposium on Systems Engineering (INCOSE)*, (1):1–14, 2008.

[60] S.A. Sheard. Twelve systems engineering roles. *Proceedings of the International Symposium on Systems Engineering (INCOSE)*, (703):1–8, 1996.

[61] S.A. Sheard and J.G. Lake. Systems engineering standards and models compared. *Proceedings of the 8th International Symposium on Systems Engineering (INCOSE)*, 1998.

[62] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, January 1992.

[63] B. Shneiderman. Treemaps for space-constrained visualization of hierarchies. *ACM Transactions on Graphics - TOG*, 1998.

[64] L.W. Smith. Guidelines for Successful Acquisition and Management of Software-Intensive Systems : Weapon Systems Command and Control Systems Management Information Systems Condensed Version Preface. *Department of the Air Force-Software Technology Support Center*, (February), 2003.

[65] M.S. Soares and J. Vrancken. Model-driven user requirements specification using SysML. *Journal of Software*, pages 57–68, 2008.

[66] R. Solingen and E. Berghout. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*, volume 10. McGraw Hill, December 1999.

[67] H. Struiksma, T. Tillema, and J. Arts. Space for mobility: towards a paradigm shift in Dutch transport infrastructure planning? *ACSP-AESOP Fourth joint Congress*, pages 1–16, 2008.

[68] SysML Forum. SysML Forum FAQ. *http://www.sysmlforum.com/ (accessed 01-12-2012)*, 2012.

[69] SysML Team. Systems Modeling Language (SysML) Specification. *OMG document: ad/2006-03-01*, (November), 2006.

[70] S.T. Teoh. A Study on Multiple Views for Tree Visualization. In Robert F. Erbacher, Jonathan C. Roberts, Matti T. Gröhn, and Katy Börner, editors, *Proc. SPIE 6495, Visualization and Data Analysis 2007*, pages 64950B–64950B–12, January 2007.

[71] M. Termeer, C.F.J. Lange, a. Telea, and M.R.V. Chaudron. Visual Exploration of Combined Architectural and Metric Information. *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–6, 2005.

[72] The Official OMG SysML site. What Is OMG SysML? *http://www.omgsysml.org/*, 2012.

[73] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: view-driven software architecture reconstruction. *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, (1):122–132.

[74] Y. Vanderperren and W. Dehaene. SysML and Systems Engineering Applied to UML-Based SoC Design. In *ECMFA'11 Proceedings of the 7th European conference on Modelling foundations and applications*, pages 299–311, 2005.

[75] M.J.C.M. Vromans. *Reliability of railway systems*. 2005.

[76] R. Wettel and M. Lanza. Visualizing software systems as cities. *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop*, pages 92–99, June 2007.

[77] L. Wilkinson and M. Friendly. The History of the Cluster Heat Map. *The American Statistician*, 63(2):179–184, May 2009.

Appendix A

Application of MBSE in Tunnel Projects at Soltegro

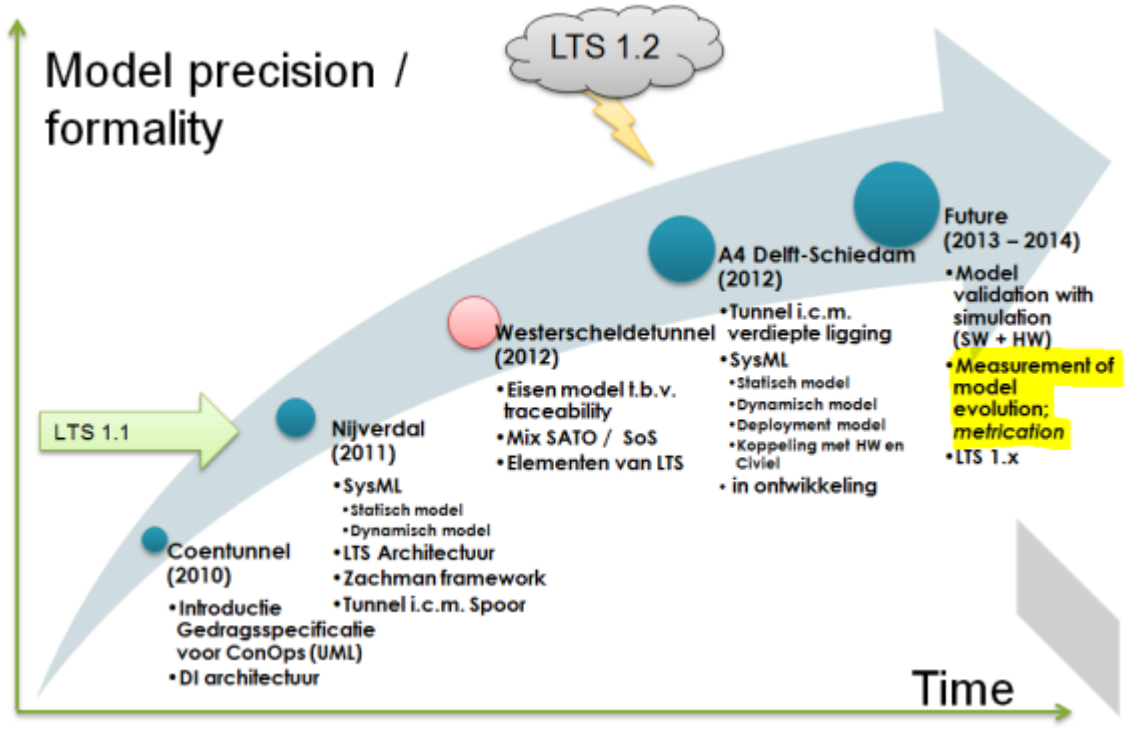


Figure A.1: Application of MBSE in tunnel projects at Soltegro. Adapted from "*INCOSE - Toepassen van Model Based Systems Engineering bij Tunnelontwerpen*" presentation by E. Burgers (December 2012). Note how a higher model precision is desired with each new tunnel project. The expected context of this work is shown under 'Future'.

Appendix B

Definitions of System and Systems Engineering in SE Standards

Standard or Model	Definition of System	Definition of Systems Engineering
MIL-STD- 499B	An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective.	An interdisciplinary approach encompassing the entire technical effort to evolve and verify an integrated and life-cycle balanced set of system people, product, and process solutions that satisfy customer needs. Systems engineering encompasses: **a)** the technical efforts related to the development, manufacturing, verification, deployment, operations, support, disposal of, and user training for, system products and processes; **b)** the definition and management of the system configuration; **c)** the translation of the system definition into work breakdown structures; **d)** development of information for management decision-making.
EIA/IS 632	Same as MIL-STD -499B	Same as MIL-STD -499B
IEEE 1220	The top element of the system architecture, specification tree, or systems breakdown structure that is comprised of one or more products and associated life cycle processes and their products and services.	An interdisciplinary collaborative approach to derive, evolve, and verify a life cycle balanced system solution that satisfies customer expectations and meets public acceptability.
EIA 632	The aggregation of end products and enabling products that achieves a given purpose.	None-the term is not used or referred to in the standard.

ISO 15288	An integrated composite that consists of one or more of the processes, hardware, software, facilities and people, that provides a capability to satisfy a stated need or objective. (ISO 12207 definition adopted for ISO 15288)	None-the term is not used or referred to in the standard.
SE-CMM	An integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective. (MIL-STD-499B definition).	Systems engineering is the selective application of scientific and engineering efforts to • Transform an operational need into a description of the system configuration which best satisfies the operational need according to the measures of effectiveness, • Integrate related technical parameters and ensure compatibility of all physical, functional, and technical program interfaces in a manner which optimizes the total system definition and design, • Integrate the efforts of all engineering disciplines and specialties into the total engineering effort.
SECM (EIA/IS 731)	The aggregation of end products and enabling products that achieves a given purpose.	An interdisciplinary approach and means to enable the realization of successful systems.
SECAMA	A set of interrelated components working together to accomplish a common purpose (CAWG). An interacting combination of elements viewed in relation to function (INCOSE).	An interdisciplinary approach and means to enable the realization of successful systems.

Table B.1: Definitions of *system* and *Systems Engineering* in SE Standards, adapted from [61].

Appendix C

List of Stakeholder Perspectives in Systems Engineering

	Description
User	Defined set of people in the field who benefit from system with relatively well understood expectations.
Developer	System developed by prime contractor and subcontractors with single program office. High degree of focus on the integration of the system components to specified level of performance.
Trainer	Trainers with high degree of knowledge of system deliver training based on system intended use; training changes incrementally when system upgrades are delivered.
Tester	Verification and validation in accordance with V-model of system to well specified set of requirements and documented needs; master test plan drives testing activities.
Sustainer	Sustainment needs and requirements planned in front end of systems effort; sustainment activities performed by single organization..
Acquirer	Sound acquisition practices used to select and manage prime system contractor. Direct relationship between acquisition and contractor program offices.
Researcher	SE research efforts are typically program focused; performed in program laboratory involving modeling and simulation; and research is performed in early phases of system development or directed at specific system enhancements in later phases. Research is directed at program/domain specific problems.

Table C.1: List of stakeholder perspectives in Systems Engineering, adapted from [57].

Appendix D

Frequently Used Terms in Systems Engineering

Term	Definition
Activity	set of actions that consume time and resources and whose performance is necessary to achieve or contribute to the realization of one or more outcomes
Baseline	a specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures
Enabling system	a system that complements a system-of-interest during its life cycle stages but does not necessarily contribute directly to its function during operation
Enterprise	that part of an organization with responsibility to acquire and to supply products and/or services according to agreements
Organization	a group of people and facilities with an arrangement of responsibilities, authorities and relationships
Process	set of interrelated or interacting activities that transform inputs into outputs
Project	an endeavor with start and finish dates undertaken to create a product or service in accordance with specified resources and requirements
Resource	an asset that is utilized or consumed during the execution of a process
Stage	a period within the life cycle of a system that relates to the state of the system description or the system itself
Stakeholder	a party having a right, share or claim in a system or in its possession of characteristics that meet that party's needs and expectations

Supplier	an organization or an individual that enters into an agreement with the acquirer for the supply of a product or service
System	a combination of interacting elements organized to achieve one more stated purposes
System element	a member of a set of elements that constitutes a system
System-of-interest	the system whose life cycle is under consideration
Trade-off	decision-making actions that select from various requirements and alternative solutions on the basis of net benefit to the stakeholders
Validation	confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled
Verification	confirmation, through the provision of objective evidence, that specified requirements have been fulfilled

Table D.1: Frequently used terms in Systems Engineering (from ISO/IEC 15288)

Appendix E

Comparison Between SysML and UML Diagrams

SysML Diagram	Purpose	UML Diagram Analog
Activity diagram	[Behavioral diagram] An Activity diagram shows system behavior as control and data flows. Useful for functional analysis. Compare Flow Block Diagrams (FBDs) and Extended Functional Flow Block diagrams (EFFBDs), already commonly used among systems engineers.	UML::Activity diagram
Block Definition diagram	[Structural diagram] A Block Definition diagram shows system structure as components along with their Properties, Operations and Relationships. Useful for system analysis and design.	UML::Class diagram
Internal Block diagram	[Structural diagram] An Internal Block diagram shows the internal structures of system components, including their Parts and Connectors. Useful for system analysis and design.	UML::Composite Structure diagram
Package diagram	[Structural diagram] A Package diagram shows how a model is organized into Packages, Views and Viewpoints. Useful for model management.	UML::Package diagram
Parametric diagram	[Structural diagram] A Package diagram shows parametric constraints between structural elements. Useful for performance and quantitative analysis.	[No analogous diagram in UML]
Requirement diagram	[Requirement diagram] A Requirement diagram shows system requirements and their relationships with other elements. Useful for requirements engineering, including requirements verification and validation (V&V).	[No analogous diagram in UML]
Sequence diagram	[Behavioral diagram] An Sequence diagram shows system behavior as interactions between system components. Useful for system analysis and design.	UML::Sequence diagram
State Machine diagram	[Behavioral diagram] A State Machine diagram shows system behavior as sequences of states that a component or interaction experience in response to events. Useful for system design and simulation/code generation.	UML::State Machine diagram

Use Case diagram	[Behavioral diagram] A Use Case diagram shows system functional requirements as transactions that are meaningful to system users. Useful for specifying functional requirements. (Note potential semantic overlap with functional Requirements specified in Requirement diagrams.)	UML::Use Case diagram
Allocation Table	[Mapping table; not a diagram] An Allocation Table shows various kinds of assignment relationships (e.g., requirement allocation, functional allocation, structural allocation) between model elements. Useful for facilitating automated verification and validation (V&V) and gap analysis.	[No analogous table in UML]
Instances Instances	As per the OMG SysML 1.2 minor revision, Instance Specifications, but not Object diagrams are allowed.	UML:: Object diagram
[No analogous diagram in SysML]		UML:: Communication diagram
[No analogous diagram in SysML]		UML:: Component diagram
[No analogous diagram in SysML]		UML:: Deployment diagram
[No analogous diagram in SysML]		UML:: Interaction Overview diagram
[No analogous diagram in SysML]		UML:: Profile diagram
[No analogous diagram in SysML]		UML:: Timing diagram

Table E.1: Comparison between SysML diagrams and their UML counterparts, adapted from [68].

Appendix F

Test Case Examples in EA

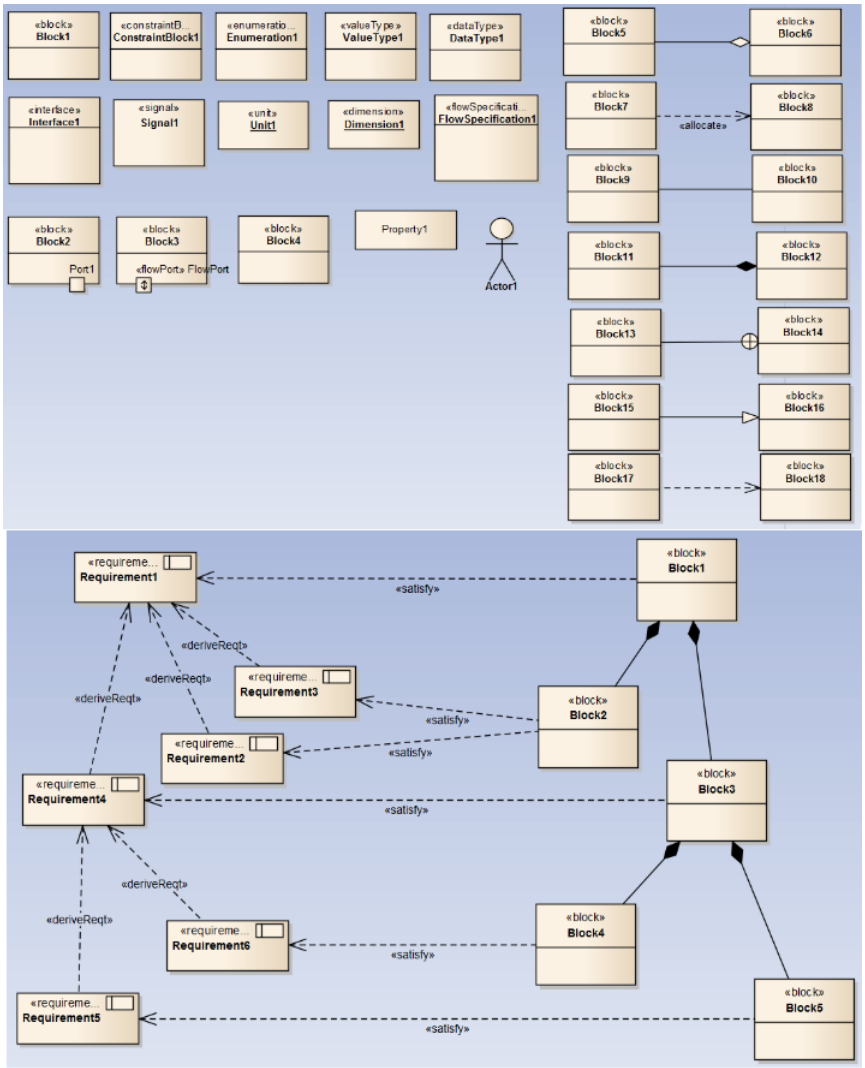


Figure F.1: Example of a test cases in EA to assess element information read. The top case is meant to assess that all elements and relations are read correctly. The bottom case is meant to assess that the project-specific structure is correctly identified and reconstructed.

Appendix G

User Feedback

G.1 Interview with Remco Luitwieler

Date: 07-01-2013.
Experience with EA and SysML: Currently working on the WST and A4DS projects.

Remco was generally positive about the project. His main point of criticism involved the current implementation of the software: *could it not be replaced with templates and/or scripts in EA which analyze the model and generate the corresponding data?* Indeed, a valid (and common) point of concern. Still, as explained before, this solution would involve relaying on the EAP files, which are not stored on the SVN. This is especially inconvenient when dealing with a model's evolution, as different project revisions would need to be downloaded and the EAP reconstructed, instead of a SVN file scan as is the case with the current solution.

He appreciated the extra functionality provided by the software when dealing with project-specific meta-models, and he did not really mind having to rely on a tool outside EA to perform the analysis. His recommendations for future work included a 'consistency check' functionality (outside of the current scope of this project). This would check the model for completeness and correctness, which apparently is not working properly in the current version of EA. His main expectations for future versions is to be able to measure 'quality' of elements, especially the ability to identify hot-spots in a model.

G.2 Interview with René Krouwel

Date: 09-01-2013.
Experience with EA and SysML: Currently working on the NVD project. In a previous project he was employed by KLM - Dutch Royal Airlines to work on solutions in EA to integrate luggage and ticket systems with Air France.

René's main concerns involved the measurements chosen to determinate the quality of an element, and the current visualization layout. The former is related to the complex question of how to actually quantify the 'quality' of an element. The latter is related to the cluttered view of the display using a tree-map layout. He proposed to apply an alternative 'list' layout to focus on the elements that may need rework. However, he also pointed many (potential) benefits. Though the current solution could not read the NVD meta-model (his current SysML project), he presented multiple ideas for future work:

1. Identify 'unbalanced' requirements. These are places in the decomposition tree with an uneven distribution of elements of levels, which may indicate the need for further decomposition.

2. Element evolution, specially requirements evolution: René showed special interest in the visualization of requirements, especially in the early stages of a project.

3. Visualize the system decomposition value: René also proposed developing a value for the 'quality' of the decomposition, which could help identify element which may require further refinement.

4. Identify requirements which have undergone multiple updates. A high number of updates (may often) indicate it is a 'mature' element, as it has experienced multiple reviews. These elements are expected to experience less changes in future iterations, and may be considered 'safe' to use.

5. List based visualization. René proposed a visualization methodology were elements were ranked according to certain (to be developed) metrics. The goal is to show elements which require immediate attention at the top. He noted that this will provide a less cluttered display than the current tree-map implementation. He also proposed the ability to select which elements to visualize. Another proposition was to assign another color to an element once the threshold value is exceed, to easily identify elements requiring refactoring. He expected that this clear and simple visualization methodology could be also be presented to the less technical project managers, so they also could get a better insight on the work that needed to be done.

6. Identify elements with a low complexity which have undergone multiple modifications. This may indicate that the elements has not been understood, or that the assigned complexity value is incorrect. Also, if a certain percentage of elements in a model show this behavior (he named a value of 30%) it could be a sign that the model has not been well understood.

7. Identify elements with high a complexity value which required a lesser amount of work or modifications than elements with a lower complexity value. This may pinpoint elements which still require work (as it is expected that a high complexity value leads to a high amount of work).

René also noted that gaining actual insight into a meta-model could lead to eventual cost savings. For example, by identifying problems with the model in early stages of development, or by having a better picture of how changes could impact the rest of the model before they are made.

G.3 Review session with André Stehouwer and Eric Burgers

Date: 11-01-2013.
Functions: André and Eric were the main company supervisors of this project. André was tasked with the general work process, while Eric was charged with the technical aspects.

The session was concerned with a central question: *have the goals set at the start of this project been met?* As with the interview with René Krouwel, this session also led to a brainstorm of ideas. The review involved a detailed walk-trough the software using the latest version of the WST project, showing all the currently implemented functionality. Eric appreciated the Project Browser and Traceability windows (similar to EA). The choice to present the WST project meta-model as two trees structures side by side seemed to provide a clear picture of the model's structure. Although this project focused on visualization with rectangular tree-map layouts, the choice to also implement multiple layouts in Treeviz paid off. When cycling through the different choices, some layouts were pointed for other possible useful tasks, such as presenting information to less technical involved project members (such as management), or identifying the possible impact of changes in the model when deliberating with other members of a project. Thus, multiple layouts could be implemented aimed to fulfill different tasks. It was also remarked that one of the stated main strengths of this project is that it provides additional functionality currently not present in visual modeling platforms, especially when it comes to project-specific meta-model identification, checking, measurement and visualization. Both users stated that functionality provided by the user-interface was clear. Eric showed a lot of interest in starting using the tool as soon as possible.

To answer the question posed at the start of this section: it was stated that the software does serve as a strong framework to build upon. The functionally and visualization of model data could lead to a better insight into project-specific meta-models. It was remarked that one of the next steps should involve a meeting between potential users to decide where to take the tool next, considering the possibilities presented through this study.

G.4 Interview with Franc Fouchier

Date: 11-01-2013.
Experience with EA and SysML: Currently working on the NVD project.

After a short introduction of the software, it was attempted to read the latest version of the NVD project. However, the NVD's meta-model is not defined in VoSMA at this point, so no useful information could be obtained (although SysML model data was correctly read, including project structure and element traceability). Instead the latest WST revision was used to showcase the tool functionality.

Franc appreciated the tree-map visualization, noting that if coupled with the correct metrics could lead to a fast identification of regions in the models which may need to be reviewed. However, he was especially concerned with (change) impact analysis: identifying the potential consequences of a change in the model. He desired a solution where selecting an element will show the user all the paths and elements which may be affected by changes. Another idea he proposed involved checking if functionality assigned to systems in a level was fully implemented in the following level, and that no systems were present with unassigned functionality.

Appendix H

Modeling Methodology in the WST Project

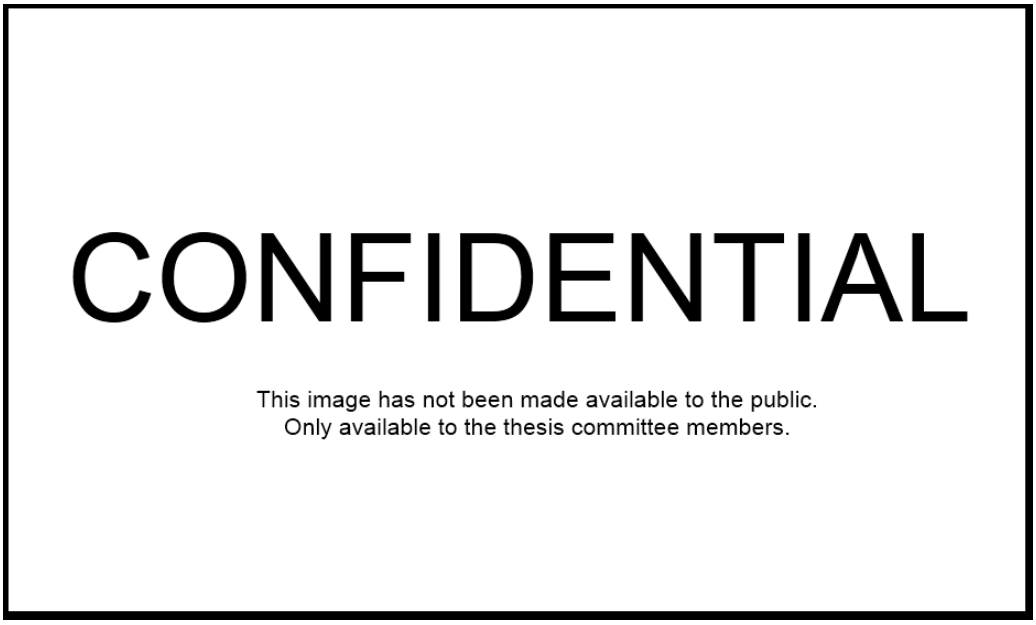


Figure H.1: Modeling methodology in the WST project at Soltegro.

Appendix I

Visualization of the WST Project Meta-Model Data

This section provides an example of results produced by the current implementation of the VoSMA software tool, and the multiple visualization layouts discussed in sections 8.2 and 8.3. The following figures present the architecture and measurement of the project meta-model from the Westerschelde project used at Soltegro (revision 9571). When presenting the model information, the structure is divided into the requirements and systems (blocks) decompositions.

VoSMA VoSMA relies on tree structures to present both decompositions side by side, with additional panels on the right to show metric information. Figure I.1 shows the "plain" layout option, while figure I.2 shows the "rich" layout variant. Although levels in the hierarchy are easy to identify, note how VoSMA struggles to display structures with a large number of elements, and metric information can only be retrieved when selecting an element.

Treeviz The same data is also visualized using the Treeviz tool. The metrics presented (whenever possible) are as follows: for blocks box size represents $BLCK_{COMPLEXITY}$ and box color $BLCK_{REQT\_DENSITY}$. For requirements, box size stands for $REQT_{DIFFICULTY}$ and box color for $REQT_{FRAGMENTATION}$. A larger box size or darker color represent a higher metric value. Every element has a label with its name, and mouse-over shows more element information to the user. Most layout options allow to choose between showing all elements at once, or showing one level at a time. Note that rectangular tree-maps provide the best layout when displaying all elements in the model at once (as the whole screen can be used). Still, a large number of elements in a project leads to very small (and thus hard to read) boxes.

Figure I.3 shows the functional decomposition using the Hyperbolic Tree layout. Note that the top-requirement element is situated in the center (level 1). The "inner" circle contains elements in the next level, and so on. The same data is also presented using the Sunburst Tree layout in figure I.4. The inner circle contains a dummy 'Files' element (constrain of the current Treeviz demo version). Elements are shown in concentric circles,

with the lowest level (1) in the middle (which contains the top-element). Lastly, the same data is presented with a rectangular tree-map layout, as shown in figure I.6. Note that one level is presented at a time (level 2 in this case). All layout options allow to display extensive element information to the user on mouse-over.

MS Treemapper Finally, the WST model data is also visualized using the MS Treemapper tool. The metrics presented are as follows: for blocks box size represents $BLCK_{COMPLEXITY}$ and box color $BLCK_{REQT\_DENSITY}$. For requirements, box size stands for $REQT_{DIFFICULTY}$ and box color for $REQT_{FRAGMENTATION}$. A larger box size or darker color represent a higher metric value. Every element has a label with its name, and mouse-over shows the element's path to the user. Note that all elements in a structure are displayed at the same time. Figure I.7 shows the structural decomposition, and figure I.8 the functional decomposition. Note how in both visualizations the leaf elements have the same size. This a consequence of using the complexity and difficulty values, which are almost always set to 'Easy' (in the case of blocks), and 'Medium' (in the case of requirements) at this point of development.

Figure I.1: WST model data using VoSMA's "plain" layout option.

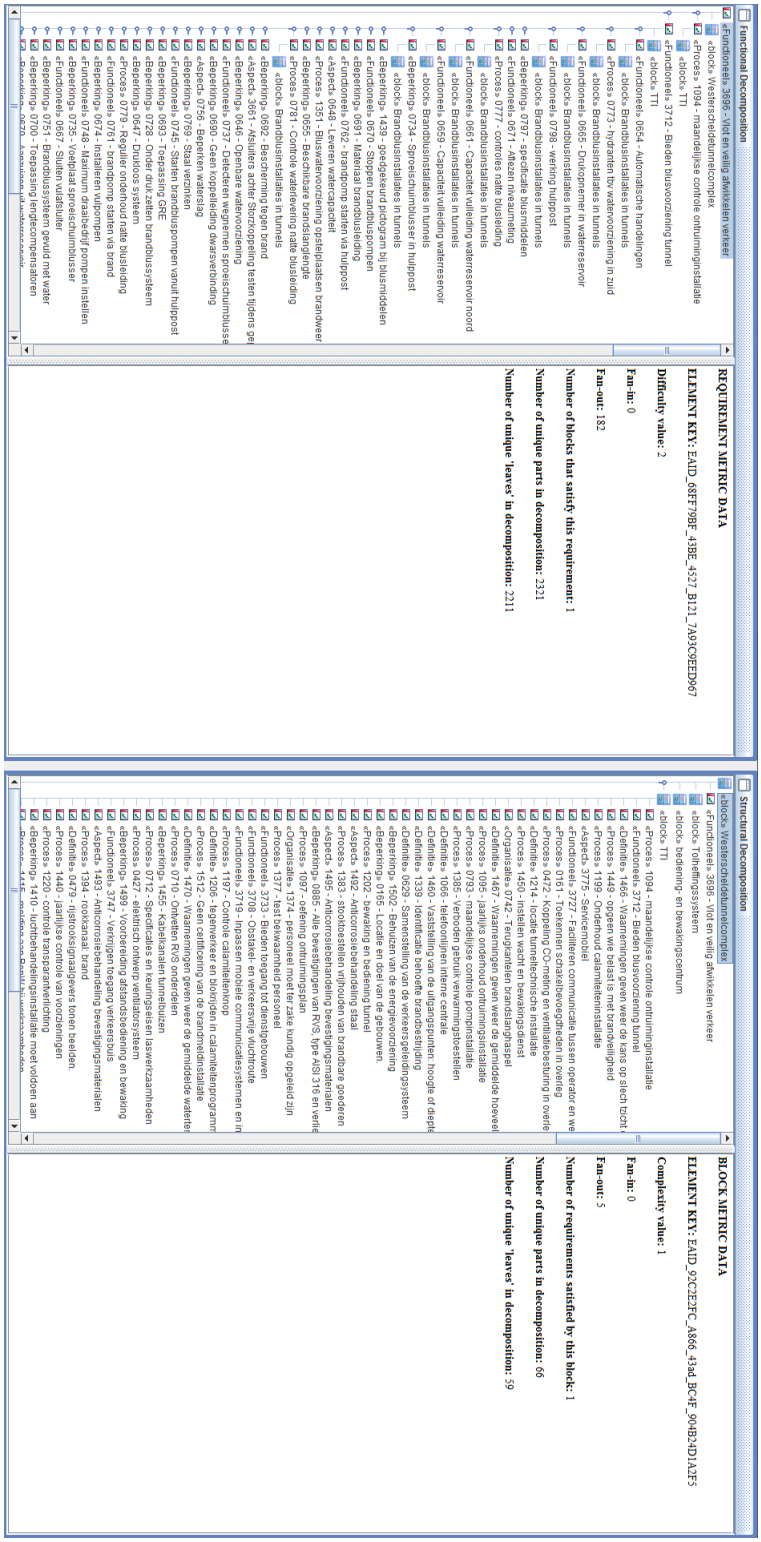


Figure I.2: WST model data using VoSMA's "rich" layout option.

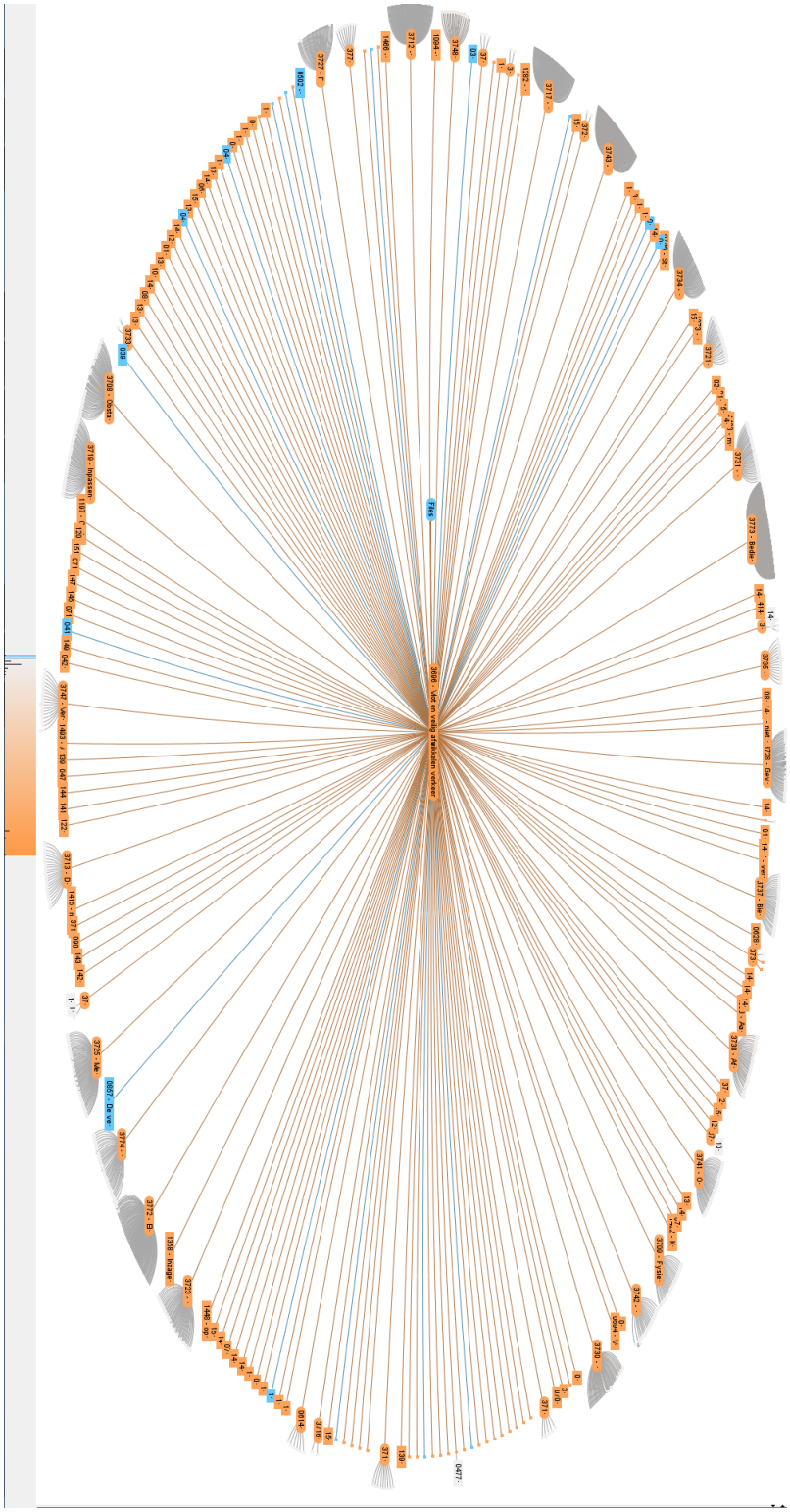


Figure I.3: WST functional decomposition in Treeviz with the Hyperbolic Tree layout.

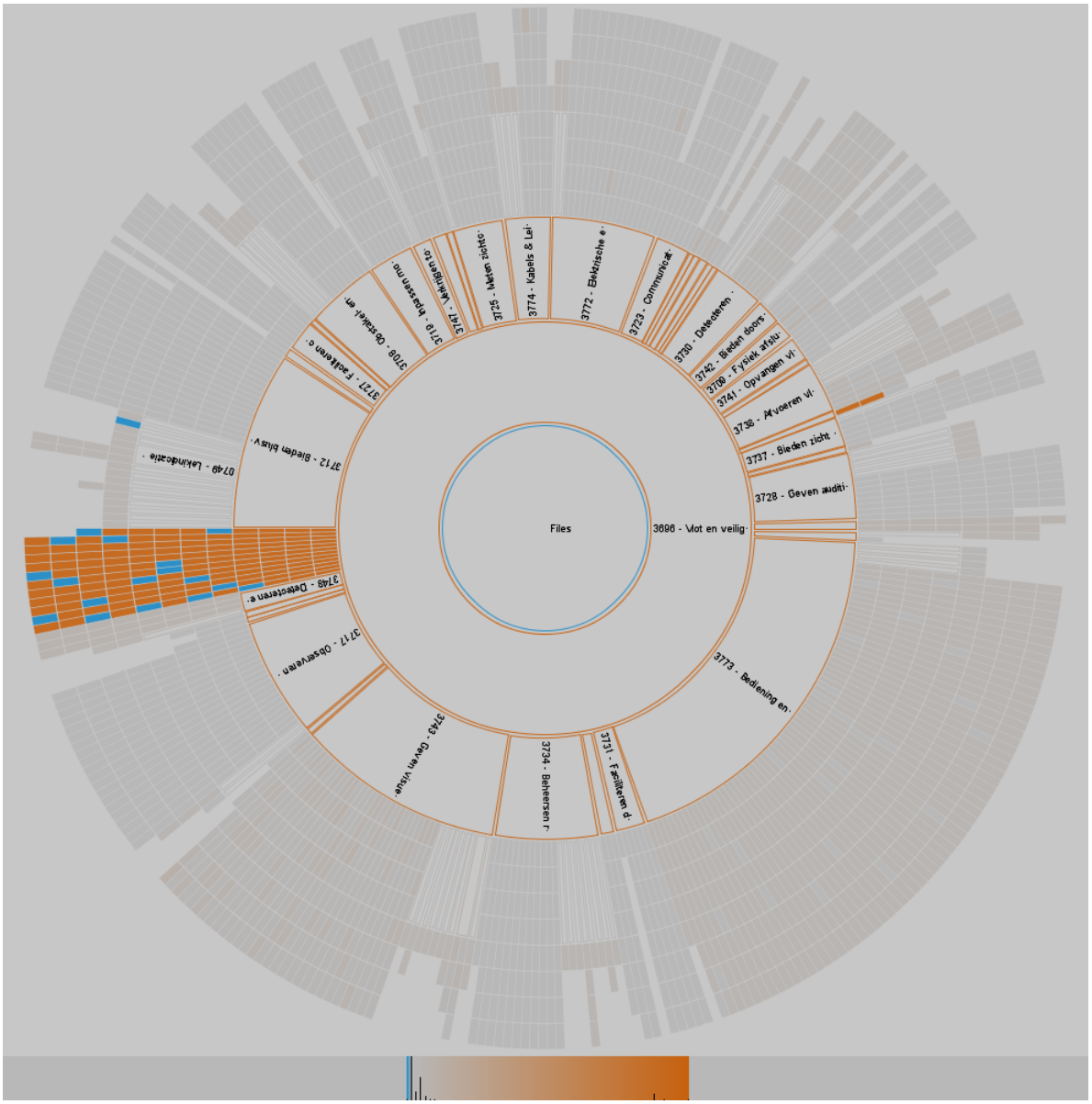


Figure I.4: WST functional decomposition in Treeviz with the Sunburst Tree layout. Note that the figure's brightness was turned down in order to properly display the elements with a grey border.

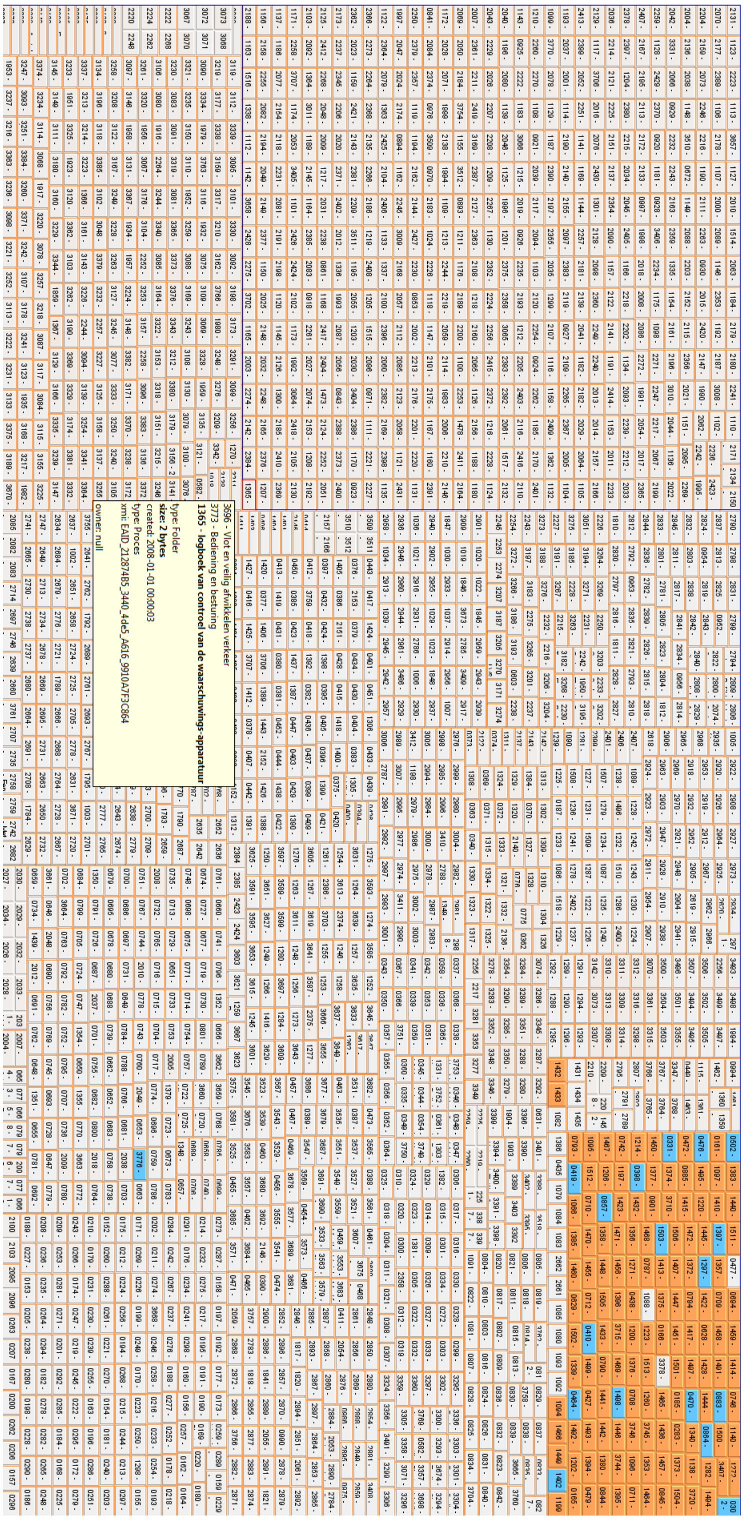


Figure I.5: WST functional decomposition in Treeviz with the Rectangular Treemap layout (full depth).

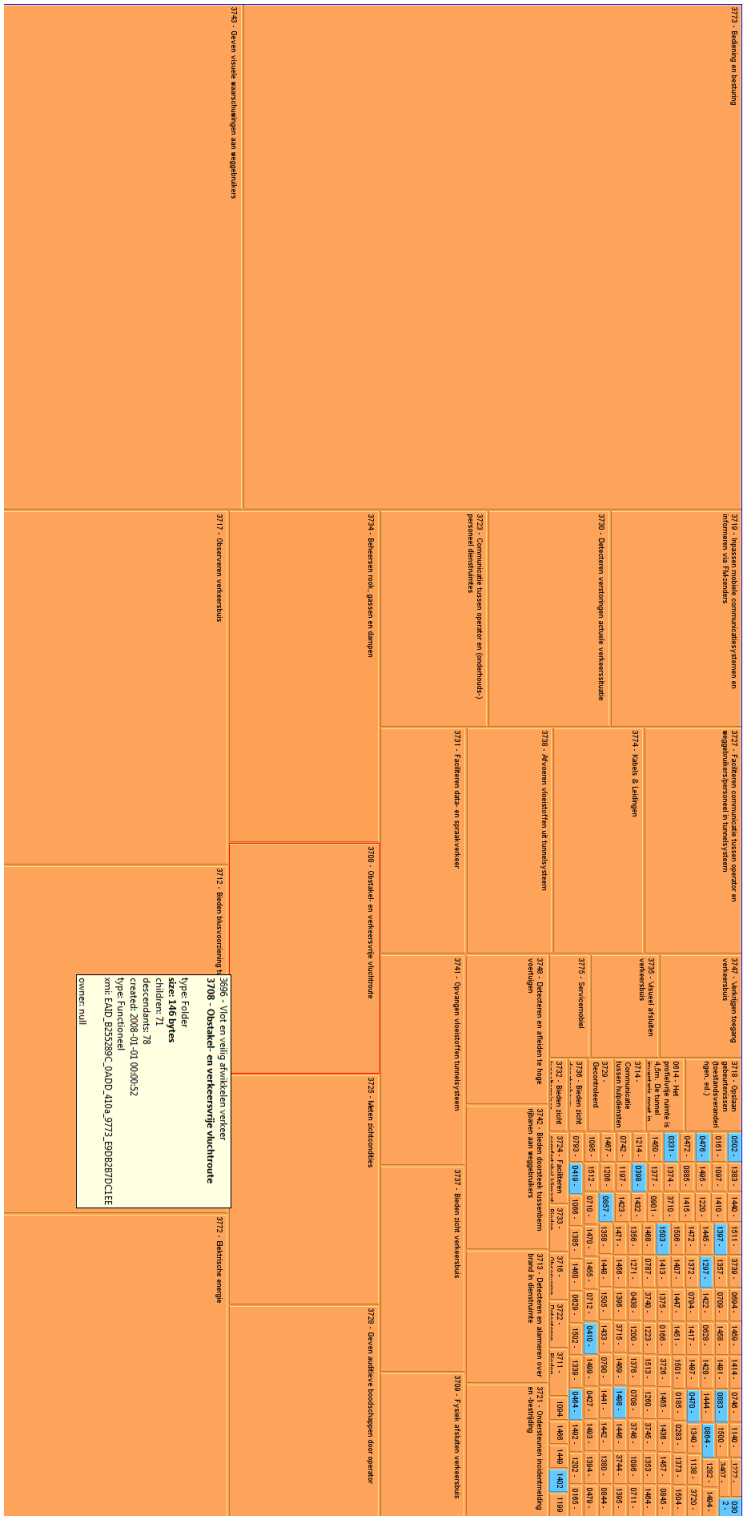


Figure I.6: WST functional decomposition in Treeviz with the Rectangular Treemap layout (current depth).

Figure I.7: WST structural decomposition in MS Treemapper.

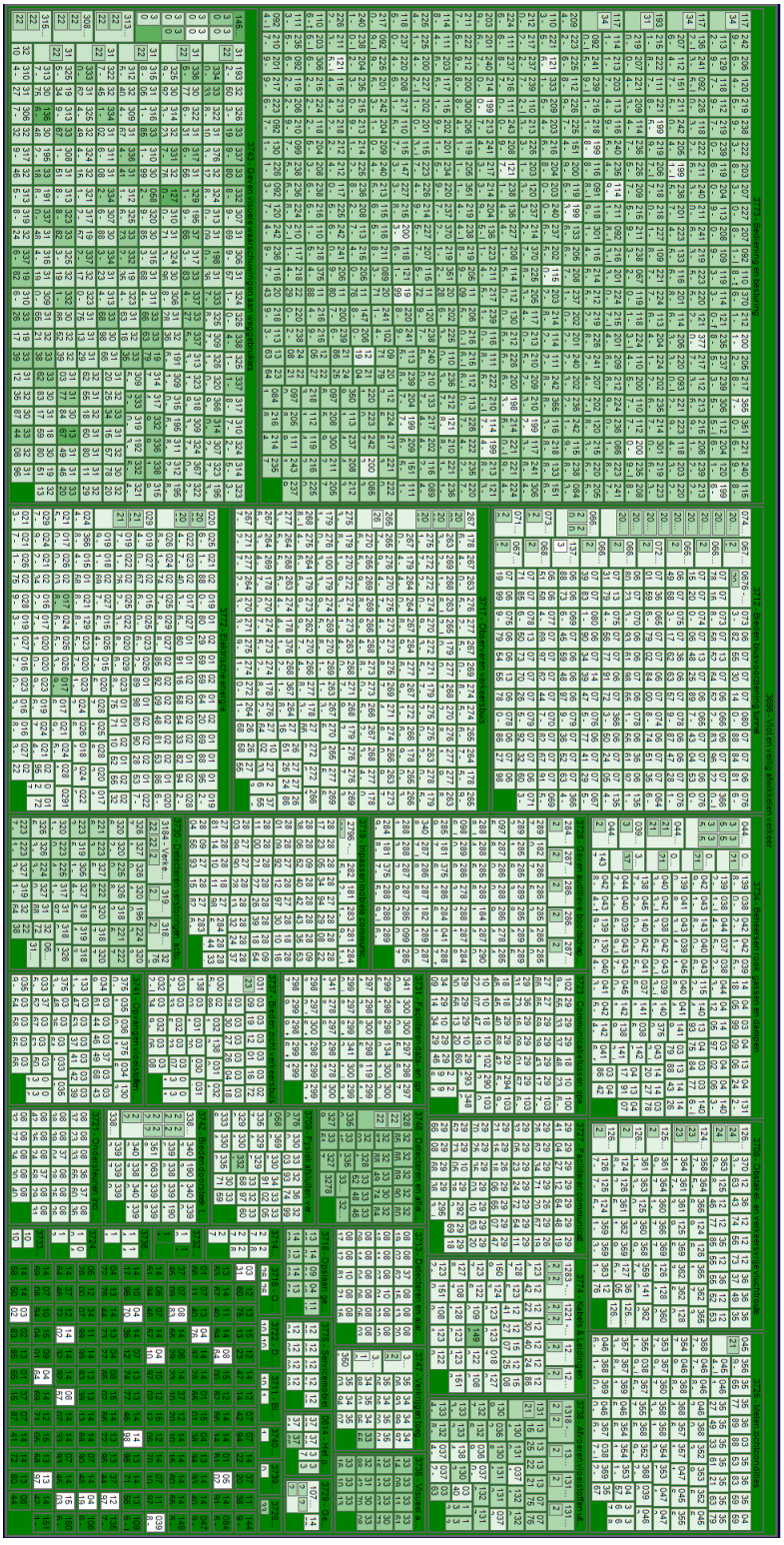


Figure I.8: WST functional decomposition in MS Treemapper.