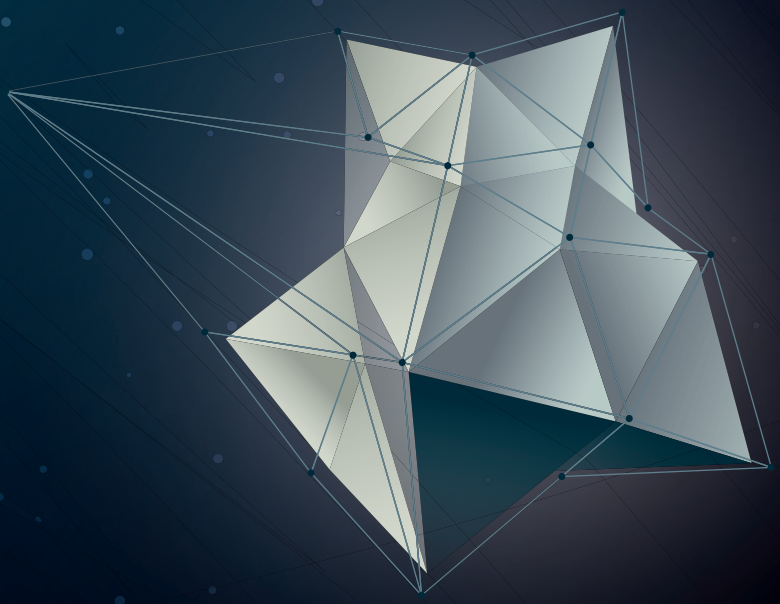
Christian Hennig



From Data Modeling to

Knowledge Engineering in Space System Design

KIT Scientific Publishing

Christian Hennig

From Data Modeling to Knowledge Engineering in Space System Design

From Data Modeling to Knowledge Engineering in Space System Design

by
Christian Hennig

KIT Scientific Publishing

Dissertation, Karlsruher Institut für Technologie
KIT-Fakultät für Wirtschaftswissenschaften

Tag der mündlichen Prüfung: 15. September 2017
Referenten: Prof. Dr. Rudi Studer
Prof. Dr. Oliver Bringmann

Impressum

Scientific Publishing

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

www.ksp.kit.edu

Print on Demand 2018 – Gedruckt auf FSC-zertifiziertem Papier

ISBN 978-3-7315-0720-8
DOI 10.5445/KSP/1000073688

Abstract

In the current approach to designing space systems, models encompassing a wide range of discipline-specific and interdisciplinary aspects, representing the overall product design, are becoming increasingly important. These system-wide models are usually based on object-oriented modeling principles, supporting numerous data management functions for enabling inter-disciplinary data exchange. However, these models often are not able to capture the actual semantics of the underlying engineering data, and thus cannot be used to determine if the represented model describes a correct system.

This work explores ways to provide such system-wide models with genuine semantics of the space engineering domain, enabling functionalities such as automated identification of single points of failure, automated identification of critical system elements, and automated determination of the implications of large amounts of system execution data.

For this purpose, three key elements are provided: A conceptual modeling language for specifying system engineering data that bridges the gap between object-oriented and ontological semantics, a methodology for deriving the data specification from actual engineering data, and a Conceptual Data Model that formalizes key aspects of the data required in space system design.

These three elements are applied to produce a representation of the hypothetical MagSat spacecraft, derived from actual design data, demonstrating the utility of the increased data exploitation functionality enabled by the described approach.

Content

List of Figures

List of Tables

Preface

The research documented in this work was initiated as the capabilities of the system models employed within today's system design processes were felt to somehow not yet being able to reach their conceivable potential. While classical data management activities such as versioning, branching and merging, import and export, basic consistency checking, and data reuse were mastered without running into major difficulties, using the same data for inferring actual design knowledge about the system proved to be more challenging.

However, technologies such as the Web Ontology Language OWL 2 that are supposed to do just that, provide genuine, mathematical-logical, machine-interpretable semantics to data, were also known, but not really employed in this context. Bringing together the domains of space system design and semantic modeling quickly proved to be an interesting, but not always easy, endeavor, for correctly grasping all of the implications of OWL 2's semantics, and making them compatible with existing system engineering problems, required considerable effort.

This thesis can be read in a number of ways. Going over all chapters in the order they appear might be quite interesting for people with a fascination in fundamental concepts of modeling languages and modeling language design. The middle chapters might be quite dry for readers that merely want to understand the end result in which case it is recommended to read Chapters 1 through 5 to understand the general context and problem, and then skip forward to Chapter 9, which explains the application of developed functionality to a concrete scenario. For an in-depth understanding of particular aspects, a selective browsing of the in-between chapters is then recommended.

Friedrichshafen, July 2017

Christian Hennig

1 Introduction

This chapter provides an introduction and overview to the research contained in this thesis. Starting with the initial problem statement, the research goal, context, and research questions to be answered will be explained. Subsequently, the overall approach is detailed, contributions are made explicit, and an overview of the structure of this thesis is given.

1.1 Problem Statement

The principle of Systems Engineering (SE) has been established as an important approach to ensuring the successful design of complex systems, such as automobiles, aircraft, and spacecraft (INCOSE, 2015). During the last years, SE activities have become more and more model-based, utilizing digital representations of the systems to be designed as an important element for facilitating and supporting engineering activities. However, the models and processes revolving around these system-wide models still exhibit numerous shortcomings:

On the one hand, the process used for specifying the data relevant for describing the system in the required level of detail is usually an ad-hoc, top-down approach without explicit guidelines, resulting in a rather loose connection to actual engineering data. Consequently, these data specifications vary significantly between modelers, and often lead to discussion about the correct way to describe data. Also, these data specifications are often significantly influenced by the implementation technologies that will be used to produce a system modeling tool or system database from the specification, often sacrificing true data semantics for ease of implementation, moving the implemented semantics of the system away from those originally intended.

On the other hand, the model specification technologies used in this context exhibit a number of shortcomings. These include the lack of capability to model constraints in a conceptual manner (Hennig, et al., 2015), the restriction to use only a single genuine typing relation for data (Hennig & Eisenmann, 2016), the lack of mechanisms to

formalize and store existing knowledge accumulated across past projects, and the capability to use it for inferring new information on current engineering data.

Thirdly, a lack of alignment to actual SE needs can be observed in current data specifications. This includes inadequate support for classical SE activities such as uncertainties engineering (Hennig & Eisenmann, 2014), and no consideration of the temporal dimension of engineering data (Hennig & Eisenmann, 2014). Furthermore, the fact that the data contained by system models has a strong connection to both product lifecycle management and discipline-specific engineering processes is not treated adequately, resulting in a manual search for, and extraction of, relevant data when moving closer towards system development milestones.

The consequence of these shortcomings is that, although a lot of data about the system to be designed is available, the utilization of data is often not as extensive as is conceivable. While a significant amount of data exists for describing a system at system level, complemented by relevant discipline data, the necessary semantic connections required for determining whether the data represents a well-designed system, an inconsistently described system, or a system with design errors cannot be made. This leaves significant room for improving the data specification as used in the design of space systems, enabling faster time to market, saving development cost, and improving system quality.

1.2 Goal, Context, and Research Questions

The main goal of this thesis is to improve the design process of space systems. This is to be achieved by enabling numerous new functionalities within the System Model (SM) that forms the digital description of the system.

Although the problems, principles, and solutions outlined later in this thesis may be applicable to other engineering domains, or even to problems outside of engineering, the claims and statements made in this thesis apply to the domain of model-based engineering of space systems in the context of the European space industry.

In order to enable new functionalities that improve the utility of the SM, requirements on the SM and its meta-artefacts are formulated. Consequently, an analysis is performed that determines how well industrially established solutions to system modeling, as well as more advanced, but less employed approaches in this field, are able to satisfy these requirements. This leads towards the first research question (RQ):

(RQ1) To what extent are current solutions to system modeling able to fulfil the needs that result from existing challenges of the MBSE process?

Given the hypothesis that currently established solutions for system modeling are not able to fulfil all requirements, an improvement approach is defined that is based on improving the SM's meta-artefacts, being domain data specification, the language in which the domain data specification is modeled in, and the procedure used for modeling, resulting in three further RQs:

(RQ2) What is an appropriate language design for satisfying the requirements on domain data specification?

(RQ3) What is an appropriate procedure for systematically specifying engineering data?

(RQ4) What is an appropriate structure and content of the system model specification in order to meet defined needs?

Given that an improved modeling approach enabling a greater utility of system design data is provided, this might result in an impact on how system design data is represented, and how it can be utilized. Especially in the case that model semantics are improved considerably, the way of executing selected engineering activities might change, meaning that, for example, an engineering activity that was performed manually can now be performed with a significant degree of automation.

Answering these questions draws a picture of current requirements on the examined engineering process and how well these are satisfied by existing modeling technologies, explaining how the given technologies and their application can be improved, and detailing the various benefits that arise from the improvements in the three identified areas.

1.3 Approach and Contributions

The approach pursued for this research starts with an analysis of the current state of the art in system modeling in both industrial and research-oriented domains. Consequently, requirements are formulated, outlining current needs on the SM, and contrasted to system modeling approaches from the different domains. Based on this analysis, a strategy for improvement is derived that is based on the hypothesis that the utility of the SM can be improved by improving its meta-artefacts, being the Conceptual Data Model (CDM), the language in which the CDM specified, and the procedure the CDM is specified with. This improvement leads to a number of benefits occurring within the SM, which are demonstrated using a variety of demonstration cases. This approach is outlined in Figure 1.1.

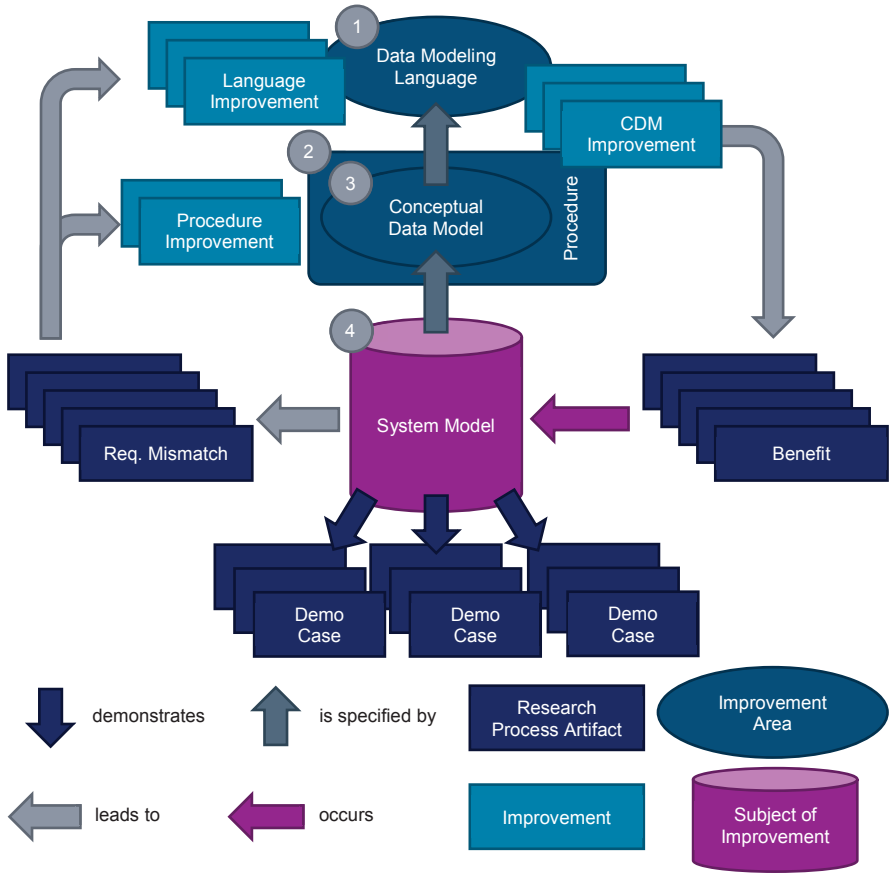


Figure 1.1: Research approach

Furthermore, Figure 1.1 highlights several contributions of this thesis, being:

- Definition of a language for describing a CDM of engineering data (1). This language picks up on an established language for software modeling and provides a link to a language oriented on knowledge modeling. Furthermore, concepts dedicated to solving challenges in a model-based based space system engineering process are provided.
- Definition of a procedure for deriving a CDM from concrete engineering data in a structured, bottom-up manner, ensuring compliance to identified needs (2).
- Definition of a CDM for space system design (3) that can be employed for the model-based engineering of such systems, supporting the activities and functions required by the model-based space system engineering process. In line

with the language architecture, the CDM consists of two models, an object-oriented model, and an ontology.

- Forming the data basis for evaluation, this work provides a representative example of a satellite dataset (4) derived from an actual spacecraft project.

In addition to the contributions explicitly outlined in Figure 1.1, a number of further contributions are made. These include:

- A definition of requirements on the system modeling approach in an MBSE context, paired with an analysis of how current approaches are able to satisfy these requirements.
- An in-depth comparison of selected modeling languages relevant for the space engineering domain, examining and comparing a variety of language properties from numerous perspectives.
- An overview of engineering activities best performed in a knowledge-based modeling environment, and those best performed in a software-driven modeling environment.

1.4 Thesis Structure

This thesis is structured ten chapters, focused on the following subjects:

Chapter 2 provides an introduction to the context in which this work is situated, describing the approach of SE, the principle of Model-Based Systems Engineering (MBSE), and the role of SMs in developing space systems.

Chapter 3 describes conceptual and technological foundations that are considered relevant background information for understanding the technical parts of this thesis. This includes a description of principles central to software engineering, such as the Model-Driven Architecture (OMG, 2014a) and Meta-Object Facility (OMG, 2015a), as well as an outline of commonly used modeling and specification languages such as the Unified Modeling Language UML (OMG, 2015b), the Systems Modeling Language SysML (OMG, 2015c), the Ecore language (The Eclipse Foundation, 2016c), and the Web Ontology Language OWL 2 (W3C, 2012a). If an understanding of these concepts is already present, this chapter may be skipped.

Chapter 4 takes a survey of existing approaches for producing an SM, with a view on both industrially established approaches, and approaches that may be of relevance, but are currently not in widespread productive use.

Chapter 5 provides an analysis of the system modeling approaches outlined in chapter 4, highlighting shortcomings in the current state of the art. These shortcomings are

compared to the requirements on system modeling in the industrial context that form the basis for answering RQ1. The analysis forms an extension and backing of positions taken earlier on the role of knowledge-oriented modeling in the domain of space engineering (Hennig & Eisenmann, 2014; Hennig & Eisenmann, 2016).

Chapter 6 describes the first part of the solution to identified shortcomings, focused on answering RQ2. This is realized by providing the Semantic Conceptual Data Modeling Language. In addition, this chapter discusses possible language designs, provides a design description of the language, and differentiates it from existing work. This work builds upon and extends already published research focused on the analysis of numerous modeling languages (Hennig, et al., 2015), and modeling language design (Hennig, et al., 2016a).

Chapter 7 describes the second part of the solution to shortcomings in terms of a Semantic Conceptual Data Modeling Procedure that improves the procedural aspect in providing a specification to engineering data, dealing with RQ3. The described procedure forms a significant evolution of work on this subject published previously (Hennig, et al., 2016b).

Chapter 8 provides the third solution element to improving the current data management approach by providing a CDM that is produced using both language and procedure, while being aligned to previously identified system engineering needs. This chapter caters toward RQ4.

Chapter 9 evaluates the proposed improvement approach consisting of language, procedure, and CDM using a number of evaluation cases and a representative example that comes in shape of the MagSat spacecraft. The evaluation involves demonstrating concrete benefits and concludes with how these benefits positively influence system cost, system time to market, and system quality. Both chapter 8 and 9 extend significantly on prototypical research published before (Hennig, et al., 2016c).

Chapter 10 provides a conclusion of the performed work, focusing on the implication of the presented results, and outlining points for future developments.

2 System Models in Model-Based Systems Engineering

This chapter describes the nature of SE by exploring its history, a number of definitions, and a concrete realization of the approach. Subsequently, the nature of MBSE is defined emphasizing its benefits while also having a general outlook on the definition of the term model. In addition, the role of the SM in the MBSE context is elaborated.

2.1 Systems Engineering

2.1.1 Definition of Systems Engineering

The term SE has been around for quite some time, occurring in numerous areas of engineering. As such a widespread term, the view of what systems engineering does varies significantly between engineering domains, organizations, and people.

2.1.1.1 Analysis of Existing Definitions

This section examines several definitions of the term SE. While each definition describes its unique viewpoint, all definitions revolve around the same core concepts, goals, and tasks.

The International Council on Systems Engineering (INCOSE) defines SE in its Systems Engineering Handbook (INCOSE, 2015), similarly to its website (INCOSE, 2016) , with the following statement:

> "Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance,

> training and support, test, manufacturing, and disposal. Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems Engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs."

This definition already mentions important aspects of SE. It states that the core goal is to provide the means for successfully realizing systems. Furthermore, a lifecycle aspect is mentioned, going from requirements definition throughout design, up to system verification. In addition, the interdisciplinary nature of SE is emphasized, and, the customer or rather is given an important role in this definition.

The National Air and Space Administration (NASA) present the following definition in their Systems Engineering Handbook (NASA, 2007):

> "Systems engineering is the art and science of developing an operable system capable of meeting requirements within often opposed constraints. Systems engineering is a holistic, integrative discipline, wherein the contributions of structural engineers, electrical engineers, mechanism designers, power engineers, human factors engineers, and many more disciplines are evaluated and balanced, one against another, to produce a coherent whole that is not dominated by the perspective of a single discipline."

NASA emphasizes a number of further characteristics of SE. First of all, SE being a science, but also an art is mentioned. The goal of making a system meet its requirements within the given design space is emphasized, as well as the interdisciplinary nature, where different viewpoints have to be considered thoroughly, but without focusing too much on a specific perspective.

The European Space Agency (ESA) maintains a set of standards that specify the agency's view on space system design activities. Among these standards SE plays an important role and is defined using the following short but concise definition (ESA, 2009a):

> "Interdisciplinary approach governing the total technical effort required to transform a requirement into a system solution."

This definition also highlights the interdisciplinary nature of SE, as well as the lifecycle aspect.

Another definition (Friedenthal, et al., 2008) states the following:

> "Systems engineering is a multidisciplinary approach to develop balanced system solutions in response to diverse stakeholder needs. Systems engineering includes the application of both management and technical processes to achieve this balance and mitigate risks that can impact the success of the project."

This definition also picks up on the interdisciplinary nature of the approach, also highlighting the orientation towards the stakeholders' needs. Furthermore, it is emphasized that both management and technical aspects have to be considered, and that risk mitigation is an important factor in the success of the design effort.

2.1.1.2 Derivation of a Definition

Considering all of the definitions, the following characteristics are of central importance. SE

- has the successful realization of a system as the main goal,
- involves an in-depth consideration of the system user's needs for driving the system design,
- has the coordination of all involved technical and management disciplines as an important activity,
- considers and balances the influence of all involved actors on the system design towards a coherent design,
- has a strong notion of lifecycle, from system specification over design to utilization and disposal,
- involves scientific aspects as well as artistic notions, and
- works towards minimizing risks and avoiding errors.

2.1.1.3 Systems Engineering vs. System Engineering

Confusion often arises regarding the correct terminology. Frequently, Systems Engineering and System Engineering (with System in singular) are used synonymously without making any distinction. However, both terms have been defined explicitly with a difference in their meaning.

SE can be seen as a domain-agnostic approach that deals with the methods, processes, and thinking involved in successfully developing a system in any given domain of engineering.

System Engineering, in contrast, is oriented towards developing a system in a specific domain. This requires mentioning of a domain, such as Automotive System Engineer-

ing, Control System Engineering, or Space System Engineering when employing this term. System Engineering consequently requires knowledge from the generically applicable SE body of knowledge, as well as an in-depth understanding of the respective domain.

This thesis emphasizes on this distinction by utilizing the term Systems Engineering when considering generically applicable principles agnostic of any engineering domain, and the specific term when talking about (Space) System Engineering.

2.1.2 The Space System Engineering Process at Airbus DS

The Space System Engineering process as employed at Airbus Defence and Space (Airbus DS) implements the principles incorporated by the definition of SE. Figure 2.1 illustrates central building blocks of the space system engineering process.

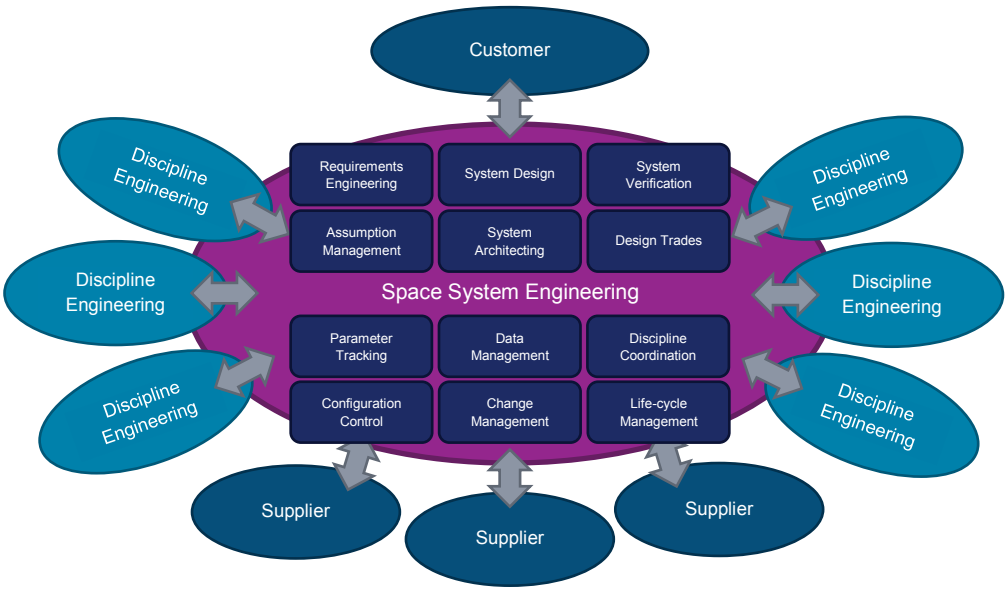


Figure 2.1: Illustration of the space system engineering process at Airbus DS

In this overall process, a variety of activities take place in the context of a specific engineering discipline, such as mechanical engineering, electrical engineering, or safety engineering. The coordination of these discipline-specific engineering activities is realized by the space system engineering process where activities such as design

trades, assumption management, parameter tracking, configuration control, and change management are performed. Usually, these disciplines and the coordinating system discipline reside within one organizational entity. However, the processes of suppliers also have to be integrated with the overall system design, as well as processes running at the system's customer. These interfaces are also realized and coordinated by space system engineering.

2.1.3 History of Systems Engineering

Although the term SE has surfaced only in the middle of the 1950s, its characteristic principles such as overcoming technical challenges, managing organizational complexity, and performing lifecycle planning, have been prevalent in many human construction efforts of larger scale (Buede, 2009). Building the pyramids of ancient Egypt involved the coordination of large numbers of people across numerous decades. The design and construction of Europe's gothic cathedrals built from the 12th to the 15th century involved solving numerous technical problems, coordinating involved disciplines, and managing the risk involved in the construction effort. Building the railroad across the United States in the 2nd half of the 19th century involved strong customer-orientation, coordinating technical as well as managerial aspects, and lifecycle management (Buede, 2009).

The first documented mention of the term SE occurred at Bell Laboratories in the 1940s (Buede, 2009), where SE then moved on to becoming an explicit organizational entity in the year 1951.

The first major project to employ SE at large scale was the development of the SM-65 Atlas Intercontinental Ballistic Missile (ICBM) in the 1950s (Hughes, 1998) that later went on to become the launch vehicle for NASA's Mercury program. Developing the Atlas booster involved coordinating 18.000 scientists, engineers, and technical experts, 70.000 people from administration and manufacturing, 200 subcontractors with 200.000 suppliers, as well as 500 military officers (Hughes, 1998), over the course of several years from initial design in 1951 to the first successful test in 1958.

The success of the SE approach led to its large-scale usage in NASA's Apollo program. SE has since been employed in industries other than aerospace and defense, such as automotive, biomedical and healthcare, infrastructure systems, and transportation systems (INCOSE, 2015).

2.2 Model-Based Systems Engineering

Over the last decades of systems engineering employment, practices and approaches have evolved continuously. A term that surfaces more and more (INCOSE, 2014) is the practice of Model-Based Systems Engineering (MBSE).

2.2.1 The Philosophy behind Modeling

The term model is nowadays used in a wide variety of contexts. Models play an important part in almost all sciences, including philosophy, economics, social sciences, psychology, biology, chemistry, mathematics, informatics, physics, and engineering (Wagner, 2014). As the name already implies, models play a central role in MBSE. This section elaborates on what characterizes and makes up a model.

An established view on models, especially in the context of informatics, was defined by Stachowiak (1973), emphasizing on three characteristic aspects that make up a model:

Mapping

A model is always a representation of something, i.e. of a physical or notional original. It is possible that this original is already a model itself. The relation of a model and the original is the mapping.

Reduction

The model does not capture every characteristic that makes up the original, but only those characteristics that are deemed relevant for the model's use case.

Pragmatics

A model replaces the original for a specific subject or subjects, in the scope of a determined timeframe, and for performing a specified set of operations.

In other words, a model is always a simplification of something that already exists. The model is built with a specific purpose in mind and consequently reduced in scope and functionality to the amount required for serving its purpose.

The term model, even when reduced to the context of engineering, is highly versatile. Models in this context can range from simple sketches to elaborate executable programs that completely represent a system under design. Examples for models include:

- a sketch of a system's shape using pen and paper,
- a rough prediction of a system's thermal behavior using a paper-based calculation,
- a calculation of a system's total weight using a spreadsheet,

- the mechanical design of a system with a Computer Aided Design (CAD) model,
- the design of a system's software using a UML model, and
- the facilities used for verifying the correctness of a system's design through simulations across a variety of system aspects.

2.2.2 Definition of Model-Based Systems Engineering

Similar to SE, a variety of definitions exists for MBSE. This paragraph picks up on a number of definitions in order to develop an understanding of the approach, also working out core concepts. INCOSE (2015) defines MBSE as follows:

> "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."

This definition contains essential components of the definition of SE, such as the life-cycle phases that SE encompasses. What it mentions in addition is the practice of formalized modeling to support SE activities.

Friedenthal, Moore, and Steiner (2008) define MBSE in the following manner:

> "Model-based systems engineering (MBSE) applies systems modeling as part of the systems engineering process [...] to support analysis, specification, design, and verification of the system being developed. This approach enhances communications, specification, and design precision, design integration, and reuse of system specification and design artefacts."

This definition mentions that the modeling of the system to be developed is a part of the overall SE process. Furthermore, the definition touches upon the motivation behind MBSE, improving communication, design precision, data reuse, etc.

A fact often cited when talking about the relation of MBSE and classical SE, e.g. by Friedenthal & Sampson (2014), and Yamaura, et al. (2016) is that

> "MBSE is SE."

This statement is usually interpreted in a way that the actual system design activities in an MBSE setting are not different from those in a classical SE setting. The difference is that they are supported by models or rather an SM, respectively, however the nature of the activities, their content, and their motivation stay the same.

Based on the examined definitions, the following things can be said about MBSE:

- MBSE involves the application of formal modeling to describe the system to be designed in a digital model.
- Models form the main interface for exchanging information between engineering disciplines involved in the system design.
- The key motivation of the MBSE approach is the support and improvement of SE activities.
- The model in MBSE supports the activities performed for coming to the system's design.

In conclusion, MBSE can be regarded as one possible implementation of the SE approach that relies on formalized modeling to describe the system in order to support SE activities.

Besides MBSE, other terms with a somewhat similar meaning have surfaced, most notably *Digital Engineering* and *Virtual Engineering* (INCOSE, 2014). All three terms describe roughly the same approach. While MBSE puts an emphasis on supporting the SE approach with models, Digital and Virtual Engineering emphasize that the whole engineering process, from specification to manufacturing, across all involved stakeholders, throughout the whole system, is supported by models. This emphasis on a complete digital/virtual/model-based consideration of a system is seen as being no different to the view defined by MBSE. Consequently all three terms are treated as being equivalent for the context of this thesis.

2.2.3 Envisioned Benefits of Model-Based Approaches

Literature on MBSE elaborates further on the benefits that come with a more model-based consideration of SE.

Improved communication among development stakeholders

Friedenthal, et al. (2008), INOCSE (2015), and Delicado (2016) state that the benefits gained from using an MBSE approach include improved communication among the stakeholders involved in the development by providing a shared core understanding of the system. A better understanding is also gained by providing the ability to regard the system using different views for specific purposes.

Improved product quality

Another motivation behind MBSE is improving the product quality (Delicado, 2016) by enabling checking of completeness, correctness, and consistency of the SM (INCOSE, 2015; Yamaura, et al., 2016), and by providing better traceability behind requirements and system design (Friedenthal, et al., 2008).

Increased productivity

Having a model-based representation of the system enables new functionality, such as the possibility to automatically perform impact analyses, and managing system complexity more efficiently (INCOSE, 2015; Delicado, 2016; Friedenthal, et al., 2008). Furthermore, a better data integration process is enabled, as well as the ability to generate documents directly from the system model, instead of having to write them manually (Friedenthal, et al., 2008).

Enhanced knowledge capture, transfer, and reuse

INCOSE (2015) states that MBSE provides better means to capture and reuse knowledge by providing standardized models. Friedenthal, et al. (2008) claim that the ability to formulate queries on the system's design enables better knowledge transfer, which is also a point emphasized by Delicado (2016).

Reduced development risk

Another motivation behind MBSE is the ability to perform system verification and validation earlier, and continuously over the whole system design cycle. Furthermore, by having a better system-wide data representation, the ability to provide solid cost estimates is improved. (Friedenthal, et al., 2008; Yamaura, et al., 2016)

2.3 The System Model

A prerequisite for performing MBSE is to effectively and efficiently exchange data between engineering disciplines involved in a system's design. The efficiency and effectivity required by this data exchange implies a model-based exchange interface.

While engineering tools inside specific engineering domains are frequently connected via defined data exchange interfaces, tools of different engineering domains are not yet connected to each other on large scale (INCOSE, 2014). Interfacing tools of different domains is a considerably larger challenge due to engineering tools being supplied by different vendors, relying on different technologies, being based on different modeling paradigms, and exhibiting different internal semantics (Kogalovsky & Kalinichenko, 2009).

One possible approach to exchange data between these heterogeneous engineering tools is to use an SM as a central data exchange hub, providing interfaces towards different engineering tools. This approach to MBSE is pursued in certain areas of the space domain (ESA, 2011a) and will serve as reference for the remainder of this thesis.

In this architecture, the core purpose of the SM is to store and manage the data that makes up the definition of a system. Besides this, the SM provides a number of important functions, scopes the data that makes up the system, and is the main location

for performing systems engineering activities, forming a primary artefact of the (MB)SE process (INCOSE, 2015).

Figure 2.2 shows the SM in its context. The SM stands between a range of other models pertaining to a specific engineering discipline. Data exchange is occurring between these disciplines or rather their models via defined interfaces, making the SM the central data exchange hub. For example, mechanical design data from a *CAD model produced with CATIA* (Dassault Systèmes, 2017) holds information about the mass of mechanical parts of a system. This mass data is an input required to producing the system's mass budget, managed by the discipline of *System Engineering*. The overall mass data then again serves as input to the discipline of *Verification Engineering*, as it is compared to *requirements modeled in DOORS* (IBM, 2017). Other data exchanges include a flow of orbit data managed by the *Mission Design* discipline to the discipline of *Simulator Engineering*, where it is used in a *MATLAB-based* (MathWorks, 2017) *orbit model*, and the flow of component power consumptions in different operational scenarios data from *Simulator Engineering* to *Systems Engineering*, serving as input to the *Power Budget*.

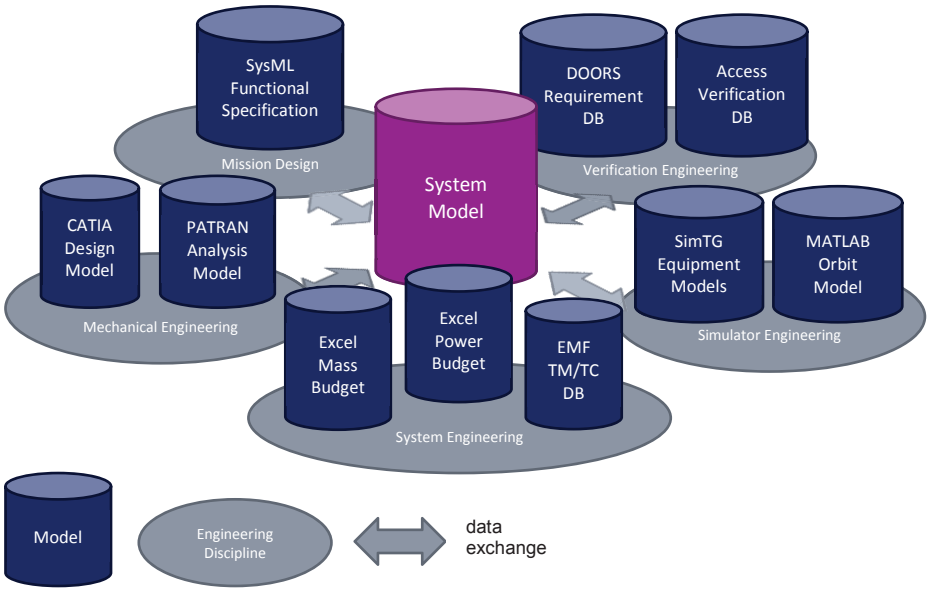


Figure 2.2: System Model and selected discipline-specific engineering tools

2.3.1 Content of the System Model

For the case of space system engineering at Airbus DS, the following data is scoped by the SM (Eisenmann & Cazenave, 2014). It is driven by the needs of the SE process of a specific domain. In general, three categories of data can be defined as being the SM's content:

Key system definition data

This data describes key aspects of the system, such as its hierarchical decomposition, configuration items, or information of the system's purpose and context.

Data required for performing systems engineering activities

This includes data of significant importance scoped across the whole system, such as the overall mass budget and overall margin considerations. Furthermore, this includes process-relevant aspects such as the tracking of assumptions across the system's design.

Data being exchanged across discipline and stakeholder borders

This includes data such as interface specifications, operational aspects, or component data, that is not only utilized inside a single discipline, but required as input for a number of engineering activities in different disciplines. Data specific to a discipline, such as information about the meshing used in the mechanical analysis of a component, is not scoped by the SM.

More specifically, this data consists of the following building blocks in the space engineering domain. The data is scoped across all decomposition levels of the system and throughout the whole system lifecycle (ESA, 2011):

Product Structure

The Product Structure (PS) describes a system's hierarchical decomposition. This includes the description of the subsystems, their components, and involved parts that make up the system. Furthermore the PS usually distinguishes between a systems element's allocation in the system's lifecycle. This means that elements are considered separately at different points in their lifecycle, distinguishing between elements *as specified*, *as configured*, and *as built*.

System Key Parameters

These parameters define essential characteristics of a system. In the case of space engineering, these include a system's orbit altitude, orbit lifetime, propellant mass, etc. This also includes a budgeting of specific properties throughput the whole system, such as the system's mass budget, power budget, link budget, and memory budget, along with parameter margins.

Requirements

This includes the specification of a system where requirements are formulated and traced to building blocks of the system.

Functional Design

This includes the representation of functions that are performed by the system.

Operational Design

This building block represents the consideration of the system from an operational perspective, including concepts such as system modes, operational activities, and on-board control procedures.

Topological Design

This data is used to describe a system in terms of its ports and its interfaces.

Verification data

Verification data represents the verification activities performed on a system. This includes a specification of how requirements are to be closed, e.g. by analysis, test, review, or inspection, and the management of these activities.

Component design data

Data received from component suppliers, containing descriptions of component interfaces, component specifications, properties, and so forth.

Monitoring and Control data

Description of system telemetry and telecommands in terms of packets, parameters, calibration values, etc.

Assembly, Integration, and Test data

Information regarding the system's production and testing process, including how to integrate components, what tests to be performed, and test execution data.

2.3.2 Functions of the System Model

The SM in the MBSE context requires a number of functions that support SE activities. Since a model usually does not directly exhibit or perform a function, these functions are realized by the application that is used to model and store the SM. These functions include (Eisenmann & Cazenave, 2014):

Consistency checking

Consistency checking ensures the consistency of modeled data. This can range from simple checks assuring correct cardinality of attributes, to more elaborate queries on

data that is regarded as important for determining if the representation of the system is accurate.

Data comparison

This can mean comparing system properties between different elements of the system, as well as comparing the same property of the same system element between different system revisions. This is used to, for example, evaluate how a component's mass changed within a given timeframe, or to compare two identical components regarding their geometric position in the system.

Data query

Used for extraction of data from the system model, such as extracting the mass values for each component inside the system.

Model migration

As the data specification for a system may change throughout its design, migration steps might become necessary. If the definition of a concept used for describing the system changes, migration has to be performed.

Reporting

The ability to generate reports from data stored in the SM. This is used to, for example, extract descriptions of components from the SM, containing their requirements, interfaces, modes, and characteristic physical properties.

Branching and merging

The ability to branch the system's design or parts of it and to later integrate data refined in the branches.

Configuration control

The configuration control functions represent the capability for baselining, versioning and releasing a system's design, represented by its SM.

Model visualization

Providing some sort of visualization for the SM. This can range from basic user interface concepts such as such as forms, trees or tables, over diagrams up to elaborate visual queries.

Data input and output

This function provides facilities necessary for realizing input of data to the SM, and output of data from the SM to another model. These inputs and outputs may take numerous forms and purposes, supporting the design of specific system aspects, performing analyses on system data, or overall system verification in dedicated verification models.

2.3.3 Interfaces of the System Model

For performing model-based exchange of engineering data as required for MBSE, the system model has to provide a number of interfaces for exchanging data with engineering domains within the organization, and with customers and suppliers outside the own organization (Eisenmann & Cazenave, 2014). SM interfaces of the space domain, along with their main data flow direction, are outlined in Figure 2.3.

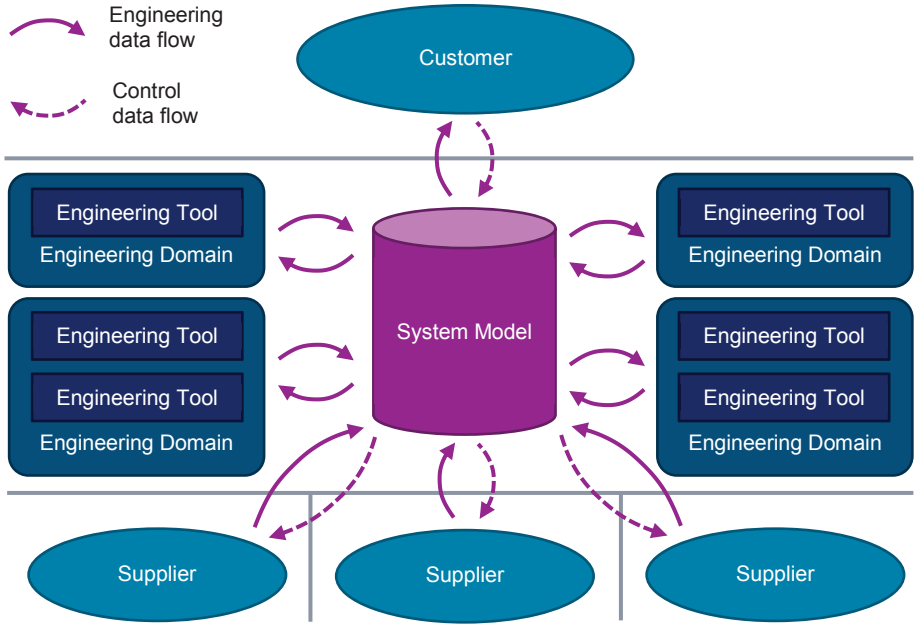


Figure 2.3: Data exchange interfaces of the System Model

2.3.4 Specification of System Model with a CDM

A prerequisite to storing information about the system in the SM is the specification of the generic concepts that make up the system, i.e. the definition of the data that is used to describe the system. This implies that the data stored in the SM has to adhere to the conceptual structure that is defined in a model one abstraction level above the SM, being its meta-model.

It is worthy to note that the term meta-model is a relative term and always refers to a model one level of abstraction above the current model of interest. The term is often

extended by adding extra *metas*, resulting in e.g. a *meta-meta-model*, describing a model two levels above the considered model.

In the context at hand, the meta-model to the SM is often called Conceptual Data Model (CDM) and defines the semantics of the concepts that make up the system to be designed. More specifically, the CDM defines the concepts of significance to developing a specific kind of system, the characteristics of concepts, and the relations between them (ESA, 2011a), (Halpin & Morgan, 2008). This generic relation is shown in Figure 2.4.

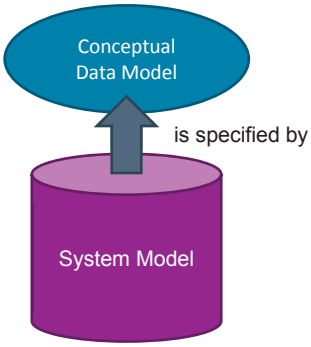


Figure 2.4: Relation of System Model and Conceptual Data Model.

The view outlined above is a rather simplistic consideration, as the relationship between SM and CDM is usually not direct. The CDM is meant to be independent of any implementation technology, focusing purely on the conceptual properties and semantics of the concepts defining a system. To come to an implementation of the CDM, two further models are frequently employed. This includes a Logical Data Model (LDM) that is developed based on the CDM. It represents the CDM in a data modeling approach that will determine the subsequent implementation, such as if it will be based on relational or object-oriented technologies. From the LDM, the Technical Data Model (TDM) is derived and defines the detailed implementation of the SM, supplying detailed implementation-focused characteristics such as technical identification schemes, parameters for code generation, and mappings to language-specific data types. Some approaches also include the code-level in this architecture, however this is left out in the description of Figure 2.5, due to it usually being a non-abstracted element. In some places, the TDM is also called Physical Data Model. (ESA, 2011a). For example, a CDM could be defined in UML (OMG, 2015b), having an LDM and its TDM in Ecore (The Eclipse Foundation, 2016c), which is then implemented in a Java application containing the code allowing the instantiation of an SM.

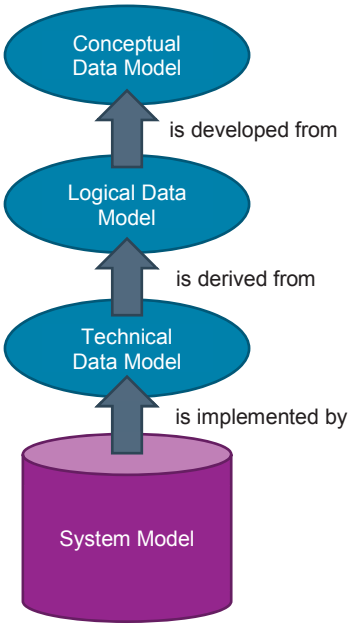


Figure 2.5: Relation of System Model to TDM, LDM, and CDM

Using this approach allows the specification of data in an implementation-agnostic way, and to implement the system model using different technologies, allowing exchange of data specified using a specific CDM between different organizations using different technology landscapes.

The concept of conceptual modeling to facilitate this kind of exchange is not new and has been around for a number of decades. The concept was made prominent by the interim report of the ANSI/X3/SPARC Study Group on Data Base Management Systems (1975), under the name of *conceptual schema*. The LDM can be mapped to the external schema, while the TDM fits closely to the internal schema described in the report. The Technical Report on Information Processing Systems (ISO, 1987) emphasizes the role of the conceptual schema as a central driver for the SM under the term of *information base*.

2.4 Modeling Engineering Data in Space System Design

Modeling systems in the domain of space engineering exhibits a number of characteristic aspects. This section details several of these characteristics.

2.4.1 Data Specification Approach

In order to enable the modeling of engineering data in the SM, the data has to be defined in an abstract way on CDM level. This specification is usually produced by modeling experts in collaboration with experts from the engineering domain that is the main stakeholder for the data.

When modeling a specific set of data, different modelers tend to produce different models for the same set of data (Leung & Nijssen, 1998). While representing the same core data, the models may well differ significantly in their underlying principles and structure. This heterogeneity can even occur inside one model, where different modelers produced different parts of one model. Furthermore, ad-hoc definition of CDMs often leads to discussion on the correct way to model a specific set of data with discipline experts, resulting in numerous iterations.

In addition, these CDMs usually do not go through a dedicated validation procedure where the CDM is validated before it is implemented in the engineering application. In the event of forgotten model elements, significant parts of the software design cycle are usually repeated in order to include the additional concepts in the CDM.

2.4.2 Tailoring

In space engineering, the practice of tailoring is often pursued (ESA, 2009a). In this context, tailoring involves taking a standard and adapting it to the project's exact needs. This can mean explicitly excluding parts of the standard, or defining new parts of the standard that support project-specific requirements. There are numerous cases where this approach is employed, most prominently for adapting the design of the monitoring and control services running on board of a spacecraft using the Packet Utilization Standard (PUS) (ESA, 2003; ESA, 2008d). Other data structures also vary from project to project, e.g. the definition of physical properties on system components, or usage of electrical interfaces aboard a spacecraft. As a number of data structures may vary from project to project, a deployment of an engineering application implementing a CDM specific to every project is not seen as feasible, leading to the usage of dynamic structures that can be adapted for each project during runtime.

2.4.3 Semantic Softening of CDM in Implementation

Although a CDM might be semantically well defined, its interpretation and realization during the implementation process might lead to a loss of semantics (ESA, 2012a). In practice, pragmatic approaches are pursued in order to effectively and quickly deploy an engineering application based on a specific CDM, sacrificing ease of implementation for accurate semantics in the process. One approach to this is the employment of generic structures that are applicable to a wide number of system model populations, but also allow populations that are logically inconsistent.

2.4.4 Relation of SM to Product Lifecycle Management

The SM takes a place between two levels of data abstraction. On the most detailed level are the disciplines that manage their data in the most granular, detailed manner. Releases of this data may be produced on a daily basis or even several times per day for bug fixing and rapid exchange. The SM interfaces with the disciplines by accommodating selected chunks of data that is coming from one discipline, and is needed by one or several other disciplines. Releases of the SM are usually performed in the timeframe of several days to weeks, providing new data input to disciplines.

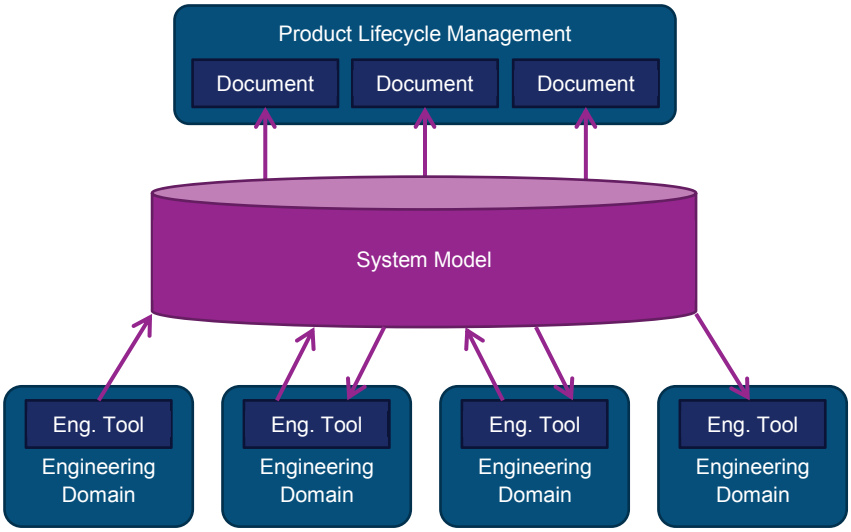


Figure 2.6: System model as bridge between domains and PLM

The level with the highest degree of abstraction is formed by the layer of Product Lifecycle Management (PLM), where data is consolidated towards specific milestones of the project/system using a document-based approach (ESA, 2009a). These milestones are usually several months apart, resulting in new releases of data in this timeframe. The system model assumes a role where it provides a bridge between data coming from disciplines and data going towards the PLM level (Figure 2.6). This data exchange is usually defined by the underlying engineering process, and consequently can be made explicit on CDM level, providing traceability of the data flows occurring across the SM.

2.4.5 Relevance of Constraints and Consistency Checks

The specification of data in the CDM usually involves a concise definition of allowed and not allowed populations. For this purpose, constraints are defined between concepts of the CDM. Such constraints involve, for example, subset constraints between references, object cardinality constraints on classes, or value constraints between attribute values.

These consistency checks are used to check basic model values, such as that the given minimum operating temperature of a component is lower than the value given for the maximum operating temperature. Similarly, the maximum non-operating temperature has to be above the minimum non-operating temperature, and the non-operating envelope has to be identical or greater than the operating envelope. Figure 2.7 illustrates this example, showing the logical dependencies between different temperature properties of a given component.

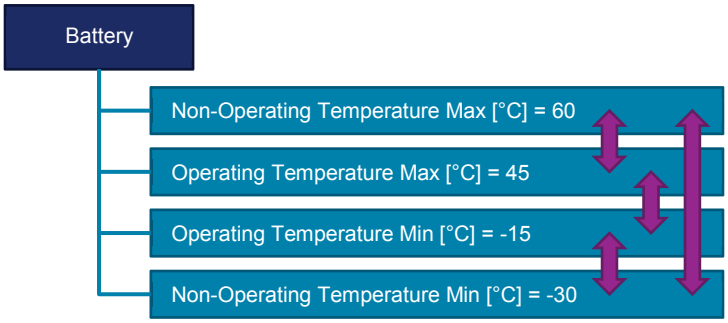


Figure 2.7: Examples for basic model consistency

2.4.6 Relevance of Closed World Behavior

Many consistency checks in the MBSE process involve examining whether required data is present in the model or not. If the data is not yet entered into the model, a check should highlight this fact. For this, it is required that the SM behaves as if it were a closed world, where all of the information possibly known to the model is contained by it. This Closed World Assumption (CWA) acts in a way that data not present is treated as false, forming the usual modus operandi for object-oriented models and applications. Its counterpart, the Open World Assumption (OWA) treats data not present as unknown, resulting in different behavior (Allemang & Hendler, 2011). A more in-depth explanation of OWA and CWA is given later in 3.6.3.

2.4.7 Functional Dependencies between Model Elements

A high number of dependencies exist between concepts specified in the CDM. For example, many engineering processes distinguish between items *as specified*, items *as designed*, and items *as built*. These concepts exhibit numerous functional dependencies between each other. For example, a system component that has an *On*, *Standby* and *Off* state with the power consumption defined on specification level should exhibit the same states on configuration level with an identical power consumption. The component *as built* has to exhibit the same states, however the power consumption can now be measured and may differ slightly from the specified value. These generic functional dependencies occur between concepts of the CDM.

2.4.8 Multitude of Element Characterization Mechanisms

The characterization of elements of a system in the MBSE process involves a number of different approaches that are used in defining their type. For example, the following statements could be true about an element of the system (Figure 2.8):

- The element in question is an element *as specified*.
- The element resides on the system level of *component*.
- The element is a *hardware element*, more specifically a *Star Tracker Electronics* unit.
- The element's operation is defined as *internally redundant*.
- The element has *electrical ports*.
- The element has *thermal design* considerations.
- The element is subject of specification through requirements and consequently also subject of verification.

These statements together form the characterization of an element in the system. They are independent from each other and, for other combinations of statements, would imply different semantics regarding the nature of the element.

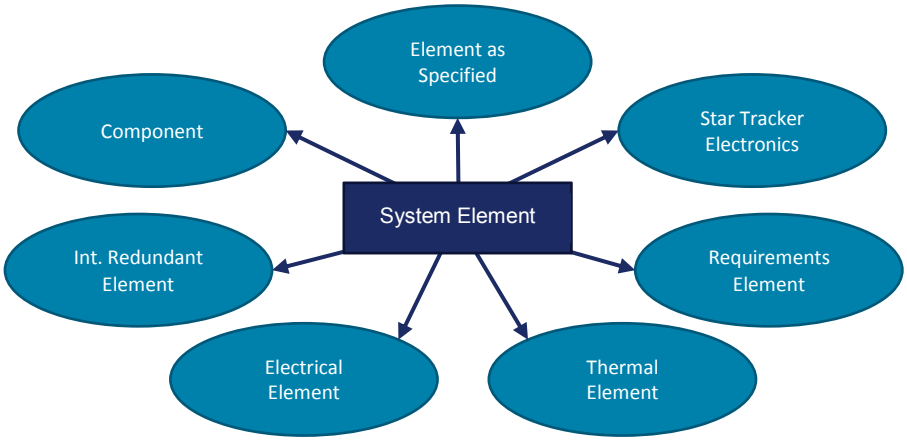


Figure 2.8: Example facets of a System Element

2.4.9 Lifecycle Aspects of Engineering Data

As space engineering has a strong notion of lifecycle, the same can be said about the data involved in this process. Usually, data evolves from a basic description towards a very detailed, fine-grained description later on. This is true for a number of aspects of space systems.

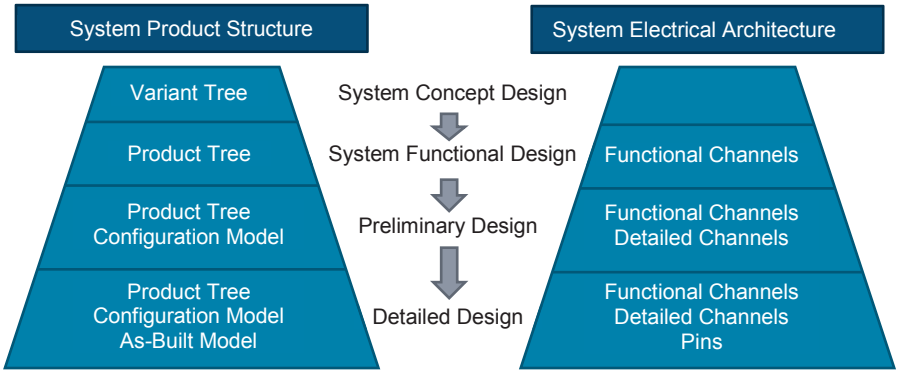


Figure 2.9: Product Structure and electrical architecture lifecycle aspects

For instance, in the beginning of a project, the description of a spacecraft should only be done using a product tree containing elements *as specified*, having no consideration of elements *as built*. The electrical architecture of a spacecraft evolves from a plain functional over a more elaborate view merely involving signals allocated to functional channels, towards a fully-fledged definition including all electrical properties of the channels, as wells as the mechanical properties of the spacecraft harness's pins. A spacecraft's testing campaign should be roughly defined at the end of the detailed design phase, but gets more elaborated during system production. This is shown in Figure 2.9.

2.4.10 High Relevance of System Execution Data

System verification and validation through simulation or other kinds of execution of the system takes an important place in the MBSE context (Fischer, et al., 2014). Using such execution data, information about a system's design correctness or production correctness can be derived. Evaluating this kind of data is not always straightforward, but involves significant knowledge about the system under test. For example, after each performed test of a satellite, accumulated data during this test has to be evaluated in order to determine if the test was a success or not. Although some condensing of the data is performed by using thrown events during the test as an anchor point for data evaluation, the data is somewhat extensive. For instance, the Abbreviated Functional Test (AFT) produces around 90.000 lines of log with around 250 events. Although a significant number of events marked critical occur during one test, the actual criticality of the event has to be examined by evaluating in which context it was thrown. Concluding on all of these events and determining if they are expected or unexpected is an important task that can become quite extensive, as a satellite usually undergoes several thousands of test sessions across its whole verification cycle.

2.4.11 Manual Application of Implicit Operational Knowledge

Engineering a system of significant complexity usually involves a lot of expertise and engineering experience. Many activities and decisions are significantly driven and influenced by the knowledge of the involved engineers. Very often, this knowledge is not made explicit by formalization, but is rather available implicitly in the form of personal engineering experience and expertise. These experts and consequently their knowledge come from a variety of different domains, being very heterogeneous, and pertaining to a specific view on the system. This implicit knowledge is then applied in a manual engineering process. For example, for identifying critical elements in the system's design, the process detailed in Figure 2.10 can be pursued.

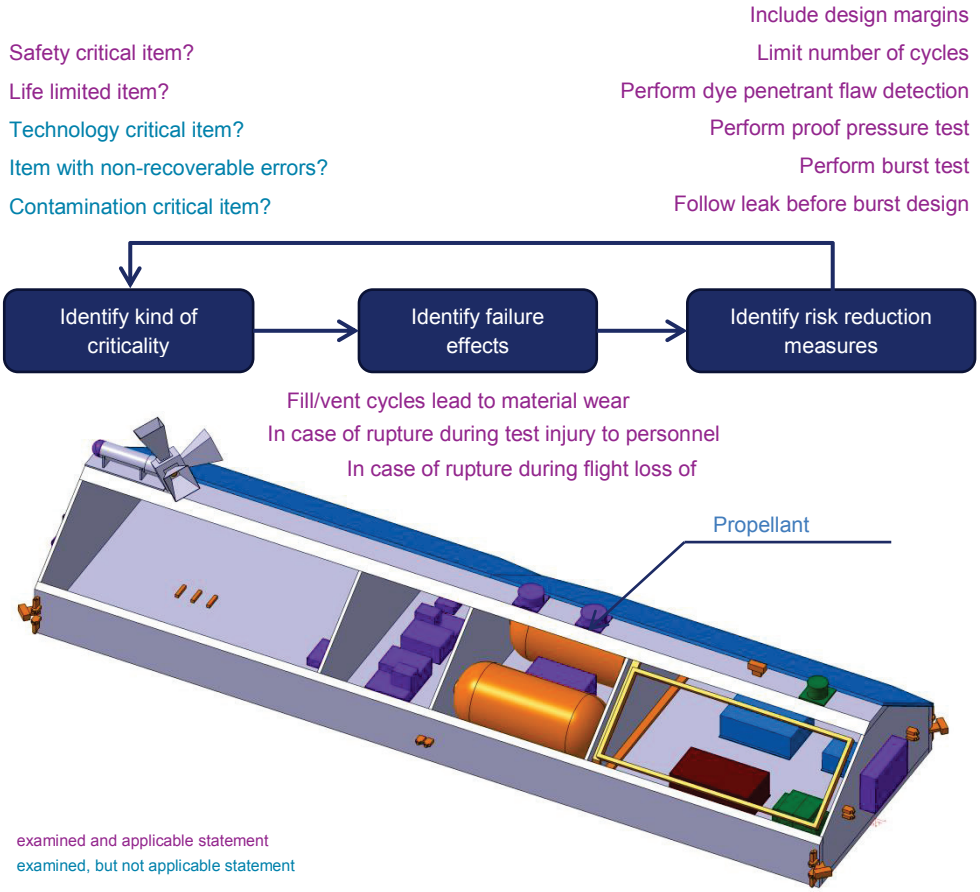


Figure 2.10: Approach for Identifying Critical System Elements

In this manual engineering process, each system component is considered regarding its criticality, determining if it falls into one of several known criticality categories. If a criticality is identified, possible failure effects are derived, and, based on these effects, risk reduction measures are defined. This process is repeated for each system component, and updated continuously throughout the system's design cycle, as newly surfaced information might have an impact on the analysis outcome.

3 Background

The layering of data models bears close ties to essential software engineering concepts such as the Object Management Group's (OMG) Model-Driven Architecture (MDA), and Meta-Object Facility (MOF). Furthermore, data models are often described using specification languages such as the Unified Modeling Language (UML), or Ecore, that have since been established in the software engineering community. Furthermore, the Web Ontology Language OWL 2 plays an important role.

This chapter elaborates on these concepts by providing a brief description. It can be skipped if the reader already has some familiarity with the mentioned concepts. Furthermore, this chapter details the foundations of the demonstration scenario that comes in shape of the MagSat spacecraft, which is used for demonstration and validation throughout this thesis.

3.1 Model-Driven Architecture

The MDA is a design approach launched by the OMG in 2001 (OMG, 2014a), originally intended to aid in the design of software.

The philosophy behind the approach lies in producing a specification of the system to be designed in terms of a number of models for better understanding and communicating about the system. An important aspect of the approach is the notion that one model is not able to capture all relevant information of a system of significant size effectively, leading to the definition of several models, each on one of the system's characteristic abstraction levels. This specification begins with a description of the system oriented towards human readability and understandability, and ends at source code. An important part is played by model transformations, partly or completely performing the transition of the specification from one abstraction level to another. The MDA considers a total of four model levels (Figure 3.1):

Computation Independent Model (CIM)

This model forms the most abstract and business-oriented view of the system It may involve a description of the system in its relevant context, or a model of the business process.

Platform Independent Model (PIM)

The PIM defines the structure of the (software) system and its behavior and is required to be independent of any implementation technology.

Platform Specific Model (PSM)

This model is made up of a platform-specific representation of the PIM, or of an extension to the PIM with platform-specific aspects. It is required to be able to be translated into application code through some sort of transformation.

Code Model

Forming the bottom level of the MDA is the code model, consisting of actual source code for the application.

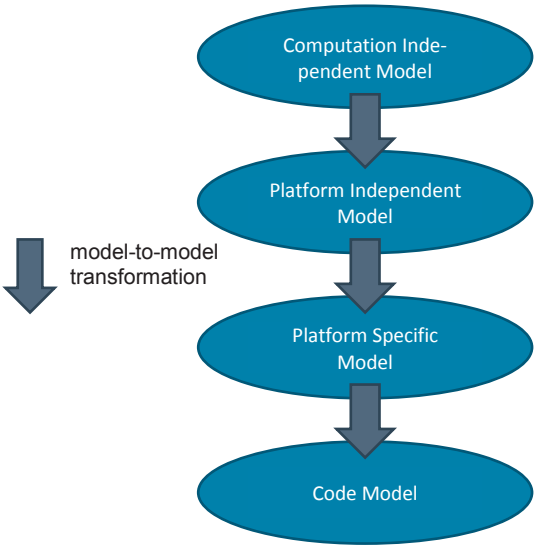


Figure 3.1: MDA Levels and Possible In-Between Model Transformations

3.2 The Unified Modeling Language UML

The UML is also maintained by the OMG and has been established as de-facto standard in describing software systems. This description is performed mainly graphically, using a set of diagrams with different application cases (Figure 3.2). A key element in this approach is that software models represented in UML are meant to form a description independent of any implementation technology. (OMG, 2015b)

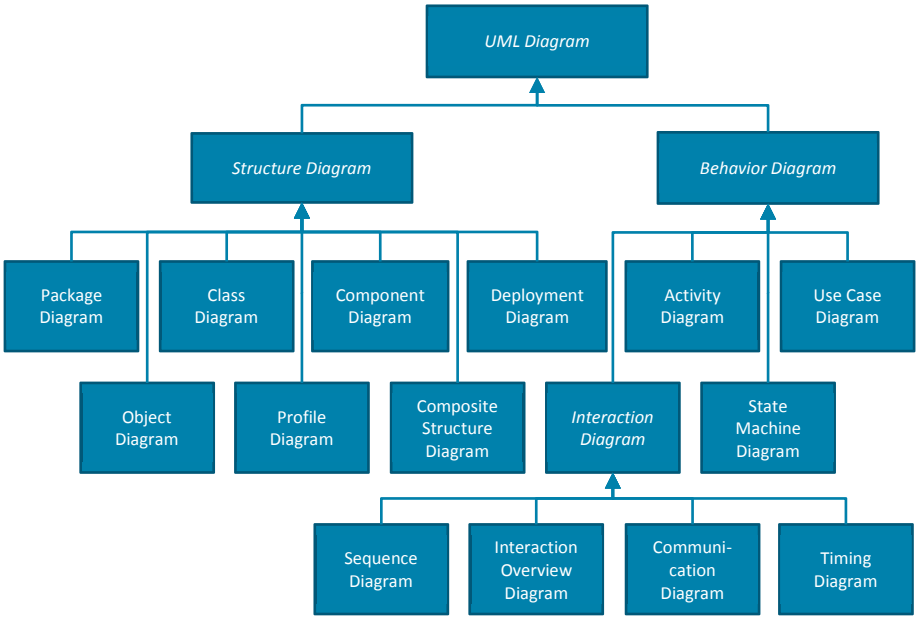


Figure 3.2: Overview on UML Diagram Types (OMG, 2015b)

UML scopes a number of different diagrams that are categorized into structure diagrams, behavior diagrams, and interaction diagrams (OMG, 2015b). The most essential ones are:

Use Case Diagram

This diagram provides a high-level functional view on the software in terms of actors and use cases on the system.

Class Diagram

The class diagram forms the backbone of any UML model by providing a description of classes, their inheritance hierarchy, attributes, operations, and relationships to other classes.

Component Diagram

The component diagram is used to illustrate a software's partitioning in terms of component structure. It describes component hierarchies and how components interact with each other in terms of interfaces.

Activity Diagram

This diagram describes activities that are being performed on the system using decision nodes, events, and other typical means of modeling control flow.

State Machine Diagram

The system's behavior in terms of its states and transitions between them is described in this diagram

Sequence Diagram

The sequence diagram expresses the involvement of objects in specific activities with an emphasis on the order in which they exchange messages.

UML offers the possibility to extend the language by defining stereotypes. These stereotypes can be applied to UML language elements, introducing custom semantics. Stereotypes are usually contained by and applied through a profile. (OMG, 2015b)

The concepts in different kinds of UML diagrams correspond to different levels of MOF. For instance, use cases and activity diagrams form CIMs of a system, whereas state machines and class diagrams, already being quite formal, represent PIMs.

3.3 Meta-Object Facility

The Meta-Object Facility is also a model-driven engineering standard maintained by the OMG and forms an integral part of the MDA. The concepts defined in MOF form a platform-independent framework for managing meta-data of a system designed to enable design interoperability of data-driven systems (OMG, 2015a).

MOF describes a layered architecture of PIMs and utilizes the concept of Classes to describe its main building blocks. These classes relate to classes on other MOF abstraction levels via a classifier-instance or class-object relationship (OMG, 2015a). This essential relationship is outlined in Figure 3.3

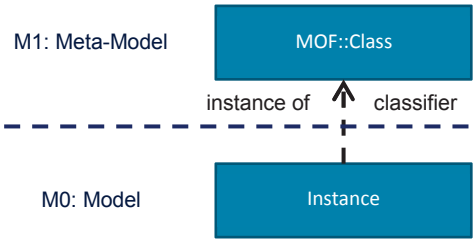


Figure 3.3: Instance Relationship between Classes on Different MOF Levels

Traditionally, MOF is regarded as four-layered, as the most prominent usage of MOF is the description of the levels involved in producing a UML-based architecture of a software system. These characteristic levels are outlined in Figure 3.4.

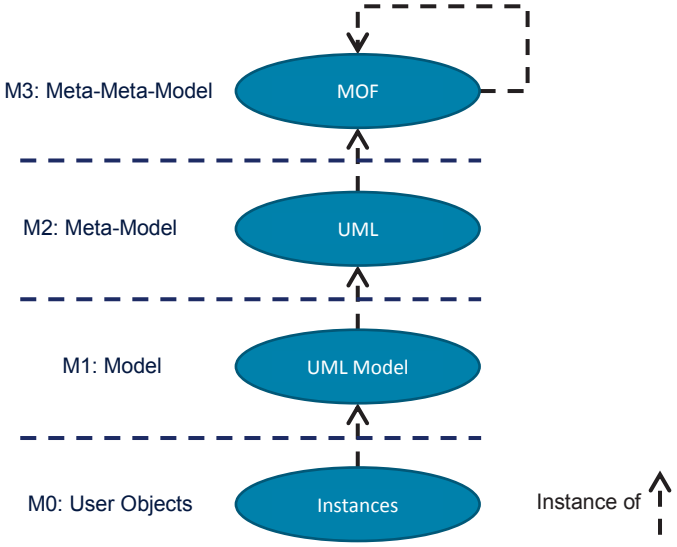


Figure 3.4: Placement of UML models inside MOF

M3 - Meta-Meta-Model

The level at the top is made up of MOF itself. The concepts on M3 are instances of the concepts also residing on M3, due to the fact that the concepts making up MOF are specified by using MOF.

M2 - Meta-Model

In this case, the meta-model is UML, made up of instances of concepts defined in MOF. This means that a *UML::Class* is an instance of *MOF::Class*.

M1 – Model

The model on M1 is characterized through instantiating the concepts of UML. For example, a concept of *Spacecraft* would be an instance of *UML::Class*, and the name of the *Spacecraft* would be an instance of the concept *UML::Attribute* with type *String*.

M0 – Instances

The bottom level is made up of the instances of concepts defined on M1. For example, an instance of *Spacecraft* might exist on this level that has the name of *MagSat*.

An essential principle in MOF is the necessity that every element residing on any level has to conform to an element on the level above. For the top level that in this context is always represented by MOF, this principle is enforced by enabling the description of MOF by itself. A wide-spread misconception is that MOF is always made up of four levels. While this is true for positioning UML inside MOF, any architecture is conceivable that employs a number of levels different from four. The minimum requirement for forming a MOF-compliant architecture is two levels. (OMG, 2015a)

Although both MDA and MOF (in the UML sense) consist of four layers, the layers used in MOF are not to be confused with the layers defined by the MDA (OMG, 2015a). The MDA layers represent different levels of abstraction for a specific model or set of models. Essentially, in the UML example, the M1 level of the MOF would be represented on the computation independent and platform independent levels. The M1 level would also have a representation on the platform-specific and code level, apart from its UML model. Each of these models on the abstraction levels forms a specific viewpoint on the system.

MOF defines two compliance points, Essential MOF (EMOF) and Complete MOF (CMOF). EMOF forms a compact model meant to provide a low barrier for making implementations based on object-oriented programming languages compliant to MOF. CMOF offers more elaborate language elements and modeling capabilities and fully includes EMOF. (OMG, 2015a)

3.4 The Systems Modeling Language SysML

The Systems Modeling Language (SysML) also lies within the responsibility of OMG and is a language highly similar to UML, but aimed at describing any kind of system, not just software. SysML uses a subset of UML and extends it with additional elements tailored for engineering systems. (OMG, 2015c)

This extension is realized by extending UML's *StandardProfile* using UML stereotypes, diagram extensions, and model libraries. Figure 3.5 shows the diagram types of SysML, together with their relation to UML 2 diagram types.

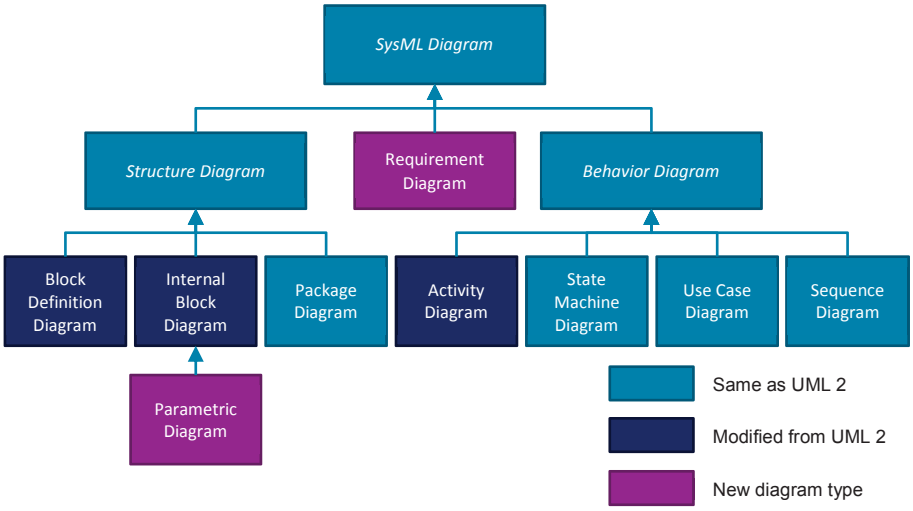


Figure 3.5: Overview on SysML diagram types (OMG, 2015c)

When contrasted to UML, some diagrams are identical, some are modified, others are extended, while yet others are excluded. The most significant differences occur in the following diagrams:

Requirement Diagram

This diagram is included for enabling the specification of requirements, their relation to other requirements, tracings to system building blocks, and describing test cases.

Block Definition Diagram

Adapted from UML's class diagram, this diagram is employed to describe a system's decomposition into different parts on different abstraction levels.

Internal Block Diagram

This diagram is adapted from UML's composite structure diagram and used to describe a system's internal connections with ports and interfaces.

Activity Diagram

This diagram extends the activity diagram from UML to also include objects that serve as input to or output from different activities.

Parametric Diagram

This diagram was newly introduced in order to specify relations between system parameters.

An important aspect of SysML is defined by one of its non-normative extensions, a model library for Quantities, Units, Dimensions, and Values (QUDV). This library defines a model for physical quantities, how they relate to units and systems of units, how different units have to be related to each other, how to interpret the semantics of unit prefixes, and so forth. (OMG, 2015c)

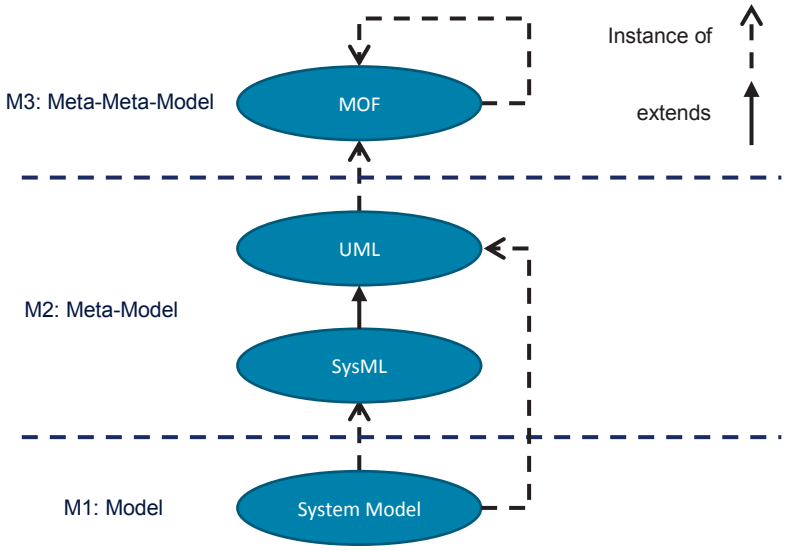


Figure 3.6: Placement of SysML models inside MOF (OMG, 2015c)

Figure 3.6 relates SysML models to SysML itself and UML throughout MOF's meta-levels. An interesting aspect in this case is that level M0 is missing, as SysML-based SMs are not instantiated in the object-oriented sense.

3.5 Ecore

The Eclipse Modeling Framework (EMF) (The Eclipse Foundation, 2016b) forms a series of extensions the Eclipse IDE (The Eclipse Foundation, 2016a) focused on model-driven engineering. For specifying EMF-compliant models, a language called Ecore (The Eclipse Foundation, 2016c) was defined.

Ecore forms the means to specify packages, classes, attributes, references, and other structural modeling elements in an EMOF-compliant architecture. For this purpose, a number of visualizations have been defined, most notably the Ecore diagram, inspired by UML class diagrams. Besides this diagram-based concrete syntax of Ecore, an abstract syntax is also provided and can principally be used on its own to specify Ecore-based models without the need for diagrams.

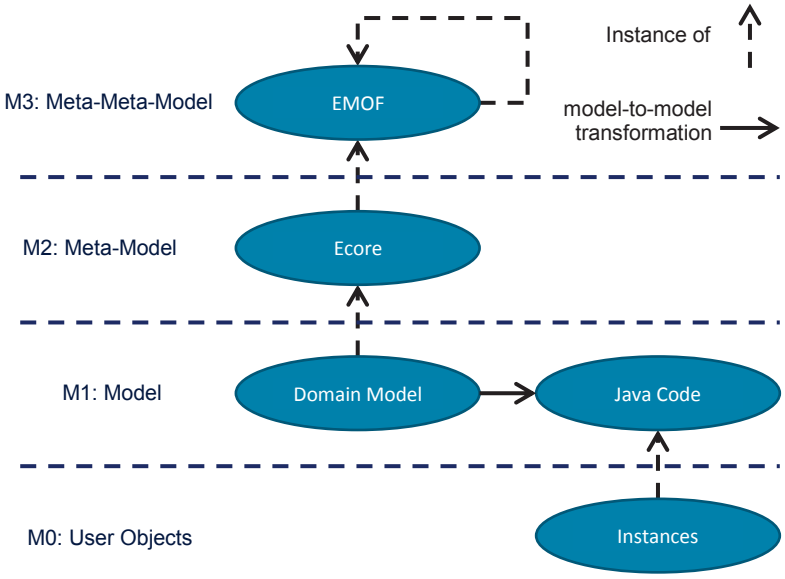


Figure 3.7: Situation of Ecore and Ecore models inside the MOF architecture

The semantics of Ecore language concepts are clearly specified in scope of EMF, as Ecore-based models are used to generate Java code, relying on a number of software engineering patterns. The Ecore language resides on the M2 level of the MOF architecture, and Ecore-based models reside on level M1 (Figure 3.7). Depending on the viewpoint, Ecore models can be seen as either platform-specific, as they are tied to the EMF environment and to Java as implementation language, or as platform-independent, as they also can be serialized into XMI.

Figure 3.8 shows the building blocks of the Ecore language in the former standard Ecore notation. The language structure conforms closely to the concepts defined by EMOF. These include, for example, the *EClass*, being the Ecore-based implementation of the *MOF::Class* concept.

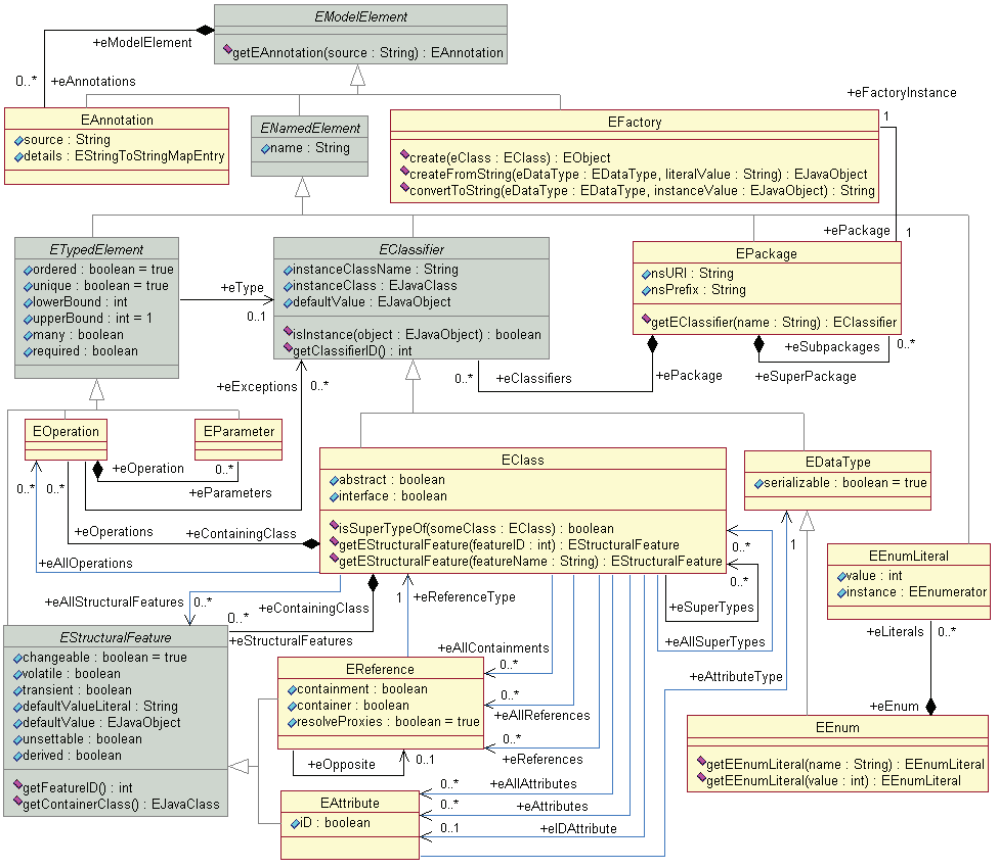


Figure 3.8: Ecore meta-model diagram (The Eclipse Foundation, 2016c)

An Ecore model can partitioned into a number of *EPackages*, which can form a hierarchy using the *eSubpackages* containment reference. *EPackages* may contain *EClassifiers*, more specifically *EClasses* and *EDataTypes*. *EClasses* can be refined by exhibiting any number of *EStructuralFeatures*, i.e. *EReferences* and *EAttributes*. These *EStructuralFeatures* all have to be *typed* by exactly one *EClassifier*, and exhibit both a *lowerBound* and *upperBound* in terms of an integer designating the maximum number of concurrent values. *EReferences* can be refined further optionally by stating that

they form an explicit composite relationship using the *containment* attribute, and by stating that they form an *opposite* relation to another *EReference*.

3.6 Ontologies and the Web Ontology Language OWL 2

Ontologies have gained considerable importance in the areas of knowledge engineering and specification over the last years (Studer, et al., 1998; Gómez-Pérez, et al., 2004). One representative from this group is currently of particular importance.

3.6.1 Definition of Ontology and Ontologies

Originally coming from Philosophy, the principle of ontology forms a philosophical discipline, dealing with "the study of the categories of things that exist or may exist in some domain" (Gašević, et al., 2009). In English, the Greek *ontos* stands for *being* while *logos* means *study* (Gómez-Pérez, et al., 2004). Thus, Ontology can be translated to "the study of being".

The word ontology has been given a more practical meaning in recent years, as it has found its place in the field of Informatics. There, ontologies are used to describe knowledge about concepts that exist in the domain of interest to an information system. For differentiating the technical ontology from the philosophical Ontology, a convention is often employed where the first uses a non-capitalized *o* as first letter, whereas the latter uses a capitalized *O*. (Gómez-Pérez, et al., 2004).

For defining the specifics of what an ontology does in the informatics context relevant to this thesis, a variety of definitions have been coined. The most often cited definition comes from Gruber (1993):

> "An ontology is an explicit specification of a conceptualization."

While being a concise definition, the nature of an ontology is not really explained. However, the definition contains two central concepts, being *conceptualization* as an abstract, simplified, model-based view on things that exist, and *specification*, emphasizing the formality and declarative nature of the approach.

Another definition comes from Hendler (2001). He defines ontology as

> "a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic."

This definition includes a number of other ontology properties. Firstly, the semantic connections between the concepts defined in the ontology are mentioned as being of relevance. Secondly, rules for inference and logic are mentioned. Ontologies usually adhere to some kind of logic that allows deriving new knowledge or draw conclusions from information that is already contained in the ontology.

Another important aspect of ontologies is mentioned by Kalfoglou (2000). He views ontologies as

> "an explicit representation of a shared understanding of the important concepts in some domain of interest. The role of an ontology is to support knowledge sharing and reuse within and among groups of agents (people, software programs, or both)."

This definition emphasizes the shared understanding ontologies provide, with the aim of promoting knowledge sharing and reuse among users and software.

In conclusion, the following characteristics define the core understanding of ontologies in the information science and knowledge engineering sense:

- Ontologies form a model of things that exist,
- enabling the drawing of inferences and the derivation of new information from already existing information,
- with an emphasis on providing a shared understanding and promoting reuse of defined concepts.

3.6.2 Categorization of Ontologies

Ontologies can be characterized by looking at specific properties. An important one of these properties is the subject of conceptualization of the ontology. This property categorizes ontologies into the following groups (Gómez-Pérez, et al., 2004):

General/Common Ontologies

These kinds of ontologies describe generic and abstract concepts applicable to any number of domains, such as time, space, or things.

Top-Level/Upper Level Ontologies

Such ontologies become more generic by focusing on concepts such as tangible things, intangible things, processes, events, etc. These ontologies are still generic enough to be applicable to a number of domains.

Domain Ontologies

These are focused on describing the characteristic entities, activities, or principles of specific domains, such as biology, medicine, economics, engineering, or their respective sub-fields. Often, these ontologies refer to and refine concepts from upper level ontologies.

(Domain) Task Ontologies

These are used to describe specific tasks. They can be specific to one domain or involve activities from a number of domains.

Application Ontologies

Such ontologies are focused on detailing knowledge about specific applications, often specializing domain and task ontologies.

Another approach for categorizing ontologies is to look at their elaboration in terms of language constructs used. This kind of classification relies on the richness of the internal ontology structure and is often done using the following categories (Gómez-Pérez, et al., 2004):

Controlled Vocabularies

Relying purely on a list of concepts, such ontologies do not elaborate at all on the concepts' meaning.

Glossaries

These ontologies extend vocabularies by providing a meaning to concepts, usually using prose text.

Thesauri

These ontologies describe concepts, their properties, and their relation to other concepts.

Taxonomies

These ontologies have a focus on describing hierarchies of concepts. This includes the specification of informal hierarchies, as well as strict is-a hierarchies.

Lightweight Ontologies

Such ontologies involve a formal hierarchy of concepts, attributes of concepts, and a specification of relations to other concepts.

Heavyweight Ontologies

These ontologies involve the utilization of all available ontology constructs such as concept hierarchies, properties, restrictions, axioms, and rules.

3.6.3 The Web Ontology Language OWL 2

The Web Ontology Language OWL in its current version OWL 2 (W3C, 2012a) can be seen as one of the most elaborated and established ontology languages (Gómez-Pérez, et al., 2004). Lying in the responsibility of the World Wide Web Consortium (W3C) it has been developed for usage in the context of the Semantic Web (Hendler, et al., 2002) with the aim of providing a machine-interpretable representation of the knowledge encoded in prose on the World Wide Web. For this purpose, existing means of describing Web resources such as the Resource Description Framework (RDF) (W3C, 2014a) and RDF Schema (RDFS) (W3C, 2014b) were augmented with elements from computational logic, more specifically Description Logics (DL) (Baader, et al., 2007).

The utilization of DL enables two activities with the help of algorithmic programs called reasoners. On the one hand, the knowledge specified in the ontology can be checked regarding its consistency, determining if it contains logical contradictions. On the other hand, knowledge that implicitly exists in the ontology in the form of logical relations can be made explicit, a process called inference. (W3C, 2012b)

OWL 2 ontologies consist of three syntactic elements (W3C, 2012a):

Entities

Entities such as *Classes*, *Object Properties*, *Data Properties*, *Annotation Properties* and *Individuals* form the core of an ontology by describing the main concepts of a domain.

Expressions

These are combinations of entities in terms of *intersections*, *unions*, *complements*, etc. and are regarded as forming entities themselves.

Axioms

Axioms represent statements of the ontology's domain that are always regarded as true and include *subclassing*, *assertions*, *disjointness*, etc.

OWL 2 ontologies revolve around a number of core concepts. First of all, every entity inside an ontology is always identified by an International Resource Identifier (IRI). In addition, due to the open nature of ontologies and the fact that an IRI is used for identification, the naming of entities may be non-unique. Furthermore, ontologies are able to import other ontologies in order to utilize and integrate existing concepts into the own ontology. Also, with the Semantic Web in mind, the *AAA slogan* "Anyone can say Anything about Any topic" (Allemang & Hendler, 2011) results in profound consequences. This principle implies that information newly introduced to the ontology cannot replace or falsify existing information. On the one hand, this results in the principle that an individual in an ontology can be an instance of more than one class,

and that this membership can be changed during the time the ontology exists. On the other hand the *AAA slogan* results in the usage of the Open World Assumption (OWA) where information not present in the ontology is regarded as missing, being possibly true or false, standing in contrast to the Closed World Assumption (CWA) that is used in object-oriented programming and relational databases, where information not present in the model is usually interpreted as false (Allemang & Hendler, 2011; W3C, 2012b).

OWL 2 features two dialects that have an implication on the syntactic diversity of produced models. OWL 2 Full offers all language constructs available in OWL 2, while OWL 2 DL allows only a subset of the language with the motivation to ease implementation of difficult to realize language elements. In addition to these dialects, OWL 2 also comes with three profiles that are as well defined to ease implementation, but also to improve ontology interpretation performance towards specific use cases. As such, OWL 2 EL is focused on providing polynomial time complexity for reasoning on large taxonomies, OWL 2 QL is tailored towards compatibility to relational databases, and OWL 2 RL is focused on rule-based reasoning. (W3C, 2012a)

There are notable constructs that can be used together with OWL 2, the first being the SPARQL Protocol and RDF Query Language (SPARQL) (W3C, 2008) that can be used to perform queries on RDF and OWL data. Another is the Semantic Web Rule Language (SWRL) (W3C, 2004) that can be used to specify specific domain rules in an ontology.

DL-based models, as is the case for OWL 2 ontologies, are divided into two parts, one being the *TBox* (terminological box), containing the terminology and relations of the domain, i.e. its meta-model, the other being the *ABox* (assertional box), containing assertional knowledge about the domain, i.e. its *Individuals* or *Objects*/*Instances*, respectively. However, these two levels should not be mapped to OMG's MOF, as the type relation in OWL 2 does not correspond to instantiation of a class in the object-oriented sense.

3.7 The MagSat Scenario

As overall demonstration scenario, the MagSat spacecraft is used. The MagSat is based on actual system design data near the project's *Preliminary Design Review* (PDR). At selected points, the data scope is extended beyond PDR for demonstration purposes. Data subject of intellectual property issues was deliberately left out.

3.7.1 Overview

The MagSat is a scientific earth observation spacecraft with the main goal to measure the Earth's magnetic field. It is 5.0 m in length, 1.4 m wide, and 0.8 m in height with a wet mass of around 600 kg.

Figure 3.9 gives an overview of the spacecraft's design. Besides a mechanical structure, it consists of a number of subsystems that each fulfils a specific purpose:

The *Electrical Power System* (EPS) contains components such as *Solar Arrays* that generate power, a *Battery* that is used to store electrical energy for periods when no external power can be provided, and a *Power Control and Distribution Unit* that manages charging and discharging cycles.

The *Data Handling System* with its *On-Board Computer* (OBC) is used to manage the distribution of commands that are sent to the spacecraft, process collected scientific data, and package this data for being sent back to the ground, among other things.

The *Telemetry, Tracking, and Command System* (TTC) forms the interface to the ground, consisting of a number of *S-Band antennas* and *radio frequency processing units*.

The *Attitude and Orbit Control System* (AOCS) contains two *Tanks*, a range of *Thrusters* for orbit and attitude control, and sensors such as *Magnetometers* and *Coarse Earth Sun Sensors*.

For fulfilling the core of its mission, a number of scientific instruments are on-board, such as *Star Trackers*, *GPS Receivers*, an *Accelerometer*, and a range of *magnetic instruments*. The latter are used to continuously measure different properties of the Earth's magnetic field, providing the core science data.

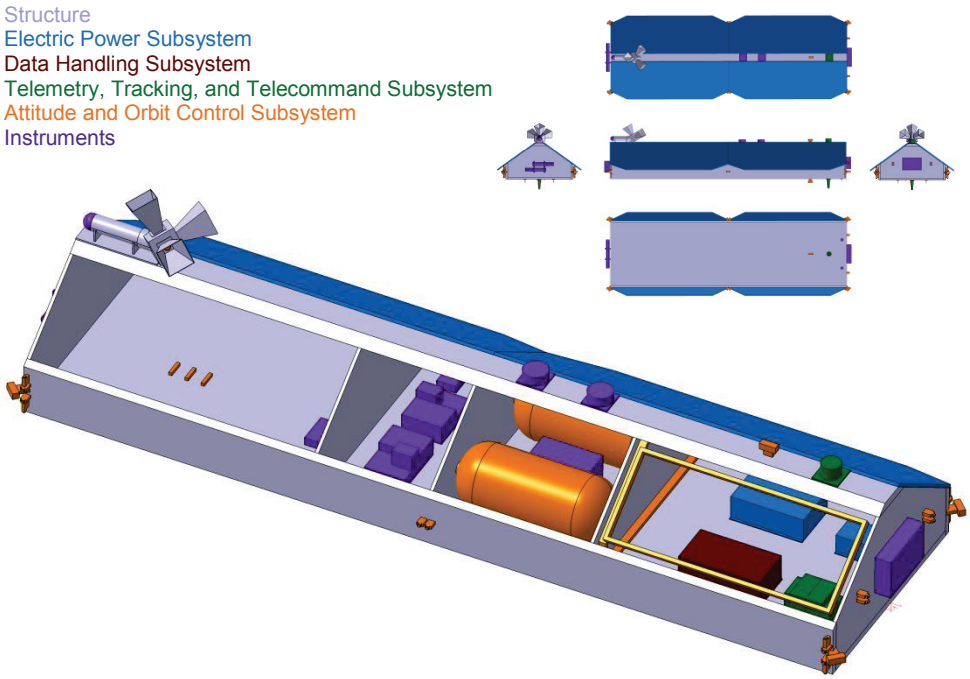


Figure 3.9: Illustration of the MagSat spacecraft

3.7.2 Design Documentation

For describing the design of a space system, a variety of documents are employed. Several of these serve as source of information for illustrating modeling issues or improvements throughout this thesis. One central piece of documentation is formed by the *Product Tree*, which describes the elements that make up the system. An excerpt of the MagSat *Product Tree* is given in Table 3.1.

Another source of information is formed by the spacecraft's System Requirements Specification. An excerpt of this is shown in Figure 3.10.

Table 3.1: Extract from the MagSat Product Tree

MagSat Product Tree			
Config Item No.	**Product Tree Item**	**Abbreviation**	**No.**
0000	MagSat		1
1000	Electrical Power System	EPS	1
1100	Power Control and Distribution Unit	PCDU	1
1200	Battery	BAT	1
1310	Solar Array +Y	SAPY	1
1311	Solar Array +Y Aft Panel	SAPYA	1
1312	Solar Array +Y Bow Panel	SAPYB	1
1320	Solar Array -Y	SAMY	1
1321	Solar Array -Y Aft Panel	SAMYA	1
1322	Solar Array -Y Bow Panel	SAMYB	1
2000	Data Handling System	DHS	1
2100	On-Board Computer	OBC	1
2200	On-Board Software	OBSW	1
3000	Telemetry, Tracking and Command System	TTC	1
3100	S-Band Transponder	SBT	2
3200	3dB Combiner	SBCP	1
3300	Nadir Antenna	NA	1
3400	Zenith Antenna	ZA	1
3500	RF Harness	SBH	1 set
4000	Attitude and Orbit Control System	AOCS	1
4100	Cold Gas Propulsion System	CGPS	2
4110	Tank	TANK	1
4120	Attitude Control Thruster	ACT	8
4130	Orbit Control Thruster	OCT	4
4140	High Pressure Latch Valve	HPLV	1
4150	High Pressure Transducer	HPT	1
4160	Low Presser Transducer	LPT	1
4170	Pipework		1 set

7 Satellite Requirements

7.1 Lifetime, Reliability, Availability and Product Assurance

GSR-1 **In-orbit Lifetime:**

MagSat shall be designed for a duration in-orbit of:

- commissioning phase: 3 months, and
- operational phase at least 48 months.

GSR-2 **On ground Lifetime:**

The MagSat satellites shall be designed for 5 years on-ground activities in controlled conditions including storage time if needed

GSR-3 **Reliability** is defined as the probability that each satellite (platform + payload) will carry out its specified mission for the specified total operational lifetime Each MagSat satellite shall be designed to provide a reliability of higher than 0.8 **(TBC)** over the total operational lifetime.

Note 1: the launch reliability of the selected launcher is considered as 1.

GSR-4 **Operational Availability A0** is the probability that each single satellite, when used under stated conditions in an actual operational environment, provides the required data to the ground segment.

A0= MTBM / (MTBM+MDT) with MTBM= Mean Time Between Maintenance and MDT= Mean Maintenance Downtime.

The MagSat constellation (3 satellites) shall be designed to provide an operational availability A0 during the operational phase (48 months) of higher than 0.9.

Note 1: this assumes an operational availability of the ground station of 0.99

7.2 Design and engineering requirements

7.2.1 Environment

DER-1 **Environment Survivability**

MagSat shall be designed to operate in the space environment as specified in SD-4, and to survive the environment and handling during assembly, integration and testing, transport and the launch.

7.2.2 Launcher and launch environment compatibility

DER-2 **Launcher Compatibility**

The MagSat satellites shall be compatible with at least two launchers.

DER-3 **Launcher Survivability**

MagSat shall be able to withstand the environment generated by the selected launchers without degradation of mission products.

DER-4 **Single Launch Propellant Mass Margin**

MagSat constellation shall be compatible with a single launch with a propellant margin of 20 %.

DER-5 **Single Launch Mass**

The mass of the MagSat constellation including adequate margin at unit and satellite level shall be compatible with a single launch.

Figure 3.10: Requirements sample data

4 Existing Approaches for Describing System Models

A number of different approaches can be used to produce SMs in the context of MBSE. Some of these approaches have become mature enough to become relevant in one or more application domains, while others have not yet gained considerable widespread recognition. This chapter sketches the state of the art in system modeling by elaborating on approaches from both the category of established modeling approaches, and more experimental, not yet widespread modeling approaches.

The analysis considers two factors. These include the overall scope of the implemented CDM, giving an outline of supported concepts, as well as the implementation architecture and technology. For the latter, an emphasis is put on considering the models throughout all involved model instantiation levels, including SMs (M0), CDM (M1), modeling language (M2), and, if applicable, the artefacts on M3 level.

4.1 Approaches Established in the European Space Industry

4.1.1 SysML

SysML has become a relevant factor in performing model-based design of systems and has been employed in a number of industries (Bone & Cloutier, 2010).

Usually, SysML is used in the very beginning of a system's conceptual or functional design, playing a role mainly for finding and elaborating a sensible system architecture inside the given constraints. After a stable architecture has been found, detailed design of system components is performed employing more specialized means of modeling, such as the Very High Speed Integrated Circuit Hardware Description Language (VHDL) (IEEE, 2009), the Automotive Open System Architecture (AUTOSAR) (AUTOSAR, 2016) or UML (OMG, 2015b).

The main way of working with SysML is relying on diagrams to architect the system. Performed activities include system specification in terms of use cases and requirements, definition of system topology using block definition diagrams and internal block diagrams, modeling function-based, message-based and state-based behavior, and defining system verification activities. (Friedenthal, et al., 2008)

4.1.1.1 Data Model Scope

The following concepts are available for architecting systems with SysML (Friedenthal, et al., 2008; OMG, 2015c):

System specification

This part of the language enables modeling stakeholders and use cases, as well as requirements and their relation to system building blocks.

Definition of system topology

The definition of system element breakdown structure and of interfaces between system elements is realized with these concepts.

Definition of system behavior

These concepts support modeling state-based, function-based, and message-based behavior.

Definition of constraints

SysML allows the modeling of constraints between system design parameters using these concepts.

Definition of verification and validation

This part of the language provides the capability to model test cases that can be traced to requirements.

4.1.1.2 Modeling and Implementation Technology

Numerous modeling tools that support SysML exist, such as MagicDraw (No Magic, 2016), Topcased (PolarSys, 2016) and its successor Papyrus (The Eclipse Foundation, 2015), the Obeo Designer (Obeo, 2016), and Modelio (Modeliosoft, 2017). The architecture of the SysML language and its relation to SysML models has been depicted earlier in Figure 3.6.

4.1.2 The Arcadia/Capella Domain-Specific Language

Another approach to system modeling is using the Arcadia/Capella Domain-Specific Language (DSL) (The Eclipse Foundation, 2014) that forms a model-based engineering (MBE) solution for system architecting.

The Arcadia/Capella DSL is inspired by UML and SysML and as such relies largely on graphical modeling. The Capella tool implements the Arcadia MBE method (Thales, 2015) that puts an emphasis on clearly separating user needs on the system from the system architecture. The core concepts of the language are highly focused on solution exploration, situating the main language application case also in the very beginning of system design.

4.1.2.1 Data Model Scope

The Arcadia/Capella DSL enables modeling of the following concepts (Thales, 2015), in the order of employment along a system's design cycle:

Operational Analysis

The activity of Operational Analysis is used to analyze needs, goals, expected missions and activities. The main modeling constructs include operational actors, entities, capabilities, roles, and activities.

System Analysis

System Analysis defines how the system can satisfy operational needs by elaborating on how system functions relate to its high-level architecture and how system operations interact with it. Key concepts include system functions, actors, capabilities, missions, and the system itself.

Logical Architecture

A Logical Architecture is used to identify the system components, their contents, relationships and properties, excluding implementation, technical or technological issues. Central concepts include logical functions, actors, and components.

Physical Architecture

The functionality to define a Physical Architecture is used to identify the system components, their contents, relationships and properties by focusing on implementation details. This focuses on physical functions, and components.

End-Product Breakdown Structure

The End-Product Breakdown Structure maps components to configuration items and defines the final architecture of the system on system engineering level in order to prepare hand-off to lower level development.

4.1.2.2 Modeling and Implementation Technology

The Capella tool is built using EMF (The Eclipse Foundation, 2016b) and as such employs Ecore (The Eclipse Foundation, 2016c) for language specification. Consequently the meta-meta-model is represented by MOF or, more specifically, EMOF (OMG, 2015a). Figure 4.1 outlines this design.

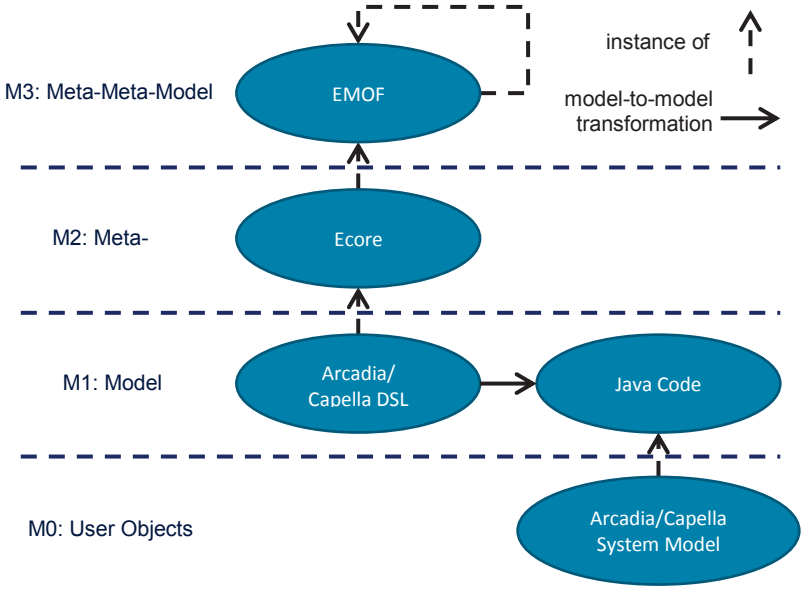


Figure 4.1: Language architecture of Arcadia/Capella DSL

4.1.3 The Concurrent Design Platform

The Concurrent Design Platform (CDP) (RHEA System, 2015) is an application to support Concurrent Engineering (CE). CE is a methodology relying heavily on design tasks running in parallel, where different domains work on different aspects of the system. This involves a high degree of interaction and a high frequency of increments.

CDP is also located in the early stages of system design, between solution exploration and solution elaboration.

4.1.3.1 Data Model Scope

Concurrent system design with CDP revolves around the following core concepts (RHEA System, 2015):

- Requirements
- Product Tree
- System parameters
- Domain analyses
- Engineering activities
- Action lists

4.1.3.2 Modeling and Implementation Technology

As the data model of CDP is not published, no technical details regarding technological architecture can be provided at this point.

4.1.4 ECSS-E-TM-10-25 and the Open Concurrent Design Tool

The European space industry relies heavily on a shared set of standards in order to ensure effective and efficient collaboration between involved stakeholders in a project. For this purpose, a family of standards termed European Cooperation for Space Standardization (ECSS) has been defined (ESA, 2013b). Defining a standard often takes the in-between step of being a technical memorandum, as is the case for ECSS-E-TM-10-25 (abbr. 10-25) (ESA, 2010). This technical memorandum defines the facilities to enable engineering design model data exchange in the context of CE. The principles defined in this document focus on designing CE facilities, enabling data exchange between models from different organizational entities, and enabling real-time collaboration.

The most significant implementation of this technical memorandum comes in form of the Open Concurrent Design Tool (OCDT) (ESA, 2014). As this tool and its data specification are also oriented on concurrent engineering, both have their main application in early system design phases.

4.1.4.1 Data Model Scope

The 10-25 data model as implemented in the OCDT revolves around the following concepts (ESA, 2010; ESA, 2014):

Organization

This concept supports modeling of organizations, disciplines, participants, and the roles they play for the system and the system design process.

Process

This package considers characteristics relevant to the engineering process, such as life-cycle phases, sessions, iterations, and snapshots.

Product

This concept provides the means to model system options, product structure, mission phases, and system modes.

Design parameters

Specification of system design parameters that form an important aspect for system characterization is supported by this concept.

Infrastructure

The infrastructure package represents concepts such as workspaces and reports.

4.1.4.2 Modeling and Implementation Technology

The 10-25 data model is specified in UML using a number of stereotypes (ESA, 2010). The data model is implemented in the three tools that make up the OCDT. A database server called *Persistent Data Store*, specified through SQL, forms the central repository for system design data. Using the *Web Services Processor*, the database offers its services to engineering tools that want to connect. These are mainly made up by numerous instances of Microsoft Excel with the ConCORDE (Concurrent Concepts, Options, Requirements and Design Editor) add-in, acting as user client (de Koning, et al., 2014). Figure 4.2 outlines the OCDT's language architecture.

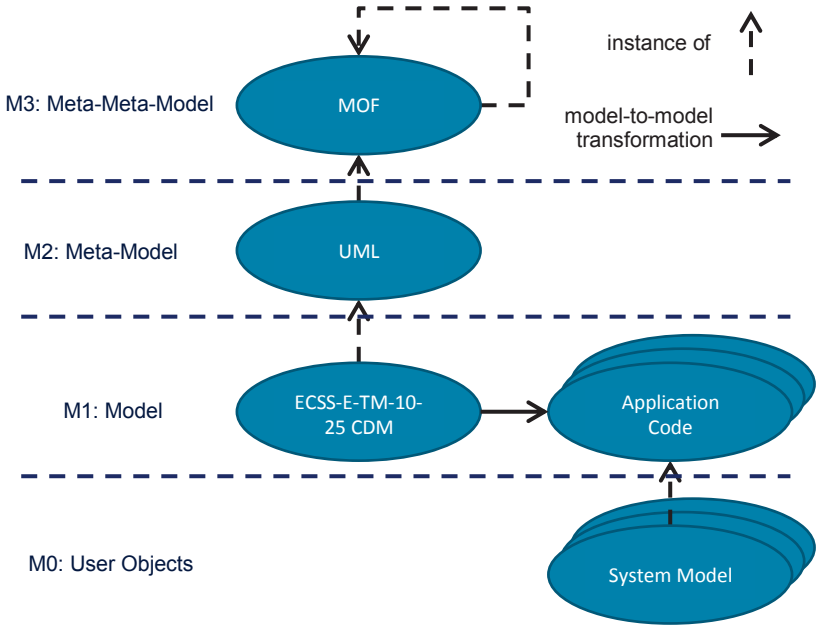


Figure 4.2: Language architecture of the OCDT

4.1.5 ECSS-E-TM-10-23 and RangeDB

In the context of ECSS, another technical memorandum has become important. ECSS-E-TM-10-23 (abbr. 10-23) (ESA, 2011a) specifies how system data is to be exchanged between customers, suppliers, engineering disciplines, across all system decomposition levels, and throughout all phases of the system's design.

Annex B of the memorandum specifies the data and data semantics using a CDM. Furthermore, the standard contains requirements on the architecture used to implement the CDM. 10-23 uses the *SystemElement* as a central concept around which many of the data describing the system is aggregated, being categorized into data modules (Figure 4.3).

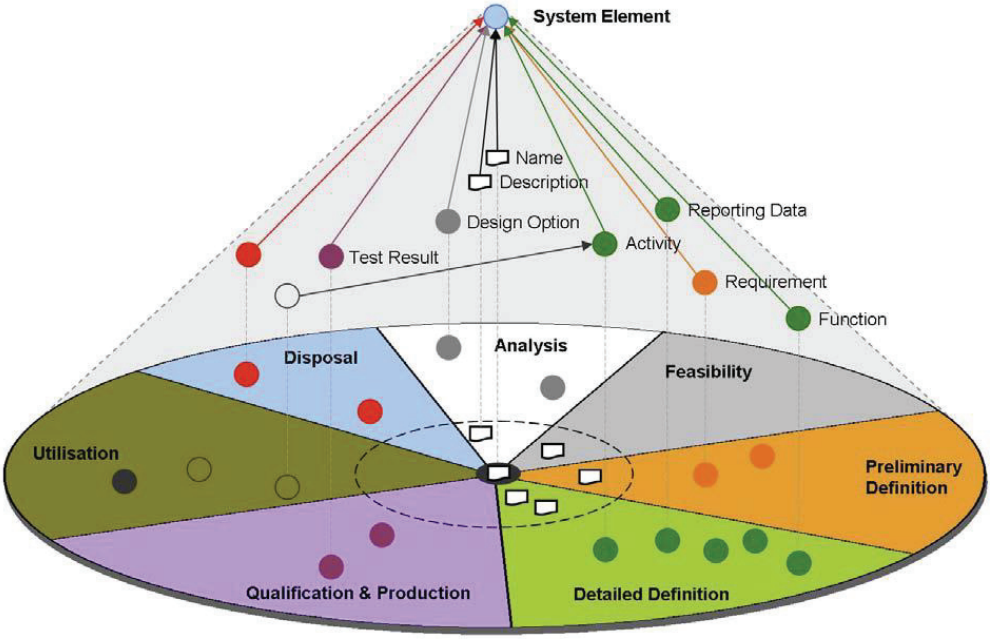


Figure 4.3: System Element Data Modules in ECSS-E-TM-10-23 (ESA, 2011a)

4.1.5.1 Data Model Scope

The 10-23 CDM scopes the following data in its original revision:

System Specification

This package supports modeling requirements and describing a variety of different traces towards elements of the system's design.

Topological Design

This package provides the capability for describing system hierarchy and classification in terms of a product structure, as well as different kinds of interfaces between system building blocks.

Functional Design

Functions, their breakdown structure, and functional flows are represented by the concepts of this package

Operational Design

Operational aspects of the system in terms of operational activities, system modes, and monitoring and control data are represented here.

Assembly, Integration, and Test Design

For specifying the sequence of how the system will be produced, a number of concepts are offered in this package.

Verification Design

This package relates different means of verifying requirements to requirement themselves. The concrete approaches to verification include tests, reviews, inspections, and analyses.

Properties

Modeling of system parameters with the possibility to relate them to physical quantities is enabled by this package.

The 10-23 CDM has been implemented in a variety of projects, such as *Virtual Spacecraft Design* (ESA, 2012a), *Functional System Simulation in Support of MBSE* (Fischer, et al., 2014), and *European Ground Systems Common Core* (ESA, 2013a). Furthermore, a product line for producing engineering data management tools called *RangeDB* (Eisenmann & Cazenave, 2014) has been developed at Airbus DS, also implementing the 10-23 CDM. In the course of these implementations, the 10-23 CDM has undergone noticeable refinement and evolution in a number of areas:

- Refinement and evolution of product structure
- Introduction of aspect principle in respect to 10-23's data module principle
- Evolution of CDM part for describing monitoring and control data
- Refinement of verification activities.

4.1.5.2 Modeling and Implementation Technology

The 10-23 CDM is defined in UML, using a number of stereotypes (ESA, 2011a). It reuses the QUDV model defined by SysML (OMG, 2015c).

For implementation in RangeDB, a platform-independent UML-based LDM is derived from the UML-based CDM. This LDM is then transformed to a platform-specific TDM that comes in the shape of an Ecore model, from which code is generated (Figure 4.4).

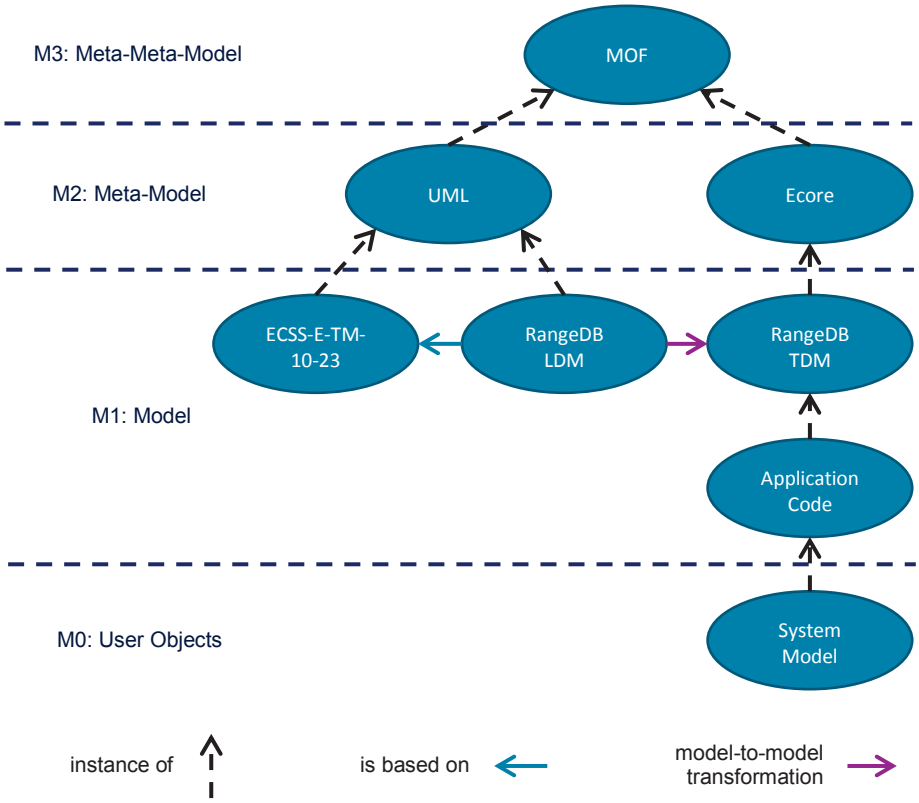


Figure 4.4: Language architecture of RangeDB

4.1.6 Summary of Industrially Established Approaches

Technologically, all examined solutions are based on software-specification languages, such as UML, Ecore, or, more generically, MOF. Figure 4.5 gives an overview on the languages behind the examined industrially employed system modeling approaches.

The examined system modeling approaches have their main usage at different points of the space system design cycle (ESA, 2009a), strongly influencing the data scoped by the underlying CDM. Figure 4.6 provides an overview of the employment of all approaches in respect to the system design cycle.

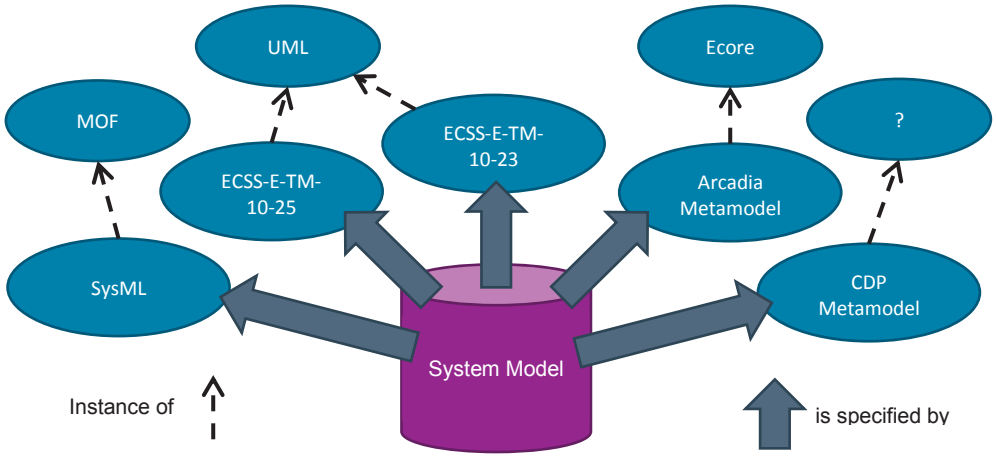


Figure 4.5: Specification languages of examined modeling approaches

SysML and Arcadia/Capella focus on data oriented towards solution exploration. CDP and 10-25/OCDT have their main phase of activity in concurrent solution elaboration. As 10-23 facilitates the representation of data throughout the whole system lifecycle, the usage of RangeDB starts as early as solution exploration, has its main utilization in detailed system design, ranging up to system production and verification.

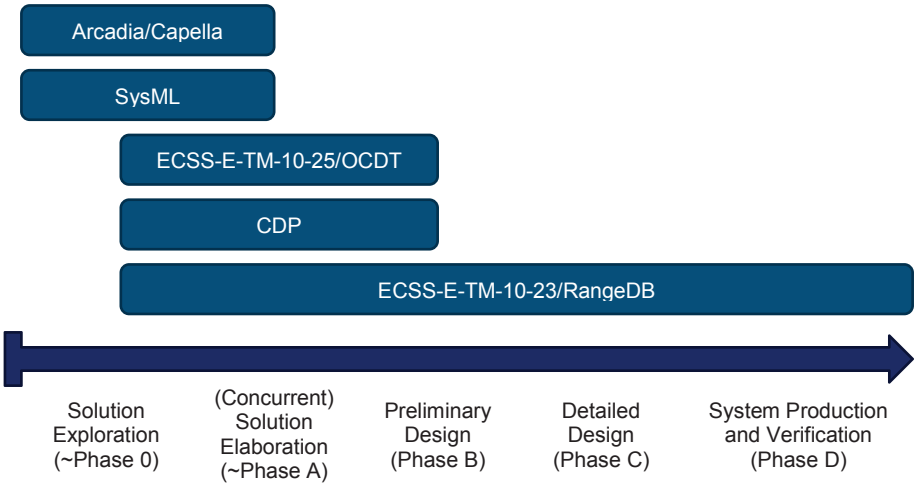


Figure 4.6: Lifecycle overview on examined system modeling approaches

This chapter only provides an overview of employed approaches and technologies. An evaluation of the most significant approaches is performed in the next chapter, putting them into context with current requirements on system modeling, and evaluating their characteristic strengths and weaknesses.

4.2 Approaches not in Widespread Use for Space System Design

Besides the industrially established approaches, other approaches to system modeling have appeared in literature that are not used productively, but are more research-oriented. These approaches have some relevance to the subject of system modeling, for instance as an exploration of a new way to describe systems, or as an alternative description of currently available data.

4.2.1 RDF and OWL Ontologies

4.2.1.1 Positions on Ontologies in the Industrial Context

Numerous authors have taken a position towards ontologies in the SE and system design context by outlining how they can benefit the system design process. This usually involves having a model of the system itself in the shape of an ontology, upon which engineering activities are performed. These works are more general positions towards the benefits of employing ontologies, with negligible detailing of concrete applications.

Oberle (2014) regards ontologies as being of benefit to numerous applications in the enterprise context, due to them being facilitators of agile conceptual modeling and reuse and formality, while offering web compliance and reasoning functionality. The author outlines how this benefits enterprise applications by opening up new business scenarios, improving the productivity of information workers, and improving information management in the enterprise context.

Graves (2007) explores the usage OWL-DL for developing products in the aerospace context by utilizing the language's formality and reasoning capabilities. The author comes to the conclusion that its usage is a good starting point, but further developments are needed to support functions not scoped by the language, but required by the considered engineering domain.

Sarder & Ferreira (2007) take a position on ontologies in the SE context by emphasizing that having an ontology defining central SE concepts might be an advantage,

providing standardized terminology, shared meaning of concepts, and access to concept definitions for all involved parties.

Ernadote (2015) highlights the idea of combining ontologies and meta-models of sub-disciplines to gain an advantage. This approach features several meta-models that are used to exactly specify discipline-specific data as well as one or several ontologies that form a more abstract description of concepts that semantically aligns the detailed data. This abstract description is then to be shared between all stakeholders without the need to understand all of the different, detailed data specifications of the original meta-models.

Graves & Horrocks (2008) employ a foundational ontology in order to evaluate the place of ontologies in the SE context. They come to the conclusion that OWL on its own will not be replacing dedicated MBSE languages such as SysML in the future. However they outline that OWL might have a place for semantic integration between different SE stakeholders.

Chourabi, et al. (2008) provide a semantic description of knowledge relevant to the SE process based on a set of layered ontologies. Their motivation is to define an understanding of concepts in a semantic manner, and to make this understanding available.

4.2.1.2 Engineering Standards Modeled in Ontology Languages

Ontologies have been employed for modeling systems and engineering data across different engineering domains with different motivations.

Van Ruijven (2012; 2015) approached two ISO standards situated in the context of systems engineering from a data point of view and produced an RDF-based specification. These standards include ISO 15926-11 (ISO, 2015a), which is focused on describing plant lifecycle phases in process industry domain, and ISO 15288 (ISO, 2015b), which can be used to describe a system along its full life-cycle from stakeholder requirements up to system disposal. Both standards are merged into an RDF/RDFS ontology with OWL deliberately not being used due to the closed world nature of the application domain. The ontology focuses especially on the stakeholder requirements definition process, the requirements analysis process, the operation and maintenance process, the verification process, and the risk management process described in the standards. Klüwer, et al., (2008) proposed using OWL 1 to model the same ISO 15926 standard. These works mainly focus on formalizing the information described in the standards, without going into utilization of modeled instance-level data.

4.2.1.3 Using Ontologies to Improve System Quality

One motivation for modeling systems with an ontology is to improve the system's quality. For this purpose, numerous authors have been producing ontological descriptions of their system and applied some mechanisms to evaluate model and system correctness. This stands in conjunction with the idea that an error in the system's model might result in a problem in the system design itself. If the problem is identified on model level, it can be corrected in the design, improving system quality.

Besides reasoning on artefacts of a software design, Wende, et al. (2013), employ OWL 2 ontologies to check specific kinds of consistency on small systems. One example is using a reasoner to check if a certain kind of add-in card may be inserted into a specific slot of a router. Furthermore, after detecting an inconsistency, the authors use the reasoner to suggest allowed card categories for a specific slot. This approach relies heavily on comparing functions required by the design vs. functions fulfilled by it, and does not consider further aspects of consistency.

Feldmann, et al. (2015) try to solve the challenges associated with managing inconsistencies in models of automation systems using RDF. For this purpose, an RDF knowledge base is defined with potential inconsistencies being modeled as SPARQL queries. The system is then modeled using the ontology. Should its model or the system itself exhibit an inconsistency, it gets highlighted by the according SPARQL query. The authors state that the maturity of the presented demonstration case is based on lab conditions, containing numerous academic assumptions. Furthermore, the presented SPARQL-based inconsistency identification approach does not support making identified inconsistencies available as additional facts in the SM, falling short of further exploiting gained information in subsequent logical steps.

Abele, et al. (2013) aim at validating interdisciplinary- models of industrial plants by transforming them into an OWL representation and applying SPARQL queries and a reasoner, with the motivation of identifying inconsistencies that came in through the interdisciplinary description. The authors consider aspects such as duplication of internal elements, validating role requirements on system components, and checking for the correctness of internal links. As this work is also based on checking consistency with SPARQL-based queries, the same disadvantages also present in the work of Feldmann, et al. (2015) apply, as identified information cannot directly be used to provide further inferences.

Thakker, et al. (2015) developed an ontology set for diagnosing tunnel pathologies. For this purpose, tunnel disorders as identified by experts are modeled in an ontology and, them being symptoms of disorders, the tunnel disorders can be inferred. The authors go further by also inferring regions of interest in a tunnel that might become problematic with a similar pathology in the near future. While the used pattern works

quite well in the detailed use case, the pattern is also highly specific to concluding on disorder-driven problems. Consequently, the approach is not suitable to problems not falling into this specific pattern.

4.2.1.4 NASA IMCE Ontologies for Merging OWL and SysML

Graves (2009) proposes integrating SysML and OWL by transforming Block Definition Diagrams to a corresponding OWL 2 ontology, representing system parts, part decomposition, and connections between parts. The motivation behind this approach is to use reasoning to logically evaluate design consistency. Graves (2012) continues to elaborate this principle by extending the scope beyond Block Definition Diagrams and maps SysML constraints to DL assertions, using the OWL 2 model as a vehicle to evaluate the consistency of the system's design originally specified in the SysML model. These works emphasize that a translation from SysML to OWL 2 can only be performed for cases that involve concepts that exist in both languages.

Jenkins & Rouquette (2012) built an ontology for SE describing foundational, discipline, and application concepts (Rouquette, 2010). The authors populate the ontology by transforming system design data contained in a SysML model for evaluating model well-formedness, evaluating adherence to business rules of the domain, and feature extraction for further transformation.

The works of Rouquette (2010) as well as Jenkins & Rouquette (2012) have led to the publishing of the NASA IMCE Ontologies (Jet Propulsion Laboratory, 2016). These ontologies form an extensive set of loosely connected lightweight ontologies describing a number of key concepts for the space system design process. For example, the following concepts are described:

Project: Program, Work Package, Facility, Organization, Stakeholder, Process, Assignment

Mission: Objective, Product, Environment, Function, Requirement

Product Breakdown Structure: Segment, System, Module, Subsystem, Assembly, Part

Artefact: Document, Document Element

Analysis: Assumption, Fact, Explanation, Constraint, Metric, Criterion

Math: Coordinate System, Coordinate, Localization

Mechanical: Geometry, Axis, Sketch, Curve, Body

Electrical: Component, Channel, Data Message, Interface, Link, Signal Flow

Behavior: Automaton, Transition, Interaction, Interaction Behavior

Risk: Decision, Event, Fault, Fault Tree

Fault Management: Detection Mechanism, Likelihood, Cause Explanation, Mission Impact, Mitigation, Prevention

While these ontologies describe a number of key space system design concepts, the significance of the relations between them is based on an application-specific interpretation that brings together the SysML and the ontology model of a system. Consequently, the ontology does only directly allow inferring basic subclass-superclass relationships and a evaluating a small amount of disjoints. Data exploitation beyond these inferences is not provided.

4.2.2 Fact-Based Modeling

Under the name of Fact-Based Modeling (FBM) (Valera, 2014) or Fact-Oriented Modeling (Halpin & Morgan, 2008) an approach that provides both a notation, and a protocol for producing conceptual models on the M2 layer of the MOF architecture has gained some relevance.

Numerous approaches have been developed over the years that follow the paradigms of FBM, such as the Natural Language Information Analysis Method (NIAM) (Nijssen, 1978), CogNIAM (CogNIAM.eu, 2015), Fully Communication Oriented Information Modeling (FCO-IM) (Bakema & Zwart, 2008), FAMOUS (Valera, 2014) and Object Role Modeling (ORM 2) (Halpin & Morgan, 2008; Halpin, 2009).

The methods underlying the FBM approach focus on capturing domain knowledge in a guided manner, decomposing this knowledge into elementary facts, and modeling these facts with a notation specific to the according FBM dialect. An elementary fact is seen as a statement of information that cannot be divided into further sub-facts without changing its meaning (Halpin & Morgan, 2008). A close relative to these languages is given by the Semantics of Business Vocabulary and Business Rules (SBVR) standard (OMG, 2015d; Bollen, 2008) in responsibility of the OMG that focuses on describing facts of the business domain in a textual manner, without a graphical notation.

The FBM dialects focus on producing CDMs, agnostic of any implementation technology. One of the most prominent CDMs modeled with an FBM dialect is the CDM of the Vega Launcher Interface Database (ViDB) that has been modeled using the ORM 2 notation (Valera, 2014).

An excerpt from the ViDB model is depicted in Figure 4.7, giving the basic conceptual structure of a monitoring and control packet. Each packet (PKT) belongs to exactly

one System Element (SE). It has exactly one description and is of exactly one type. It may optionally have a description of parameter locations (PLF), assigning the beginning of a parameter to a bit address inside the packet. A packet is further subclassed into either an IRIG packet, or into a 1553 packet, each having a different set of mandatory characteristic data.

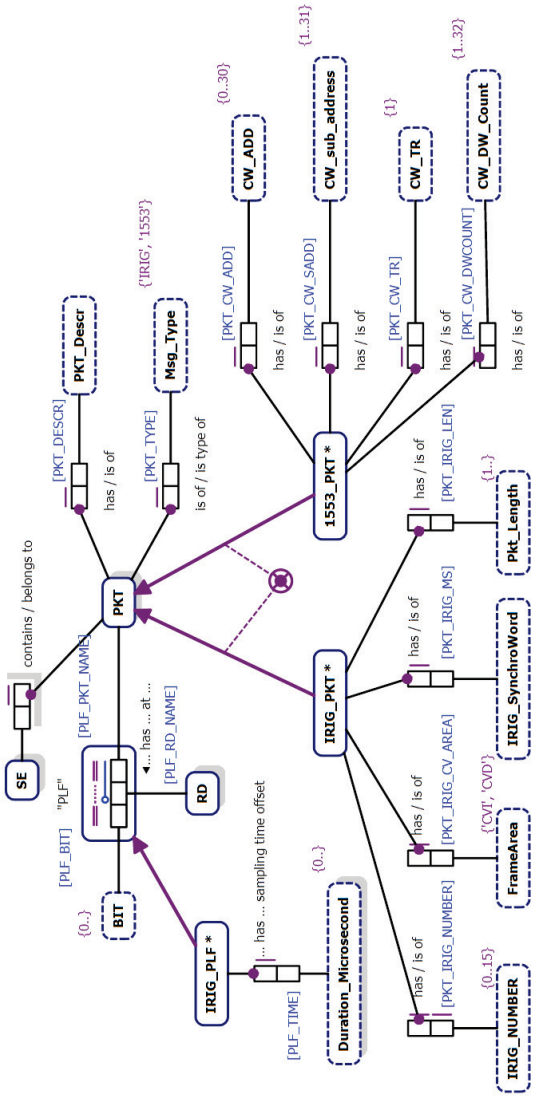


Figure 4.7: Snippet from the ViDB CDM in ORM 2 notation (Valera, 2014)

4.2.3 The OpenMETA Tool Chain

A tool chain named OpenMETA (Sztipanovits, et al., 2014) was developed over the last years that is primarily focused on improving the overall design of Cyber-Physical Systems (CPS). For realizing this, an approach consisting of three framework layers is proposed:

Model Integration Framework

On this level, the models of domains involved in the system's design are integrated with each other. An integration meta-model is provided per domain model that maps its data to an integrated system-wide model based on the language CyPhyML (Neema, et al., 2015). Data from this integrated model is then mapped to the OpenMETA Semantic Backplane, realized with the FORMULA tool (Microsoft, 2017), providing the system model with a set of logic-based semantics, such as design constraints between components, the relation of components to specific requirements, or the nature of interfaces a component offers.

Tool Integration Framework

This layer provides a number of model transformations for enabling model-based design flows, handing over data from one model to another. These transformations are used for a number of purposes. One of these is translating a model from one syntactic form into another for the purpose of using it in another tool. Another motivation is realizing virtual prototyping, transforming a design model to an analysis model. Another transformation is given by systematically varying the parameters of a given model for the purpose of design space exploration.

Execution Integration Framework

This level realizes the execution of the design and analysis flow, performing the model-to-model transformations in their defined sequence, transferring data from one model to another model.

These three levels are used to integrate the views of the various domains involved in the system's design, enabling the model-based consideration of the system in its entirety. The motivation behind this framework is to improve the design process by providing a number of options for system modeling and analysis, reducing the overall amount of design-build-test-redesign cycles.

The OpenMETA framework is strongly focused on the component-based design of CPS and does not consider other system design approaches. Other aspects of the system engineering process such as discipline coordination and lifecycle management of data are also not considered. Furthermore, the framework architecture as described does not allow inferring new knowledge from already existing system design information.

4.2.4 Other Work

Borst (1997) promotes the usage of ontologies on the design of systems with the motivation of knowledge sharing and reuse in mind. The author constructs a variety of domain ontologies focused on engineering physical systems by weaving together smaller, more focused ontologies.

Van Renssen (2005) developed an ontology that specifies a formal language for describing systems, based on the principles of natural language. This language called Gellish can be seen as a formal, controlled subset of natural language, with representations in different languages, such as English, German, or Dutch, that can be keyed to the same ontological concepts and exchanged accordingly. Gellish relies on using tables to represent information. The structure of the tables is given by the data model, whereas the tables can be filled by using pre-defined concepts offered by the Gellish ontology, or rather by each of its language-specific variants. As such, Gellish refrains from making a strict level-based distinction between instances and CDM.

4.3 Conclusion

This chapter performed a survey on existing approaches for modeling systems, encompassing established technologies in the European space industry, as well as experimental approaches that are still under research. The next chapter will perform an evaluation of the most important approaches mentioned throughout this chapter, based on requirements defined for an industrial system modeling approach applicable to the space domain.

5 Analysis of System Modeling Approaches

In the beginning of this chapter, an analysis of the industrial MBSE process is performed for deriving current requirements in this context. Subsequently, these requirements are mapped to the established system modeling approach at Airbus DS, drawing a picture of the current state of the art in system modeling in the model-based space engineering context. In addition, the same analysis is performed for approaches that are not industrially established, evaluating how well models based on OWL 2 and ORM 2 are able to support the specified needs. Consequently, this chapter answers the initial research question:

(RQ1) To what extent are current solutions to system modeling able to fulfil the needs that result from existing challenges of the MBSE process?

After a preliminary conclusion is drawn on the outcome of the analysis, an improvement approach is derived detailing how all defined requirements can be satisfied, starting with existing technologies.

5.1 Requirements on an Industrial System Modeling Approach

For supporting MBSE with an engineering tool that is based on data specified in a CDM, a number of requirements can be formulated on the CDM definition process, CDM definition concepts to be available, and to be specified CDM content. These requirements are based upon the characteristics outlined in Chapter 2, and especially the characteristics detailed in section 2.4.

5.1.1 Requirements on CDM Specification Capabilities

The data scoped by the CDM often has ties to the artefacts of the engineering process on PLM level (see 2.4.4). While the artefacts form a very abstract representation of engineering data, the CDM represents the data in detail. To make these relations or mappings explicit, a requirement is defined:

(REQ-1-1) Availability of explicit mappings between discipline data and process artefacts.

Mentioned in 2.4.5 is the fact that ensuring the consistency of engineering data is tied to evaluating the constraints defined in the CDM. For this purpose, it has been established that the constraints should be available as direct CDM elements, and not in text-based complement to the CDM in a language such as OCL (OMG, 2014b). This leads to the next requirement:

(REQ-1-2) Availability of required constraints in a conceptual manner in the CDM.

For ensuring that the SM can cope with closed world semantics (see 2.4.6), an additional requirement is defined:

(REQ-1-3) Ability to specify closed world facts.

Between CDM elements, a variety of functional dependencies may exist (see 2.4.7). This includes the necessity for a specific element to exist based on given preconditions, or the necessity for one instance to mirror data structures at a related instance. These functional dependencies between CDM elements are usually not defined on model level, but are present implicitly by their implementation in the program code. Due to this, although these concepts have a high conceptual relevance, they do not appear at all in the CDM. Thus, an additional requirement is defined:

(REQ-1-4) Capability to specify functional dependencies between model concepts.

In order to capture the numerous element characterization mechanisms appearing throughout various kinds of engineering data (see 2.4.8), their consideration on CDM level is required. If these various typing mechanisms are not explicitly considered, workarounds usually occur where functionality and model structures specific to a given typing mechanism are separately implemented. Consequently the next requirement is defined:

(REQ-1-5) Support for multiple explicit element characterization mechanisms.

For catering to the lifecycle aspect on engineering data in the space system design context (also see 2.4.9), another requirement is defined:

(REQ-1-6) Support to define life-cycle aspects on data.

5.1.2 Requirements on the CDM Specification Process

As outlined in 2.4.1, numerous modelers tend to produce different models of the underlying engineering data. In order to enforce some structure on this process and to harmonize the modeling activity, the requirements to have some kind of process for CDM definition is defined.

(REQ-2-1) Availability of an overall process for CDM design.

Furthermore, in order to closely align the content of the CDM to the underlying engineering data, some kind of procedure should be available that allows deriving the CDM directly from existing engineering data, leading to the next requirement:

(REQ-2-2) Availability of a procedure to derive the CDM from engineering data.

Also, in order to minimize iterations on the CDM, a mechanism should be present that ensures that as many constraints as possible are captured during design time of the CDM, reducing iterations on the CDM after it has been deployed in an engineering application. For this purpose, a procedure ensuring that all constraints existing in the engineering data are captured is desired.

(REQ-2-3) Availability of a procedure to ensure exhaustiveness of modeled concepts.

With the same underlying motivation to reduce iterations on the CDM after it has been implemented, some kind of CDM validation is required that ensures that the CDM can accurately instantiate the engineering data that it is supposed to represent, leading to the next requirement:

(REQ-2-4) Availability of CDM validation procedures.

For supporting the practice of tailoring, a widespread concept in space system design (see 2.4.2), another requirement is formulated:

(REQ-2-5) Capability for providing project-specific customizations.

Experience has shown that the semantics of the CDM may differ from those given by the implementation of the CDM (see 2.4.3). For ensuring that the SM does not allow the specification of data that would be inconsistent in respect to the CDM, an additional requirement is formulated:

(REQ-2-6) Semantic accuracy of implemented CDM identical to specified CDM.

5.1.3 Requirements on Support of System Engineering Processes

This section deals with requirements on the CDM in terms of content, enabling support for integrating the specific discipline-specific engineering processes with the SE process.

For this purpose, the data already covered by 10-23 has to be scoped by the CDM (ESA, 2011a), resulting in the need of the following requirements:

(REQ-3-1) Support for product structure definition

(REQ-3-2) Support for requirements definition

(REQ-3-3) Support for operational design definition

(REQ-3-4) Support for system architecture definition

(REQ-3-5) Support for system verification definition

(REQ-3-6) Support for system property definition.

For catering to the high relevance of execution data (see 2.4.10), the prerequisites to semantically correlating the results of a system execution with system design data should be provided on system level, leading to the following requirement:

(REQ-3-7) Usage of execution data for system validation.

Furthermore, the fact that operational knowledge has a high relevance in the SE process (see 2.4.11) also has an impact on the SM. After experts in a specific engineering activity move on to other responsibilities, operational knowledge only existing implicitly often gets lost. Also, operational knowledge may be documented explicitly, but might not exist in a semantic description so it can be applied to a system, resulting in a manual knowledge application process. As such, a mechanism for capturing and formalizing operational knowledge about a specific system into a knowledge base is advantageous. Furthermore, a mechanism that facilitates the automatic application of this knowledge to a system currently under design is required, leading to the next requirement:

(REQ-3-8) Existence of a mechanism for capturing and applying operational knowledge.

5.1.4 Process Constraints

Besides requirements on the processes and concepts revolving around the CDM, constraints given by the organizational context it is embedded in are also of relevance.

As first experience with developing engineering tools employing the concept of MDA with technologies like EMF has shown promise (ESA, 2012a), this has since become the main approach for deployment of engineering applications in this context (Fischer, et al., 2014; Eisenmann & Cazenave, 2014). Consequently, the compatibility of any engineering data management approach to the principles of MDA and to EMF is regarded as necessary, leading to the following constraint:

(REQ-4-1) Compatibility to MDA and EMF.

5.2 Requirements Analysis

In this section, an analysis is performed on how each of the specified requirements is satisfied by the currently established modeling approach. In addition, a further analysis is performed on how well other modeling approaches can cope with the requirements, where applicable.

5.2.1 Requirement Fulfilment by RangeDB/10-23 Approach

For evaluating the defined requirements, RangeDB (see 4.1.5) is chosen as representative analysis subject from the category of industrially established approaches. On the one hand, RangeDB implements the 10-23 CDM, incorporating recent evolutionary updates produced by related studies. On the other hand, RangeDB encompasses the largest part of the system lifecycle, covering data from early system design up to system verification (also see Figure 4.6). This approach is considered in its entirety, including the specification technology of the CDM, the specification process, the CDM itself, and its implementation.

The approach stands as an example for system modeling based on object-oriented technologies, incorporating aspects such as UML and Ecore as languages, and MDA/MOF as important principles. As such, it stands as a representative example for realizing object-oriented system modeling, also encompassing many of the elements of SysML.

5.2.1.1 Fulfilment of Requirements on CDM Specification Capabilities

RangeDB in its role as system database integrates data from various engineering disciplines and is used as a source for extracting PDM-relevant data. However, the explicit data mappings as required by REQ-1-1 are not scoped by this approach.

A certain amount of constraints is available (REQ-1-2), but several constraints are not considered on CDM level, such as subset constraints, object cardinality constraints, and specific kinds of ring constraints. Although subset constraints are scoped by UML (OMG, 2015b), these do not get transformed to the Ecore model, as Ecore does not support the subsetting of references (The Eclipse Foundation, 2016b).

As 10-23 and RangeDB are based upon the usual object-oriented programming principles, the data modeled in the according system models exhibit closed world behavior (REQ-1-3).

The 10-23 CDM defines concepts that exhibit strong functional dependencies (REQ-1-4) between each other, as is the case for the product structure. However, these functional dependencies are neither specified exhaustively in prose, nor specified in a semantic manner. The behavior of these functional dependencies is distributed across numerous points of the program code of the engineering application, but there is no real specification on a conceptual level.

The 10-23 CDM involves multiple layers of element characterizations, across various areas of the CDM. For example, in order to describe the nature of an electrical port, the element in question is an instance of the class *InterfaceEnd*. However, it also has a reference with name type to an *InterfaceEndDefinition*, and can be assigned *Categories* for further refinement. This results in three heterogeneous characterization approaches for a single concept. In some cases, the semantics of these characterizations are implicitly given in the program code, in other cases these are not specified at all. Consequently, 10-23 as defined in UML fails to address REQ-1-5.

Lifecycle aspects on data as outlined in REQ-1-6 are also not scoped by 10-23 or any of its implementations.

5.2.1.2 Fulfilment of Requirements on the CDM Specification Process

Regarding a process driving the CDM definition (REQ-2-1) in its specification language UML, an ad-hoc approach was pursued without extensive procedural guidelines. Also, the CDM was not directly derived from engineering data (REQ-2-2), but more in an iterative process, resulting in numerous iterations until a fully validated CDM design was found (ESA, 2012a). Furthermore, as constraints do not play a significant role in the UML-based CDM, no process to derive constraints is available (REQ-2-3). The CDM was validated after a first application was produced, without a dedicated activity existing for pre-validation the CDM prior to implementation (REQ-2-4).

The capability for project-specific customization of the CDM (REQ-2-5) is given using two approaches. On the one hand the concept of *Categories* was introduced, forming a runtime-loadable library of system properties that can be project-specific. On the

other hand, should customizations apart from these properties be required on an engineering database for a specific project, the CDM is adapted and the application re-deployed based on a new implementation.

For implementing the 10-23 CDM (REQ-2-6), a number of pragmatic approaches have been taken that enabled the effective deployment of the engineering application, but opened up possibilities to specify semantically incorrect model populations. This applies to several areas, such as the allocation of system element aspects to the product structure, and the allocation of categories to system elements.

5.2.1.3 Fulfilment of Requirements on Support of System Engineering Processes

10-23 supports the definition of a system's product structure (REQ-3-1), however system variants are not considered. The definition of requirements (REQ-3-2) is well supported, as is the definition of a system's operational design (REQ-3-3). The part for operational parameters was elaborated significantly within the EGS-CC project (ESA, 2013a). Defining system architecture (REQ-3-4) and system verification (REQ-3-5) is adequately supported.

The definition of system properties (REQ-3-6) is supported by 10-23 and RangeDB, but does not involve managing uncertainties in properties, such as margins and assumptions.

Semantically relating system design data and system execution data (REQ-3-7) is not scoped by 10-23 and RangeDB.

A mechanism to capture operational knowledge across a number of projects and apply this knowledge to later projects (REQ-3-8) is not scoped by 10-23. It is scoped by RangeDB in a limited way as there is the possibility to hard-code consistency checks in the application.

5.2.1.4 Satisfaction of Process Constraints

The constraint that the specification technology used for defining the CDM has to be compatible to MDA and EMF (REQ-4-1) is fully satisfied. UML as specification language of 10-23 is an essential part of the MDA, and a bridge to EMF is provided via MOF throught their common ancestor, EMOF.

5.2.2 Requirement Fulfilment with ORM 2

As RangeDB does not yet fully address all defined requirements, it is of benefit to know to what extent the other two approaches that have exhibited some usage for modeling engineering data can satisfy them. For this purpose, a similar evaluation of requirement fulfilment is pursued with both ORM 2 and OWL 2.

As ORM 2 has by far the most extensive publication state and is one of the most recent and widespread dialects of FBM and will thus be used as an example from the family of Fact-Based Modeling languages. The analysis performed in this section and the following section picks up on analyses of performed in earlier publications (Hennig, et al., 2015; Hennig, et al., 2016a; Hennig, et al., 2016b). These analyses are integrated at this point and evolved to be aligned to the industrial needs defined in 5.1.

5.2.2.1 Fulfilment of Requirements on CDM Specification Capabilities

The aspect of mapping data in the system model to data scoped by the embedding engineering process is not scoped by ORM 2 (REQ-1-1). The constraints available in ORM 2 have proven to be a good fit to for specifying CDMs in the MBSE context (REQ-1-2). ORM 2 is based upon the CWA and as such is a good fit to the MBSE process (REQ-1-3). ORM 2 offers the capability to specify business rules between model concepts with its rule-based extension FORML 2 (Halpin & Wijbenga, 2010). However these rules do not fully address the functional dependencies as required REQ-1-4. Multiple characterization mechanisms as required by REQ-1-5 are not supported by ORM 2 as the language implies that any instance in the SM is the instance of exactly one *Entity Type* of the CDM. The definition of life-cycle aspects on data as described by REQ-1-6 is not scoped by ORM 2.

5.2.2.2 Fulfilment of Requirements on the CDM Specification Process

A number of methodologies exist for deriving fact-based models from an underlying set of data, such as the methodologies of NIAM (Nijssen, 1978), CogNIAM (CogNIAM.eu, 2015), and the most recent one being the Conceptual Schema Design Procedure (CSDP) (Halpin & Morgan, 2008) that directly supports ORM 2. The latter adequately supports the requirement (REQ-2-1) for deriving CDMs in the MBSE process under the assumption that CDMs are in ORM 2 syntax. The ORM 2-based methodologies also give strict guidelines of how facts derived from any underlying data documentation are to be translated to a model. For deriving a CDM from instance-level data, strict guidelines are provided form ORM 2-based models (REQ-2-2). The same applies to deriving conceivable constraints from available data (REQ-2-3).

CDM validation procedures as defined in REQ-2-4 are not scoped by the ORM 2-based methodologies. Providing project-specific customizations as outlined in REQ-2-5 is not scoped by any part of ORM 2. Similar to the approach used for implementing the 10-23 CDM in UML with the technologies offered by EMF, semantics of the ORM 2 CDM are also often sacrificed for efficient implementation (REQ-2-6). For instance, if an ORM 2 based CDM is mapped to an Ecore model for effective implementation, some of its semantics are lost as they are not scoped by the Ecore language. This is the case, for example, for n-ary fact types.

5.2.2.3 Fulfilment of Requirements on Support of System Engineering Processes

REQ-3-1 throughout REQ-3-6 imply the availability of domain concepts in the CDM. No version of a 10-23 CDM is available in ORM 2, but in theory, these concepts can be represented without major issues in this language. A mechanism for relating system design data to system execution data as specified in REQ-3-7 is not scoped by ORM 2. As ORM 2 does not exhibit any notion of a knowledge base or any kind of knowledge application mechanism, no support for capturing operational knowledge from one system and applying it to another system (REQ-3-8) is available.

5.2.2.4 Satisfaction of Process Constraints

REQ-4-1 highlights the compatibility to both MDA as concept and EMF as technology, being an important constraint towards the industrial deployment of the modeling approach. ORM 2 does not per se exhibit a direct compatibility with MDA and EMF, however a transformation-based implementation toward Ecore is possible, as was already demonstrated in preceding research (Hennig, et al., 2016a).

5.2.3 Requirement Fulfilment with OWL 2

A similar analysis is pursued with OWL 2. OWL 2 is regarded as the most widespread and advanced ontology modeling language and will be used as representative example from the world of knowledge-oriented languages.

5.2.3.1 Fulfilment of Requirements on CDM Specification Capabilities

Mapping data in the system model to data scoped by the embedding engineering process is not part of the OWL 2 language (REQ-1-1). OWL 2 does not have a constraint concept in the traditional sense and instead relies on axiomatically specifying

information. Several constraints required for MBSE CDMs are found to be incompatible with OWL 2's OWA, while the concept of disjoints can be regarded as offering basic logical constraining (REQ-1-2). OWL 2 with its OWA is causing problems to the specification of engineering data (REQ-1-3). For specifying rule-based behavior (REQ-1-4), OWL 2 offers rule modeling capabilities via SWRL (W3C, 2004), but fails to fully address the defined requirements. Regarding typing, an *Individual* in an OWL 2 ontology can have an arbitrary number of types associated with it. In object-oriented terms, it can be an instance of multiple classes at the same time, and this membership can be changed during model runtime, fully addressing REQ-1-5. The definition of life-cycle aspects on data (REQ-1-6) is also not supported directly by OWL 2.

5.2.3.2 Fulfilment of Requirements on the CDM Specification Process

In the ontology world, a number of methodologies for building ontologies have come up over the years, such as METHONTOLOGY (Fernández, et al., 1997), OTKM (Sure, et al., 2004), or the NeOn Methodology (Suárez-Figueroa, 2010). These methodologies largely focus on high-level activities for building ontologies, providing rough guidelines, but do not detail how to derive specific data structures from an underlying information base. This does not fully address REQ-2-1. Such an activity to directly derive a CDM from available instance-level data is not scoped for the OWL 2-based methodologies (REQ-2-2). The same is true for ensuring exhaustiveness of modeled concepts (REQ-2-3). CDM validation procedures (REQ-2-4) are not scoped by the methodologies associated with OWL 2. As the instantiation mechanisms in OWL 2 work somewhat differently (more in 0), project-specific extensions or customizations to a CDM (REQ-2-5) can be performed during system model runtime with the changes automatically being propagated. OWL 2 ontologies do not have to be implemented in order to allow producing a system model, making the employed conceptual data structures on instance level identical to the originally specified one (REQ-2-6).

5.2.3.3 Fulfilment of Requirements on Support of System Engineering Processes

As with ORM 2, no 10-23 CDM is currently available that is based on OWL 2. However, as the original 10-23 CDM encompasses mainly classes, attributes, and relations, the available language concepts in OWL 2 are sufficient to represent relevant domain data (REQ-3-1 throughout REQ-3-6). A mechanism for relating system design data to system execution data as specified in REQ-3-7 is also not scoped by OWL 2 directly, but has to be provided in an according CDM. OWL 2 offers the capability to import ontologies and to use information specified in other ontologies with help of a reasoner, utilizing it for deriving knowledge about a system (REQ-3-8).

5.2.3.4 Satisfaction of Process Constraints

OWL 2 has in theory be made compatible to MOF with help of the Ontology Definition Metamodel (OMG, 2014c), but a real integration to the MDA does not exist. Several implementations of the ODM have been proposed, but all of these approaches do not yet realize genuine dynamic multi-instantiation, with either having shortcomings with realizing multiple instantiation, or with dynamic reclassification (Hoppe, et al., 2017).

5.2.4 Analysis Summary

Table 5.1 provides an overview of how each of the evaluated approaches fulfils the specified requirements.

Table 5.1: Summarized comparison of system modeling approaches

REQ	Requirement	10-23/RangeDB	ORM 2	OWL 2
1-1	Availability of explicit mappings between discipline data and process artefacts	not scoped	not scoped	not scoped
1-2	Availability of required constraints in a conceptual manner	partially	yes	different concepts, CWA problematic
1-3	Ability to specify closed world facts	yes	yes	no adequate support
1-4	Capability to specify functional dependencies between model concepts	not scoped	inadequate, via business rules	inadequate, SWRL
1-5	Support for multiple explicit element characterization mechanisms	no	no	yes
1-6	Support to define lifecycle aspects on data	not scoped	not scoped	not scoped
2-1	Availability of an overall process for CDM design	rough guidelines	strict guidelines	rough guidelines
2-2	Availability of a procedure to derive the CDM from engineering data	with requirements as in-between step	strict guidelines	not scoped
2-3	Availability of a procedure to ensure exhaustiveness of modeled concepts	not scoped	strict guidelines	not scoped

REQ	Requirement	10-23/RangeDB	ORM 2	OWL 2
2-4	Availability of CDM validation procedures	through requirements verification	not scoped	not scoped
2-5	Capability for providing project-specific customizations	Via application re-deployment	not scoped	full on-the-fly change
2-6	Semantic accuracy of implemented CDM identical to specified CDM	often sacrificed for efficient implementation	often sacrificed for efficient implementation	no implement. performed, disjoints for basic logic
3-1	Support for product structure definition	partial, no product variants	currently not available, but realizable	currently not available, but realizable
3-2	Support for requirements definition	yes	currently not available, but realizable	currently not available, but realizable
3-3	Support for operational design definition	yes	currently not available, but realizable	currently not available, but realizable
3-4	Support for system architecture definition	yes	currently not available, but realizable	currently not available, but realizable
3-5	Support for system verification definition	yes	currently not available, but realizable	currently not available, but realizable
3-6	Support for system property definition	partial, no uncertainties	currently not available, but realizable	currently not available, but realizable
3-7	Usage of execution data for system validation	no	no	realizable
3-8	Existence of a mechanism for capturing and applying operational knowledge	basic, hard-coded consistency checks	not scoped	yes
4-1	Compatibility to MDA and EMF	yes	demonstrated	no

5.3 Concluding on the Analysis

It becomes evident that 10-23 in its implementation with RangeDB does currently not fully support all requirements that were defined on the data modeling approach in the MBSE context. The CDM itself already encompasses most of the required domain concepts, but some need extension. Other concepts, such as the knowledge capture

mechanism, are not adequately scoped. In the group of model specification capabilities, the 10-23/RangeDB approach does not fulfil many of the requirements, as most of these are not scoped by the CDM's specification language UML. Another shortcoming lies in the lack of a CDM definition process.

In turn, some of the approaches not yet established in the given industrial context offer helpful functionality for coping with several requirements. ORM 2 offers the constraints for providing the required semantics to CDMs in the MBSE context. OWL 2 offers the functionality required for fulfilling the requirements on multiple characterization of system model elements, as well as what is required for collecting knowledge across a variety of projects and applying them on other systems that will be designed in the future. Furthermore, OWL 2 allows the flexible adaption of a CDM during runtime, offering sufficient support for the activity of tailoring.

5.4 Improvement Approach

Several points in the areas of specification language, specification procedure, and activity support have been identified where currently available technologies and concepts do not fully address the requirements of the MBSE process. The CDM itself, its specification language, and specification procedure represent three meta-artefacts of the SM that have a significant impact on its content and functionality. These three meta-artefacts will be improved in order to enhance the overall utility of the SM, addressing all defined requirements, and enabling more extensive SM data exploitation. The approach is outlined in Figure 5.1.

The hypothesis behind the proposed approach is that improving the semantics of the CDM modeling language will improve the semantics of the CDM, and consequently the semantics and utility of the SM. Furthermore, employing a procedure for deriving the CDM's structure in a prescriptive manner from engineering data will improve the proximity of the CDM to actual engineering processes. Aligning the CDM content to currently required needs will also improve the utility of the system model in the MBSE process.

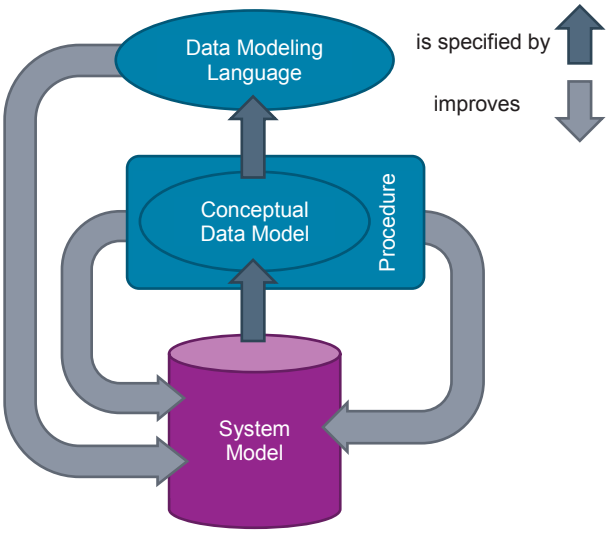


Figure 5.1: System Model improvement strategy

As the analysis has shown that, while there is no silver bullet that satisfies all necessary requirements at once, the possibility exists to fulfil each requirement by employing the technology best suited for dealing with it. This involves developing a procedure for deriving a CDM in a bottom-up approach, inspired by a Fact-Based methodology, such as CSDP. For fulfilling the requirements on the CDM's specification language, aspects from Ecore, OWL and ORM 2 will be brought together. In order to cope with further requirements on the MBSE process, the existing 10-23 CDM will be updated utilizing both language and procedure.

The following three chapters will deal with these three improvements, starting with designing a new data modeling language, followed by a modeling procedure, and going into design of the CDM.

6 The Semantic Conceptual Data Modeling Language

The analysis in 5.2 made evident that no approach is able to fulfil all requirements on its own. For some requirements, no fulfilment by any of the examined approaches is provided. However, the hypothesis is that a combination of technologies from the analyzed approaches, together with functionality specifically developed for addressing uncovered points, can fulfil all requirements. For this purpose, an analysis is presented that evaluates different conceivable language architectures, and how well these are able to fulfill the defined requirements.

The first point to consider is the language used for describing the CDM. The CDM's description language has the largest impact on the approach used to deal with engineering data, as it directly defines concepts available on M1/CDM level, in turn influencing the functionality available in the SM on M0 level.

In order to converge on the most suitable language architecture, an in-depth analysis of the characteristics of each of the examined approaches is performed initially. This analysis examines the characteristics of each language to considerable depth, and puts them into context with each other. This gives an idea of where exactly characteristic strengths and weaknesses of the languages are situated, and where the semantics of language concepts, although seemingly identical, differ greatly (section 6.1). Based on this analysis, different architectures are discussed and traded against each other (section 6.2), selecting one architecture that fulfils all requirements. The selected language design is then described in section 6.3, while 6.4 differentiates the design from existing work. Consequently, this chapter answers the second research question:

(RQ2) What is an appropriate language design for satisfying the requirements on domain data specification?

The three sections dealing with language design form new contributions. The analysis part picks up on the properties usually associated with each of the language, while breaking new ground by contrasting properties with each language that are not traditionally associated with them, with the aim of making visible every conceivable bit of

functionality. Furthermore, while being broader than the existing analysis, it also goes into further depth compared to previously published works. The section dealing with conceptualization of the language deals with selecting the best way to enable functionality from each of the examined languages on a single SM, also breaking new ground. Finally, the language itself is a new contribution on its own.

6.1 Differences between Data Model Specification Languages

The analysis performed in 5.2 deals with needs resulting from the embedding space engineering process, forming a high-level overview on available technologies to get an understanding of the shortcomings. This section on the other hand analyzes representative examples from the three identified language groups, forming a detailed picture on their fundamental properties and differences. This result of this analysis is then used in the next section to choose a suitable language design.

As analysis subject representing the first approach, Ecore is selected. This is done for numerous reasons. On the one hand, Ecore has extremely well defined semantics inside its framework, EMF (The Eclipse Foundation, 2016c), mitigating the ambiguous or rather not thoroughly defined semantics of UML (Evans & Kent, 2003). On the other hand, most of the UML concepts used in the description of data in 10-23 can be represented by language concepts of Ecore (ESA, 2011a). Third, Ecore provides effective and efficient means to provide an implementation of a CDM in an automated fashion, also forming a key element of the RangeDB 10-23 implementation. Lastly, compatibility to EMF has been formulated as a key requirement (REQ-4-1), so this should be an important consideration for the language design.

As candidate from the fact-oriented world, ORM 2 is selected and as candidate from the ontology world OWL 2 is selected, both for the reasons outlined earlier in 5.2.2.

These languages are compared regarding a variety of characteristics. The characteristics have been derived from numerous publications that deal with these languages on a very detailed level (W3C, 2006; Kiko & Atkinson, 2008) and form a significant evolution of an analysis performed earlier (Hennig, et al., 2016a). The analysis involves comparing central characteristics of the languages and their semantics, as well as evaluating differences in class modeling, property modeling, instantiation, and reasoning capabilities.

6.1.1 Central Characteristics

Initially, central aspects of the languages are compared. These aspects involve elaborating on and comparing language core concepts, the semantics inherent in the description of M1 models, and the M0 models consistency semantics, among others.

6.1.1.1 Core Concepts

Each of the three languages revolves around similar core concepts. Ecore uses the *EPackage* for partitioning the model, and uses *EClasses* as the main classifier. *EClasses* are refined using *EReferences* for property relations with an *EClass*, and *EAttributes* for properties with an *EDataType*. Instances are described using *EObjects*, which are typed by exactly one *EClass*. (The Eclipse Foundation, 2016c)

ORM 2 uses the *Entity Type* as main class concept, with *Value Types* being the data type concept. Class properties are defined by the *Roles* that *Entity Types* can play, which are connected via *Fact Types*. The instance concept is represented by the *Entity*, which is instance of exactly one *Entity Type*. (Halpin & Morgan, 2008; FBM WG, 2014)

In OWL 2, *Ontologies* are used to partition modeled information. This information is described using *Classes*, *Object Properties* for relations between *Classes*, and *Datatype Properties* for relations between Classes and *XSD Datatypes*. Further refinement of any concept in the *Ontology* is realized through *Annotation Properties*. Instances are represented with the concept of *Individuals* (Allemang & Hendler, 2011; W3C, 2012a).

While the vocabulary differs in some cases between each of the three languages, the availability of core concepts is identical. All three languages use a class concept to describe terminological information, with relations existing either between classes, or between classes and datatypes. This information is instantiated with an instance concept.

This analysis will try to keep a neutral vocabulary for comparison, using the terms *Class*, *Relation*, and *Instance* when talking about the three languages in a generic manner.

6.1.1.2 Containing Base Construct

The container for the body of information represented by the model is called *Model* in Ecore and ORM 2. The container is represented by the *Ontology* itself in OWL 2 (Halpin & Morgan, 2008; W3C, 2012a; The Eclipse Foundation, 2016c).

6.1.1.3 Main Abstraction Levels

For Ecore-based models, the abstraction levels can be clearly mapped to MOF, with Ecore CDMs residing on level M1, and their UMs, consisting of *EObjects*, residing on M0 (OMG, 2015a). The same applies to ORM 2 models (Lemmens, et al., 2007). For OWL 2 ontologies, the M1 level is often called *TBox*, containing the terminological part of a model, while the M0 level is called *ABox*, making up the assertional component of a model (Allemang & Hendler, 2011). For all three languages, these abstraction levels are essentially identical.

6.1.1.4 General M1 Model Semantics

However, the languages begin to differ when it comes to the overall semantics of the model on the M1 level. In the case of Ecore, the M1 level contains the description of a software system, and nothing more (The Eclipse Foundation, 2016c). While the software system may be used to describe objects that are not software-related at all, the meaning of described concepts is directly translated into software-related artefacts. ORM 2 models describe *Fact Types* that can be played by *Entity Types* and *Data Types* inside a body of knowledge (Halpin & Morgan, 2008). Both approaches essentially define the possible and allowed populations of their respective M0 models.

The OWL 2 *TBox* describes terminological knowledge from a specific viewpoint (W3C, 2006; Kiko & Atkinson, 2008). It is entirely possible that *ABox* populations exist that are not scoped by its corresponding *TBox*. However, if data in an *ABox* matches certain descriptions in its *TBox*, *Individuals* may be classified as belonging to specified *Classes*.

6.1.1.5 Semantic Context of M1 Model

For each of the examined languages, the models residing on M1 level have a specific purpose, based on their context. While the Ecore language is used in the context of software engineering, OWL 2 on the other hand is prominently used within the Semantic Web. Consequently, the meaning of the M1 models differs between the three languages.

The semantic context of the Ecore language is very well defined inside its framework EMF (The Eclipse Foundation, 2016c), as essentially all generated code is directly or indirectly driven by the Ecore model. Outside this framework however, an Ecore model does not convey formal semantics in a logical or mathematical sense.

The semantics of ORM 2 models are more generally applicable, as they describe the types of elementary facts that can exist between the *Entity Types* described in the M1

model (Halpin & Morgan, 2008). The scope of these facts is valid for the domain, largely influenced by the domain's vocabulary that defines the M1 model.

OWL 2 semantics are the most well-defined of the three languages, as OWL 2 is based upon DL (Allemang & Hendler, 2011). The reliance on DL provides a mathematical-logical meaning (Baader, et al., 2007) to the concepts of both M1 and M0 models, where *Classes* follow a particular set theory-based approach to which *Individuals* are allocated.

6.1.1.6 World Assumption

The world assumption of an Ecore model follows the classical approach of object-oriented models, where the model uses the CWA. Models using this principle assume that every bit of information of relevance to the model is contained by it and that there is no relevant information not contained in the model. This principle leads to the behavior that information not contained in the model is regarded as false. This means that, for example, if a query asking for the number of batteries is executed on a closed world model of a *Spacecraft* that has one *Battery* modeled, then the query would return *one*. As all information about existing *Batteries* is contained by the *Spacecraft* model, the existence of *exactly one Battery* can be proven with certainty.

ORM 2 is based on the CWA, except for the concept of *Unary Fact Types*, where the world assumption can be selected between CWA, OWA, and OWA with negation (FBM WG, 2014). In the CWA case, each unary fact not recorded is treated as *false*. In the OWA case, the facts recorded are *true*, while the facts not recorded are *unknown*. In the OWA with negation case, the set of unknown facts may be reduced by explicitly stating facts that are *not true*.

OWL 2 on the other hand is inherently based upon the OWA (W3C, 2006; Kiko & Atkinson, 2008; Allemang & Hendler, 2011), due to its primary domain of usage lying within the Semantic Web. The OWA is based on the notion that a model under consideration may only contain a small part of all the information existing about a specific subject, and that other knowledge bases may well exist that extend this information. In an open world scenario, the same query as above would return *at least one* as a result, as the existence of one *Battery* aboard the *Spacecraft* is known for sure, but others may also exist, that are just not represented in the model. In addition to this, a query asking for all *Spacecraft* that have one *Battery* on board would return *empty*, as it cannot be said for certain that the *Spacecraft* contains *only one Battery*. In order to get a definite result on this query, the model has to be closed down using specific axioms on M1 level.

6.1.1.7 Availability of Negation as Failure

The world assumption of a language has a profound impact on the way of working with the M0 model. One of the characteristics influenced by a language's world assumption is the availability of Negation as Failure. This concept states that a failure to derive a statement from a model does automatically mean that the statement does not hold in the world represented by the model. In the world of Ecore and ORM 2, Negation as Failure is available due to their reliance on the CWA, while for OWL 2, Negation as Failure is not supported, as it would interfere with the ability to decide on specific reasoning tasks.

6.1.1.8 M0 Model Consistency Semantics

Both Ecore and ORM 2 rely on using some form of constraining on M1 level to scope the amount of valid populations for their M0 models (Halpin & Morgan, 2008; The Eclipse Foundation, 2016c). In both cases, a violation of these constraints implies an invalid model, as some sort of defined boundary is violated. OWL M0 models, on the other hand, can by definition not be incorrect, as the information represented may make sense to the person who originally provided it. An OWL 2 M0 model however may be unsatisfiable in respect to the own M1 model if it violates the logics intrinsic to its definition.

6.1.1.9 Identification Schemes

In Ecore, the elements of M1 models are identified by their name and path inside the current model. Consequently, *EPackages* play a large role in partitioning the model, defining the path on which specific instances can be found. On M0 level, instances are uniquely identified by a *unique ID* (The Eclipse Foundation, 2016c).

In ORM 2, elements of the M1 model are also identified by their name and path. In addition, an identification scheme such as identification via a specific integer, a specific string, or a combination of both has to be defined for each *Entity Type* and *Value Type* by the user (Halpin & Morgan, 2008), enabling a custom instance identification approach to be used on M0 level.

In OWL 2, concepts on both M1 and M0 level are uniquely identified by an IRI (W3C, 2012a). The IRI contains a path to the *Ontology* itself, as well as the local name of the concept to be identified.

6.1.1.10 Name Assumption

Ecore, being based on classic object-oriented programming paradigms, relies on the *Unique Name Assumption* (UNA). The same can be said for ORM 2. This assumption says that model elements on M1 and M0 level that have a different identifier, which has to be unique, are treated as being distinct entities. OWL 2, on the other hand, uses the *Nonunique Name Assumption* (Allemang & Hendler, 2011) which states that, although model entities are identified with different names, they might represent the same thing in the real world, just with a different name, and perhaps having a different view on it.

6.1.1.11 Synonym Semantics

As Ecore and ORM 2 rely on the UNA, both languages do not have a concept for describing that different elements in the model are merely different descriptions for the same entity. OWL 2 on the other hand supports the use of synonyms or rather equivalent entities for *Classes*, *Properties*, and *Individuals* (W3C, 2012a). This enables the information to be represented that multiple *Classes*, although under different names, describe the same set of instances, that multiple *Properties*, although having different names, convey the same semantics, or that multiple *Individuals* represent the same real world object.

6.1.1.12 Concept Versioning Approach

The concepts defined in Ecore-based models can be versioned directly in the M1 model by assigning a version to packages through their URI (The Eclipse Foundation, 2016c). ORM 2 does not incorporate a dedicated internal versioning scheme directly in the M1 model. OWL 2 offers a number of different versioning concepts inside an o*ntology*. For instance, an *ontology* may be associated with a *versionIRI*, besides its own *ontologyIRI*. Furthermore, anything with an IRI, so essentially all central modeling constructs, can be annotated with versioning information. The *versionInfo* field can be used to specify information in prose about the current version of a concept, while *priorVersion* may be used to point to the IRI of the prior version of the concept. The *deprecated* property may be set if the concept is outdated, while *backwardCompatibleWith* and *incompatibleWith* can be used to point to an IRI that denotes a compatible or incompatible version of the same concept (W3C, 2012a).

6.1.1.13 Usage of Other M1 Models

Ecore can reference concepts from other Ecore-based models for reuse (The Eclipse Foundation, 2016c). ORM 2 on its own does not have a dedicated mechanism to reference other ORM-based models. In OWL 2, other ontologies can be imported, either locally or via URL, enabling the referencing of all imported concepts (W3C, 2012a).

6.1.1.14 Separation of M1 and M0 Levels

In Ecore and ORM 2, both main abstraction levels are strictly separated. Classes and properties are defined on M1 level, and only after the M1 model has been implemented in an application can instances be created on M0-level.

In OWL 2, *Classes* and *Individuals* can exist in the same *Ontology*. This means that, although *TBox* and *ABox* are regarded as containing different types of information, both are not strictly allocated to M1 and M0 levels. In OWL 2 Full, it is even possible to specify that a *Class* is the same thing as an *Individual* (Kiko & Atkinson, 2008).

6.1.1.15 Differentiation of Model Development and Usage

In consequence, in both the Ecore and ORM 2 context, a strict differentiation has to be made between M1 model development time and M0 model runtime, as no instances can exist before the model is implemented. Iterations on the M1 model require a new iteration on its implementation.

In OWL 2 ontologies, these two periods are not technologically differentiated and the *TBox* may still change during *ABox* runtime, as more and more information about the real world, or its *TBox*-representation, becomes available (W3C, 2006).

6.1.1.16 Summary of Central Characteristics

Table 6.1: Summarized comparison of central language characteristics

Characteristic	Ecore	ORM 2	OWL 2
Core Concepts	EPackage, EClass, EReference, EAttribute, EDataType, EObject	Entity Type, Value Type, Fact Type, Role, Entity	Ontology, Class, Object Property, Datatype Property, Annotation Property, Individual
Containing base construct	Model		Ontology

Characteristic	Ecore	ORM 2	OWL 2
Main abstraction levels	CDM level/M1 level/terminological level and instance level/M0 level/assertional level		
General M1 semantics	Description of a software system, defining possible and allowed populations	Description of fact types about a body of knowledge, defining possible and allowed populations	Description of terminological knowledge about a domain, valid for own viewpoint
Semantic context	Semantics fixed inside framework	Semantics fixed as factual statements about things	Semantics fixed generally by mathematical-logical constructs
World Assumption	Closed World	Closed World, Unary Fact Types flexible	Open World
Availability of negation as failure	available		not available
M0 model inconsistency semantics	Instance model may be incorrect		Instance model not necessarily incorrect, just unsatisfiable regarding own view
Model element identification scheme	Model-path for M1 model, generated unique ID for M0 model	Model-path for M1 model, user-defined ID for M0 model	Unique IRI
Name Assumption	Unique Name Assumption inside current scope		Nonunique Name Assumption
Synonym semantics	No synonym semantics		Equivalent classes, equivalent properties, equivalent individuals
Model versioning approach	Per package through URI	Not scoped	For ontology versionIRI, versionInfo; for classes deprecated, priorVersion, backwardCompatibleWith, incompatibleWith
Usage of other models	Loading of other model and equivalent usage of other model elements	not scoped	Ontology imports and equivalent usage
Separation of M1 and M0 levels	Strict, two separate models		May be in same model, Individuals may be identical to Classes
Differentiation of model development and usage	Strict differentiation of development-time and run-time		No differentiation of development-time and run-time

6.1.2 Class Characteristics

In line with the general model semantics, the semantics of the *Class* construct differs equally between the three languages. In the Ecore case, *EClasses* define types of *EObjects*, with a large amount of code generation semantics involved in each class (The Eclipse Foundation, 2016c). In the ORM 2 world, *Entity Types* form possible types of *Entities*, without any meaning towards code generation (Halpin & Morgan, 2008). In the OWL 2 case, *Classes* define sets of *Individuals* in a mathematical way (W3C, 2006; Allemang & Hendler, 2011).

In all three cases, classes may form a taxonomy, with *subclasses* inheriting the properties of its *superclasses* (The Eclipse Foundation, 2016c; FBM WG, 2014; Allemang & Hendler, 2011).

However, things get different when looking at the capabilities to handle inheriting properties. While in the Ecore world, it is not possible to override properties inherited by a given superclass, this can be realized on implementation level, as Java explicitly supports this behavior (The Eclipse Foundation, 2016c). In ORM 2, this is not foreseen. With OWL 2 ontologies, overriding any of the inherited properties is also not possible. Overriding a property of a superclass would mean that the subclass is now not a member of the superclass anymore, as it does not exactly exhibit its properties. This would be in violation of one of the foundational notions of OWL 2 (W3C, 2006), so this functionality is explicitly excluded.

ORM 2 supports the concept of *Independent Classes*, which describes instances that can exist without taking part in any mandatory roles, i.e. exhibiting properties that are defined as mandatory (FBM WG, 2014). Ecore does not support this concept. In OWL 2, this principle is not explicitly mentioned, but fully covered by the OWA.

In object-oriented modeling, the notion of A*bstract Classes,* which cannot be instantiated, but play an important role for abstracting common characteristics of the M1 model, is frequently used. As such, it is fully covered by Ecore (The Eclipse Foundation, 2016c). The functionality is also supported by ORM 2 with the *Exlclusive-Or subtyping constraint*. In OWL 2, this behavior is again excluded, as all subclasses of a *Class* are by definition also part of the set defined by their superclasses (W3C, 2006; OMG, 2014c).

OWL 2 has the notion of an *Anonymous Class*. *Anonymous Classes* play a key role in defining *Class Axioms*, where they are used to define not explicitly modeled sets of *Individuals* that can be treated as a *Class*. This applies to, for example, *intersections of Classes and unions of Classes*, where the set defined by their intersection or union forms the *Anonymous Class* (W3C, 2012a). Neither Ecore, nor ORM 2 have or are in need of a comparable construct.

In Ecore, class variables are described by *EStructuralFeatures*, with *EAttributes* being typed by an *EDataType*, and *EReferences* being typed by another *EClass* (The Eclipse Foundation, 2016c). In ORM 2, a similar thing is achieved by assigning roles to *Entity Types*, which can be played between numerous *Entity Types*, or between *Entity Types* and *Value Types* (FBM WG, 2014). OWL 2 does not support any notion of class variables, as an entirely different concept is used with *Necessary Conditions* of a *Class*, defining the properties an *Individual* has to have in order to be a member of the *Class* (W3C, 2006).

OWL 2 has a strong emphasis on set-theoretic aspects (Baader, et al., 2007), more than the other two languages under consideration. While some of these are somewhat implicitly covered, others are not scoped at all. One of these aspects is the definition of *Disjoint Classes*, which is explicitly supported by OWL 2, and makes up an important mechanism for ensuring the logical consistency of ontologies (W3C, 2012a). In Ecore, a similar construct is given by two subclasses of an abstract superclass (Kiko & Atkinson, 2008), although there is no possibility to instantiate an *EObject* that is an instance of both. In ORM 2, a similar construct is offered by the *Exclusive-Or subtype constraint* (FBM WG, 2014).

Class equivalency is not a concept covered by either Ecore or ORM 2. OWL 2 offers an *Equivalent Classes Axiom* to denote that several classes represent the same concept, just under different names (W3C, 2012a).

Class intersections can be emulated by both Ecore and ORM 2 by producing a class that inherits from the classes that should be intersected (Kiko & Atkinson, 2008). OWL 2 has an *ObjectIntersectionOf Class Expression* that can be used in *SubclassOf* and *EquivalentClasses Class Axioms* (W3C, 2012a).

In Ecore and ORM 2, a union of classes can be implied by having a class A and classes B and C that both subtype class A. OWL 2 has a dedicated *Object Union Of Class Expression* that can be used to express unions (W3C, 2012a).

The same is true for class complements, which are not covered by Ecore and ORM 2, but are able to be modeled in an OWL 2 ontology with the *ObjectComplementOf Class Expression*.

ORM 2 supports the concept of *Object Cardinality*, which expresses that of one class, only one *Instance* or rather *Entity* can exist at one point in time (FBM WG, 2014). This concept is not covered in Ecore, and also not covered in OWL 2, as it stands in contrast to the OWA.

Table 6.2: Summarized comparison of language class characteristics

Characteristic	Ecore	ORM 2	OWL 2
Class semantics	EClasses are types of EObjects, code generation semantics	Entity Types are types of entities	Classes are sets of individuals
Inheritance behavior	Classes can be subclasses of multiple superclasses and inherit their properties		
Overriding of behavior and properties	Not scoped by Ecore, but may be done in implementation	Not scoped	Explicitly excluded, as all subclasses are per definition members of the superclass and cannot exhibit behavior or properties different from it
Independent classes	Not explicitly, but possible as long as property assignments permit	Explicitly	Not explicitly, but covered by OWA
Abstractness of classes	Supported, an abstract class may not have any instances	Through Exclusive-Or Subtyping Constraint	Explicitly excluded, as all subclasses are per definition part of the set scoped by their superclass
Definition of anonymous classes	No class membership possible besides for defined classes		Anonymous classes key construct used for referencing not explicitly defined sets of individuals
Notion of class variables	Through EStructuralFeatures	Through roles played by Entity Types	Not scoped, entirely different concept with necessary conditions
Class disjunction	Disjoint classes implicitly given by subclasses of an abstract superclass	Exclusive and ExclusiveOr Subtyping Constraint	DisjointClasses axiom
Class equivalency	Not scoped		EquivalentClasses axiom
Class intersection	Not scoped, emulation by producing a new class that inherits from the two intersecting classes		Class to exhibit SubclassOf or EquivalentClasses axiom with expression ObjectIntersectionOf
Class union	Via subtyping		Class to exhibit SubclassOf or EquivalentClasses axiom with expression ObjectUnionOf

Characteristic	Ecore	ORM 2	OWL 2
Class complement	Not scoped		Class to exhibit SubclassOf or EquivalentClasses axiom with expression ObjectComplementOf
Constraint for class cardinality	Not directly, workaround via containment reference cardinality	Object Cardinality Constraint	Not scoped, incompatible with OWA

6.1.3 Property Characteristics

When expressing that specific classes exhibit specific properties in terms of attributes and references, the three languages under consideration also behave differently. In the case of Ecore, properties are defined locally for each class, with a *name* and a *type*. Depending on the type of the property, it either comes down to an *EAttribute*, or an *EReference* (The Eclipse Foundation, 2016c). This means that with the definition of the property, it is already assigned to an *EClass*. In ORM 2, *Fact Types* assume the place of properties, as they group together the roles that that can be played by *Entity Types* and *Value Types*, connecting these concepts. Consequently, property definition follows a more global approach, as they can exist without being assigned to any *Entity Type* (FBM WG, 2014). In OWL 2, properties are defined globally and are regarded as first-order entities that can be related with each other (Allemang & Hendler, 2011). The definition of an *Object Property*, connecting two classes, or a *Data Property*, connecting a *Class* and a *Literal*, does not imply that it actually has to be used by any *Class*.

OWL 2 offers the capability to define *necessary and sufficient conditions* that express what characteristics an *Individual* is required to have, and what characteristics are sufficient for it to have in order to be a member of a specific *Class*. One way to express such conditions is to define a *domain* and a *range* for a *Property* (W3C, 2012a). Each *Individual* that takes part in the relation as defined in the *Property* will be inferred to be a member of a specific *Class*. For instance, if a *Property orbits* is defined that has *Satellite* as *domain* and *Planet* as *range*, and if two *Individuals* exists that have this relation asserted, it can be inferred that one *Individual* is of type *Satellite*, and the other is of type *Planet*. No such concept is offered by Ecore, nor ORM 2.

In OWL 2, *Properties* can form a taxonomy, meaning that all P*roperty Assertions* for an *Individual* also imply assertions of their super-properties (W3C, 2012a). Property taxonomies are not scoped by Ecore, as properties are not considered as being first-order entities. The same applies to ORM 2.

The semantics of property assignments to classes also differ between the three languages. In the case of Ecore, *EStructuralFeatures* in terms of *EReferences* and *EAttributes* define the possible variable populations for *EObjects*, with constraints on the multiplicity of populations. Furthermore, the assignment of *EStructuralFeatures* has implications on code generation (The Eclipse Foundation, 2016c). In the case of ORM 2, this concept involves *Roles* that can be played by *Entity Types* and *Value Types*, with a strong emphasis on the constraints on these roles (Halpin & Morgan, 2008). In the case of OWL 2, property assignment to a *Class* is achieved using *SubclassOf* and/or *EquivalentTo Class Axioms* being made up of a wide range of *Restrictions*. The meaning of these axioms is that they represent the conditions *necessary* for *Individuals* to be a member of a specific *Class* (*SubclassOf*), and the conditions that are *necessary and sufficient* in order to be a member of a specific *Class* (*EquivalentTo*).

In all three languages, property assignments must be typed. In Ecore, *EAttributes* must be typed by an *EDataType*, and *EReferences* have to be typed through an *EClass* (The Eclipse Foundation, 2016c). For the *Fact Types* in ORM 2, their predicates need to be connected to either *Entity Types*, or *Value Types* (FBM WG, 2014). In the case of OWL 2, *Class Axioms* consisting of *Property Expressions* have to reference exactly one *Data Property* or *Object Property*.

Ecore offers the possibility to constrain the multiplicity of property assignments using the *lowerBound* and *upperBound* attributes of *EReferences* and *EAttributes* (The Eclipse Foundation, 2016c). In ORM 2 models, property multiplicity is defined by using a combination of *Mandatory Role Constraint*, *Uniqueness Constraint*, and *Role Cardinality Constraint* (Halpin & Morgan, 2008). In OWL 2 ontologies, this is realized through assigning *SubClassOf* axioms with the expressions *MinCardinality*, *ExactCardinality*, or *MaxCardinality*. However, OWL's OWA makes these expressions difficult to be evaluated, as only information that exceeds the number used in the multiplicity assignment gets evaluated as being inconsistent. Cases where information is missing from the ontology are not flagged as an inconsistency (Kiko & Atkinson, 2008).

By default, an assignment of a property to an *EClass* is interpreted as a *necessary condition*. *Necessary and sufficient conditions* on property assignment level are not scoped. The same is true for the ORM 2 case. In the case of OWL 2, *necessary conditions* are modeled using *Class* to exhibit *SubClassOf* axioms with the expressions *HasValue*, *MinCardinality*, *ExactCardinality*, or MaxCardinality (Allemang & Hendler, 2011). *Necessary and sufficient conditions* are expressed using *EquivalentClasses* axioms using the expressions above, plus the *SomeValuesFrom*, and *AllValuesFrom* expressions (Allemang & Hendler, 2011).

The Ecore language supports the specification of unary properties by using *EAttributes* with type *Boolean*. Binary properties are supported through the other *EAttributes*

and *EReferences*, while n-ary properties are not scoped by the language (The Eclipse Foundation, 2016c). ORM 2 scopes *Unary Fact Types*, *Binary Fact Types*, and *N-ary Fact Types* (FBM WG, 2014). OWL 2 scopes unary properties with the assignment of *Data Properties* with range *xsd:Boolean* and binary properties with the other *Data Properties* and *Object Properties*. N-ary properties are also not scoped (W3C, 2012a).

ORM 2 allows properties or rather *Fact Types* to be *objectified*, meaning that the property itself becomes a class that can also play roles (Halpin & Morgan, 2008). This concept is neither covered by Ecore, nor by OWL 2.

Mandatory properties are expressed in Ecore using *EAttributes* and *EReferences* with a *lowerBound* greater than 0 (The Eclipse Foundation, 2016c). ORM 2 uses a dedicated concept for this with the *Simple Mandatory Role Constraint* (Halpin & Morgan, 2008). In OWL 2, the same can be specified with a *Class Axiom* involving a *Cardinality Expression* denoting more than 0 property assertions, but this cannot be directly enforced due to the OWA (Kiko & Atkinson, 2008).

OWL 2 ontologies may use the concept of *Functional Properties* with the *Functional Property Axiom*, constraining its multiplicity to either 0 or 1. This axiom also makes the property participating in the function of uniquely identifying *Individuals* conclude that two *Individuals* with the same value for their functional property are in fact the same *Individual* (Kiko & Atkinson, 2008). This also behaves as a *necessary and sufficient condition* for inferring knowledge about an *Individual*. For this concept, only the necessary part, i.e. multiplicity of either 0 or 1, is scoped by both Ecore and ORM 2, not the sufficient part.

In Ecore, properties, more specifically *EReferences*, can be made unique by using the unique attribute. This constrains the possibility for population of this specific *EReference* to each *EObject* to only occurring once (The Eclipse Foundation, 2016c). ORM 2 has a dedicated concept for this with the *Internal Uniqueness Constraint* that can be applied to any *Binary* or *N-ary Fact Type* (FBM WG, 2014). In OWL 2, *Functional Properties* can be used to denote some kind of unique population for the property, but the evaluation behaves differently due to the Nonunique Name Assumption. In this case, *Individuals* exhibiting similar populations will be marked as equivalent, instead of inconsistent.

Ecore relies heavily on explicit, unique hierarchies, using the containment property of *EReferences* that conveys a kind of composition semantics (The Eclipse Foundation, 2016c). Containment properties are neither scoped by ORM 2, nor by OWL 2.

Reference chains are scoped by neither Ecore, nor ORM 2. In OWL 2, *Object Property Chains* can be defined, stating that a defined chain across several *Object Properties* between several *Individuals* actually implies the existence of a specific, additional *Object Property* (W3C, 2012a).

Enumeration properties are supported by all three languages. In Ecore, this is realized with an *EAttribute* that is typed by an *EEnum*, containing a number of *EEnumLiterals* (The Eclipse Foundation, 2016c). For ORM 2 models, this is realized through participation of an *Entity Type* in a *Binary Fact Type* with a *Value Type*, that has an *Object Type Value Constraint* associated with it, containing the valid enumeration values (Halpin & Morgan, 2008). In OWL 2, enumerations can be realized by using a *Class Axiom* with a *Data Property Restriction* including the expression *DataOneOf* to a set of *Individuals* (Allemang & Hendler, 2011).

The information that two properties are actually equivalent cannot be specified in Ecore models. This concept is also not scoped by ORM 2 and is not to be confused with an *Equality Constraint* between roles, which conveys different semantics. OWL 2 supports the specification of *Equivalent Data Property* and *Equivalent Object Property* axioms, specifying that two properties convey identical meaning, just under different names (Allemang & Hendler, 2011).

Property inverses can be specified on assignment level in Ecore with the *eOpposite* property of an *EReference* (The Eclipse Foundation, 2016c). In ORM 2, this can also be done on assignment level with names assigned to both predicates in a *Binary Fact Type* between two *Entity Types* (Halpin & Morgan, 2008). In OWL 2 ontologies, the *Inverse Object Property Axiom* can be used to convey the semantics that, if one of these properties is set, the inverse property of the other participating *EClass* is also required to be set (Kiko & Atkinson, 2008).

Setting *reflexivity, transitivity, symmetry, and acyclicity* for a property is not scoped by Ecore. Implicitly, *EReferences* with *containment* are *acyclic*, but this cannot be specified separately. For ORM 2 models, *Ring Constraints* with each of these characteristics can be assigned to specific *Fact Types* (FBM WG, 2014), conveying the semantics that the *Fact Type* is only correctly populated if the specified conditions of the *Ring Constraint* are satisfied. OWL 2 has a *Reflexive Property Axiom*, *Transitive Property Axiom*, and *Symmetric Property Axiom*, but these convey the meaning that, if one condition holds, then another condition also has to hold (Kiko & Atkinson, 2008). An acyclic property also cannot be expressed in OWL 2.

Other constraints can also exist between properties. These include *Value Comparison Constraints*, denoting, for example, that the value of one property must always be greater than the value of another property. *Object Type Value Constraints* constrain the possible property values directly on property definition level, while *Role Value Constraints* do the same on property assignment level. *Subset Constraints* imply that a specific property can only take the values that are already set in its superset property, while an *Equality Constraint* means that values must always be equal. An *Inclusive-Or Constraint* implies that at least one of the properties taking part in it must have a value. An *Exclusion Constraint* between properties means that the values of all in-

volved properties have to be mutually exclusive, while an *Exclusive-Or* constraint implies that for each property some value must exist, but none of the values in one property can be set for the other properties. ORM 2 supports the definition of all of these constraints between properties (FBM WG, 2014), while none of these is supported by Ecore. OWL 2 does not support *Value Comparison*, but supports *Object Type Value Constraining* with *Data Ranges* and *DataHasValue Expressions*. The *Subset Constraint* is not scoped due to incompatibility with the OWA. *Equality Constraints* are not scoped and not to be confused with *Equivalent Properties*, as these convey different semantics. The *Inclusive-Or Constraint* is also not scoped, as is the *Exclusive-Or constraint*, as both are, again, not in accordance with the OWA. The *Exclusion Constraint* evaluates in a way similar to the *Disjoint Properties Axiom*.

Table 6.3: Summarized comparison of language property characteristics

Characteristic	Ecore	ORM 2	OWL 2
Property definition approach	Properties defined locally for a class	Fact Types defined independently of Entity Types	Globally as first-order entities
Property definition necessary and sufficient conditions	Not scoped		Domain and range for inference of instance class membership (sufficient conditions)
Property definition taxonomy	Not scoped, properties are not first order entities		Properties can form a taxonomy with semantic implications (super-property includes all sub-properties)
Property assignment semantics	Possible properties of objects, constraints on these properties, code generation semantics	Possible roles that objects may play, constraints on these roles	Axioms and restrictions, defining necessary and sufficient conditions for individuals in order to be members of a class
Property assignment typing	Property assignments must be typed with an EClass or an EDataType	Predicates of Fact Types must be assigned roles to Entity Types or Value Types	Axioms consisting of property expressions must reference a property
Property assignment multiplicity	lowerBound, upperBound of EReference and EAttribute	Combination of Mandatory Role Constraint, Uniqueness Constraint, Role Cardinality Constraint	Class to exhibit SubClassOf axiom with expression MinCardinality, ExactCardinality, or MaxCardinality. However OWA problematic for evaluation

Characteristic	Ecore	ORM 2	OWL 2
Property assignment necessary conditions	Property defined as a typed variable to a class	Property defined as a Role played in an Fact Type	Class to exhibit SubClassOf axiom with expression HasValue, MinCardinality, ExactCardinality, or MaxCardinality
Property assignment necessary and sufficient conditions	Not scoped		Class to exhibit EquivalentClasses axiom with expression SomeValuesFrom, AllValuesFrom, or HasValue
Assignment of unary properties	EAttribute with type Boolean	Entity Type playing a role in an Unary Fact Type	Data Property with range xsd:Boolean
Assignment of binary properties	EStructuralFeature	Binary Fact Type	Class to exhibit SubClassOf axiom with expression for property
Assignment of n-ary properties	Not scoped	N-ary Fact Type	Not scoped
Property objectification	Not scoped	Objectification of Fact types	Not scoped
Mandatory property	lowerBound of EStructuralFeature greater than 0	Simple Mandatory Role Constraint	Cardinality Expressions in Class Axioms, although evaluation limits due to OWA
Functional properties	Only the necessary part is scoped (max cardinality 1), not the sufficient part		FunctionalPropertyAxiom
Uniqueness of properties	unique for EReferences	Internal Uniqueness Constraint	FunctionalPropertyAxiom but different behavior due to NUNA
Containment references	containment for EReferences	Not scoped	
Reference chains	Not scoped		ObjectPropertyChain axiom
Enumeration properties	EAttribute typed with an EEnum	Participation in a Fact Type with a Value Type that has an Object Type Value Constraint	Class with DataPropertyRestriction including DataOneOf to a set of individuals
Property equivalence	Not scoped	Not scoped, not to be confused with Equality Constraint	EquivalentDataProperty, EquivalentObjectProperty axioms

Characteristic	Ecore	ORM 2	OWL 2
Property inverse	On assignment level, eOpposite property of EReference	On assignment level, Binary Fact Type with names assigned to both predicates	InverseObjectProperty axiom
Property reflexivity	Not scoped	Fact Type with Ring Constraint with reflexivity	ReflexiveObjectProperty axiom
Property transitivity	Not scoped	Fact Type with Ring Constraint with transitivity	TransitiveObjectProperty axiom
Property symmetry	Not scoped	Fact Type with Ring Constraint with symmetry	SymmetricObjectProperty axiom
Property acyclicity	Implicitly for EReferences with containment	Fact Type with Ring Constraint with acyclicity	Not scoped due to OWA, cycles may lead to inconsistent ontology
Property constraint value comparison	Not scoped	Value Comparison Constraint	Not scoped
Property constraint on value range (definition level)	Not scoped	Object Type Value Constraint	Data Ranges
Property constraint on value range (assignment level)	Not scoped	Role Value Constraint	Class to exhibit SubClassOf axiom with DataHasValue expression
Property constraint subset	Not scoped	Subset constraint	Not scoped, as incompatible with OWA. Subproperties with different semantics.
Property constraint equality	Not scoped	Equality Constraint	Property equality, but with different semantics
Property constraint inclusive-or	Not scoped	Inclusive Or Constraint	Not scoped
Property constraint exclusion	Not scoped	Exclusion Constraint	Disjoint properties
Property constraint exclusive-or	Not scoped	Exclusive Or Constraint	Not scoped, incompatible with OWA

6.1.4 Instance Characteristics

Regarding instantiation of the concepts defined on M1 level, Ecore and ORM 2 behave similarly, while OWL 2 follows an entirely different instantiation philosophy.

In Ecore, an *EObject* is always typed by exactly one *EClass* (The Eclipse Foundation, 2016c). An *EObject* cannot exist without being assigned a specific type, and it cannot have more than one type. The same applies to ORM 2, where an *Entity* is always typed by exactly one *Entity Type* (FBM WG, 2014). In OWL 2 on the other hand, *Individuals* may exist that are typed by no *Class* at all, or by multiple *Classes* at the same time (W3C, 2012a). This marks a clear break from traditional object-oriented principles, enabling different behavior required in order to cater to the openness of the Web (Allemang & Hendler, 2011). In this approach, any data from an *ABox* can be integrated with the own *TBox*, regardless of whether it matches no, one, or several *Classes* that are defined there. In addition, this enables the same *Individual* being classified differently, but simultaneously, in different ontologies, that are defined by different stakeholders.

In addition to the class-instance-relationship, the instantiation behavior also differs. While in Ecore and ORM 2, the class structure is fixed during development of the M1 model and unable to be changed during M0 model runtime, OWL 2 enables *Individuals* to change their *Class* membership after the M0 model has been instantiated (W3C, 2006).

In Ecore and ORM 2, instances are always considered as distinct entities. OWL 2, due to its Nonunique Name Assumption, treats different *Individuals* as potentially representing the same object in the real world, until explicitly stated otherwise (Allemang & Hendler, 2011). For specifying these facts explicitly, the possibility to assert *Same Individuals* and *Different Individuals* on M0 level is provided (W3C, 2012a).

When setting properties, Ecore and ORM 2 follow the approach of assigning values to the properties defined on M1 level. In OWL 2, this is accomplished by using *Object Property Assertions* and *Data Property Assertions* that involve information about the actual property to be set, about its *value*, and about the *Individual* that shall exhibit the property (W3C, 2006; W3C, 2012a).

In Ecore and ORM 2, values can only be assigned to the properties defined in the corresponding M1 model. In OWL 2, *Individuals* may exceed the properties defined by the *Classes* they have as *type*. Also, *Individuals* may not exhibit all properties, or even exhibit no properties at all of the set specified by their corresponding *Classes*. Furthermore, due to the OWA, OWL 2 offers the possibility to specify *Negative Object Property Assertions* and *Negative Data Property Assertions*, conveying the information that specific properties are known to not hold for the *Individuals* in question (W3C, 2006; W3C, 2012a).

Table 6.4: Summarized comparison of language instance characteristics

Characteristic	Ecore	ORM 2	OWL 2
Instance class membership approach	Instance is typed by exactly one class	Entity is typed by exactly one Entity Type	Individuals may be instances of no class at all or of multiple classes
Instance class membership behavior	Class membership of an instance strictly defined during development, cannot be changed during runtime		An individual's class membership may change at any point during runtime
Instance identity approach	Instances are always distinct entities		Individuals have to be assumed to be identical or distinct until explicitly stated or inferred otherwise
Instance equivalency	Not applicable due to UNA		SameIndividual, DifferentIndividuals
Property setting approach	Assignment of values to properties		Assignment of ObjectPropertyAssertion, DataPropertyAssertion
Property setting strictness	Class members must exactly conform to defined properties		Individuals may exist that exceed the properties defined by their asserted classes. Properties defined on a class may also be ignored
Negative property setting	Not applicable due to CWA		Negative Object Property Assertion, Negative Data Property Assertion

6.1.5 Reasoning Functionality

The languages under evaluation all allow some form of deriving new information based on information that is already in the M0 model, by using specific algorithms. While for some languages, this functionality is very basic, ontologies allow the derivation of complex logical relations on M1 and M0 level. The approach of inferring new information based on already existing information in the model is what is meant by reasoning in this context.

Regarding M1 model reasoning, OWL 2 enables the detection of implicit superclass-subclass relationships. This means that, although two *Classes* are defined separately from each other, one class may exhibit a subset of another *Class' Properties*. This fact will be highlighted by a reasoner, which infers this hierarchical relation. Furthermore, reasoners on OWL 2 models are able to highlight unsatisfiable *Class* definitions, where *Classes* are defined in a way where they can never be consistently populated, as

their definition contains logical contradictions (Allemang & Hendler, 2011). Both of these reasoning functionalities are not available in both Ecore and ORM 2 models.

Another kind of reasoning functionality is given by instance classification. While this is not scoped by Ecore, ORM 2 allows the allocation of an *Entity* to one of several *Entity Types* of a common superclass. This is achieved by asserting *subtype derivation rules* to their common superclass (Halpin & Morgan, 2008). OWL 2 allows the allocation of *Individuals* to any *Class* that has *necessary and sufficient conditions* defined (Allemang & Hendler, 2011).

In addition to inferring the class membership of *Individuals*, OWL 2 allows the inference of new instance properties that implicitly have to hold. This means that, for example, properties have to hold for an *Individual* since it is member of a specific *Class*, and each member of the *Class* must have this property, or that a property has to hold because it is the super-property of an asserted sub-property of the *Individual*, or that a property has to hold as it is formed by a chain of asserted properties (Allemang & Hendler, 2011). This functionality is not provided by Ecore or ORM 2.

Ecore and ORM 2 allow the identification of inconsistent instance populations on M0 level in respect to constraints defined on M1 level. This includes, for example, cardinality violations, or uniqueness violations of properties (Halpin & Morgan, 2008; The Eclipse Foundation, 2016c). A similar functionality is offered by OWL 2, although this form of consistency checking can only happen in respect to open world semantics. This means that, in essence, an *Individual* may never be inconsistent because it does not exhibit mandatory properties, but only because it has too many properties in terms of cardinalities, or in terms of logical contradictions.

Table 6.5: Summarized comparison of language reasoning functionality

Characteristic	Ecore	ORM 2	OWL 2
Detection of implicit subclass/superclass relationships	Not scoped		With reasoner
Detection of unsatisfiable class definitions	Not scoped		With reasoner
Classification of instances	Not scoped	Basic, subtype derivation	Inference of class membership for individuals based on class restrictions
Inference of instance properties	Not scoped		Inference of individual property assertions based on class restrictions
Identification of inconsistent concepts	In respect to constraints		In respect to class definitions and considering open world semantics

6.1.6 Miscellaneous Characteristics

Ecore allows the modeling of functional aspects of the software to be produced in addition to its structural aspects. This can be done using *EOperations* (The Eclipse Foundation, 2016c), which represent the methods of *EClasses* that should be implemented manually later on. As ORM 2 focuses on conceptual domain modeling, the modeling of functional aspects is not scoped. The same is true for OWL 2, where structural domain information is the main subject of interest, and functionality of a software supporting activities inside the domain is not considered (W3C, 2006).

Ecore models do not directly allow encapsulation or access restriction on variables (The Eclipse Foundation, 2016c). This can only be done in the according Java model after code was generated, and not directly for the model code. Java provides *public*, *private*, *protected*, and *package protected* access types. This is not scoped by ORM 2. OWL 2 deliberately avoids access restrictions, as all resources are meant to be accessible by anybody on the Web (W3C, 2006).

Ecore allows partitioning of an M1 model by using *EPackages* that form the container of *EClasses*, *EDataTypes*, and other *EPackages*. This comes along with a significant impact towards code generation (The Eclipse Foundation, 2016c). ORM does not have any model partitioning concept. OWL 2 does also not have an explicit partitioning concept inside *Ontologies*, however *Ontologies* can import other *Ontologies* that each have their own namespace, allowing the partitioning of an *Ontology* using *Ontologies* themselves (W3C, 2012a).

Each M1 model of the three languages comes with a number of foundational concepts. These represent initial populations of selected M1 model concepts that are used in virtually any model design. In Ecore, these foundational concepts are given by pre-modeled *EDataTypes*, such as *EString*, *EInt*, or *EBoolean*, that can be used for typing *EAttributes* (The Eclipse Foundation, 2016c). In ORM 2 models, a pre-defined set of *Value Types* are available for the same purpose (Halpin & Morgan, 2008). In OWL 2 ontologies, the concepts of *owl:Thing* and *owl:Nothing* are available, of which the first forms the common superclass of any *Class* in the model, and the latter is used to denote unsatisfiable concepts. Furthermore, XSD data values are pre-populated in each *ontology* and are used for typing *Data Properties* (Kiko & Atkinson, 2008).

Ecore models allow adding miscellaneous information to any *EModelElement* using the concept of *EAnnotation* (The Eclipse Foundation, 2016c). An annotation concept is not part of the ORM 2 language. In OWL 2, *Annotation Properties* can be asserted to any concept that has an *IRI* (W3C, 2012a).

ORM 2 models allow the modeling of optional constraints by including the concept of *Deontic Constraints*. These are also evaluated and mark an inconsistent model, but

the inconsistency may not be of grave consequence to the domain (FBM WG, 2014). Such an optional constraint concept is neither scoped by Ecore, nor by OWL 2.

Regarding a diagrammatic representation of the M1 model, Ecore uses *Ecore Diagrams* (The Eclipse Foundation, 2016b), while ORM 2 uses *ORM 2 Diagrams* (Halpin & Morgan, 2008). OWL 2 does not have a standardized diagram for graphically representing its M1 models, but several non-standardized views exist. Diagrams of M1 models are often called concrete syntax.

The Ecore language has its language described in Ecore itself (The Eclipse Foundation, 2016c), and ORM 2 has its language described in ORM 2 itself (FBM WG, 2014). OWL 2 does not use an explicit model described in itself for defining language semantics. Instead, OWL 2 is specified in *BNF notation* (W3C, 2012a).

Table 6.6: Summarized comparison of further language characteristics

Characteristic	Ecore	ORM 2	OWL 2
Modeling of functional aspects	EOperation	Not scoped	
Concept encapsulation	Not scoped, only in Java code model once generated	Not scoped	Not scoped, all resources public by intent
Model partitioning	EPackage	Not scoped	No dedicated concept, ontologies may import other ontologies
Model partition semantics	Container for EPackages, EClasses and EDataTypes, code generation semantics	Not scoped	Container for Class, Object Property, Datatype Property, Annotation, Individual
Foundational concepts	Pre-defined set of EDataTypes	Pre-defined set of Value Types	Pre-defined owl:Thing, owl:Nothing and xsd data values
Annotations	EAnnotation for any EModelElement	Not scoped	Assertion of Annotation Property for any IRI
Optional constraints	Not scoped	Deontic constraints	Not scoped
Concrete syntax	Ecore diagrams	ORM 2 diagrams	No diagramming
Language specification syntax	Ecore	ORM 2	BNF notation

6.1.7 Conclusion of Language Comparison

The analysis of the three languages confirmed the rough outline of the initial analysis in section 5.2. Furthermore, it made visible further nuances, as well as significant characteristics, that differentiate the languages from each other.

Ecore is semantically very well defined inside its framework, EMF, with a strong influence on the code that is being generated from an Ecore-based model. Although the direct semantics are focused on describing the structure of a software system, this maps to the structure of the domain in question quite well. Ecore only supports very basic constraining focused on its class properties, leaving out common constraints such as subsets, as well as more specialized ones. Ecore exclusively describes necessary conditions for its M0 models, not considering any form of inference. With its CWA, Ecore is in line with the current engineering data management approach.

ORM 2 does not focus on describing any software-related aspects, but focuses on capturing the facts of a domain of interest. ORM 2 puts a more finely-grained view on relations between model concepts, with the possibility to employ sophisticated constraining to these relations. All in all, Ecore and ORM 2 behave very similarly when regarding general class structure, class attributes, and binary relations. The constraints offered by ORM 2 are based on established logical concepts and are not exclusive to ORM 2 syntax. ORM 2 also relies on a closed world and also focuses on defining necessary conditions on a model with no emphasis being put on reasoning.

OWL 2 differs significantly in a number of characteristics from both Ecore and ORM 2. Being based on the OWA makes M0 models behave differently from the currently employed data management approach, with missing information unable to be queried for. Consequently, no focus is put on modeling constraints. With its incorporation of DL, the language has strong mathematical-logical semantics that enable the inference of new information from an existing model population on both M1 and M0 levels. This includes making implicit superclass-subclass relationships explicit, inferring the class membership of instances, and inferring new instance attributes. Furthermore, OWL 2 differs from the other two languages by being able to change the M1 model during runtime, the ability for instances to be member of multiple classes or no class at all, and the ability to supply M0 model information that is not scoped by its corresponding M1 model.

However, there are also aspects identified in the requirements analysis in 5.2 that are not covered by any of these languages. These aspects include the modeling of lifecycle aspects on data, the relation of M1 concepts to PDM artefacts, and the modeling of functional dependencies on M1 level.

All in all, Ecore offers a sound basis for producing software, relying on closed world semantics. ORM 2 has its strengths in model constraining. OWL 2 opens up the world

of reasoning by supplying mathematically sound semantics to its models, in addition to multiple, dynamic classification and M1 model adaption during M0 model runtime.

6.2 Language Design Discussion

Different solutions for the language architecture are conceivable. These solutions range from standalone architectures, relying solely on one of the selected languages, to combinations of all of them. This section elaborates on alternatives for the language's design.

For approaching this trade, a brief functional analysis is performed. For this purpose, functions that the language has to fulfil are defined, derived from the requirements identified in 5.2. The ability of each of the three languages to directly support these functions is then evaluated, as outlined in Table 6.7.

Table 6.7: Direct function realization capability per language

Language Function	Derived from REQ	Directly realizable by		
		Ecore	ORM 2	OWL 2
Artefact modeling and CDM mapping	1-1			
Constraint modeling	1-2		•	
Closed world fact support	1-3	•	•	
Functional rules	1-4			
Multiple explicit characterization mechanisms	1-5			•
Data lifecycle aspects	1-6			
Project-specific adaption of CDM	2-5			•
Disjoint reasoning	2-6			•
Reasoning capability	3-8			•
MDA and EMF compatibility	4-1	•		

6.2.1 Standalone Language Architectures

As can be seen from Table 6.7, no language is able to fulfil all requirements on its own. In addition, there are functions that are currently not covered by any of these

languages. In consequence, this means that, if solely one language is used as basis, it has to be heavily extended in order to incorporate all of the functionality.

6.2.1.1 Ecore Only

Ecore offers the capability to extend *EClasses* in custom models, so an extension of the language is technically possible. As Ecore only brings along MDA and EMF compatibility, as well as closed world fact support, all other functions have to be separately integrated. For integrating reasoning capability, this implies integrating complete OWL 2 or at least DL semantics into the language. An additional challenge is given by the introduction of multiple explicit element characterization mechanisms that stand in strong contrast to object-oriented design principles.

6.2.1.2 ORM 2 Only

ORM 2 does not offer an extendibility interface. It brings along elaborate constraint modeling, and closed world support, but is not directly MDA or EMF compatible. All other features, such as reasoning support, also have to be integrated manually.

6.2.1.3 OWL 2 Only

OWL 2 supports the required reasoning aspects, but circumventing its OWA is rather difficult. For some cases, a local closed world can be accomplished with OWL 2 using specific operators (Mehdi & Wissmann, 2013), but reasoning support for these approaches is very limited. This, and the lack of a language extension mechanism, makes introducing constraints and the other required aspects quite difficult.

6.2.2 Transformation-Based Architectures

Pursuing a standalone language architecture does not seem adequate for providing the required functions. On the one hand, the extensions to languages are extensive, on the other hand the semantic implications of introducing non-native semantics to sophisticated languages comes with the risk of breaking the semantics entirely, and brings in additional concepts that were originally not meant to exist in the language's context. Therefore, other options are evaluated.

In order to bring together the features of multiple languages, an integration in terms of transformation from one language to another language can be done. However, such an approach always has the problem of semantic loss of the concepts supported by the transformation source language, but not covered by the transformation target lan-

guage. In consequence, only concepts covered by both source and target language can be preserved in a back-and-forth transformation between models based on languages with a different semantic scope.

Figure 6.1 illustrates this issue. While both languages A and B support similar semantics to some degree, there are also semantics that are specific to each language. In a transformation from language A to B and vice versa, only the data relying on common semantics can be exchanged. Data represented using semantics specific to one of the languages cannot be exchanged, as its describing abstract concept is not available in the other language.

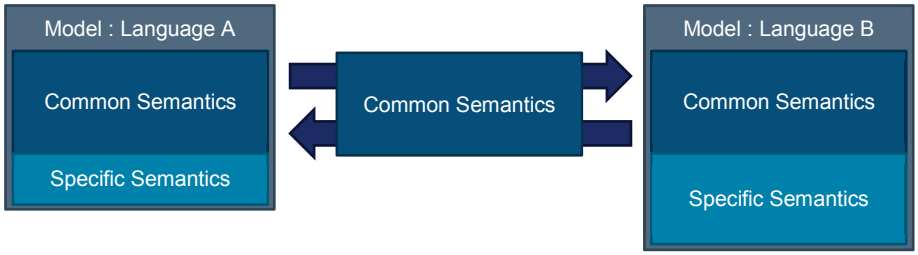


Figure 6.1: Semantic loss caused by model-to-model transformation

The consequence of this is that a transformation between languages does not bring any additional value if the concepts from the source language are not covered by the target language.

This section explores several conceivable transformation-based language designs, discussing the overall functional coverage and the semantic loss occurring during the transformation. While several dozen combinations of transformations between Ecore, ORM 2, and OWL 2 are possible, only those that cover at least some of the defined language requirements are mentioned explicitly.

6.2.2.1 ORM 2 to Ecore

A conceivable approach is to use ORM 2 for modeling the CDM, and to transform it to Ecore for instantiation. This enables the usage of ORM 2 constraints in the Ecore-based model, under the assumption that OCL is employed in the Ecore model to represent the constraints. The realization of this approach was demonstrated in Hennig, et al. (2016a). However, many of the required functions are not supported by this approach, including project-specific CDM adaptions, functional rules, multiple characterization mechanisms, and especially all required reasoning aspects.

6.2.2.2 Ecore to OWL 2

With this in mind, building an Ecore M1 model and transforming it to OWL 2 will lead to semantic loss of the concepts not covered by OWL 2, and shift the M0 model interpretation from closed world to open world, which is not desired. Furthermore, Ecore itself already does not cover all required concepts. This alternative is not further pursued.

6.2.2.3 Extended Ecore to OWL 2

In an approach where Ecore is extended to support all required functionality, except the functionality covered by OWL 2, a significant portion would not be persisted in the transformation towards an ontology. Although the modeling of the relation of CDM entities and PDM artefacts could also be realized in OWL 2, the realization of functional dependencies or consistency checks is difficult to be realized in the OWL 2 scope. Furthermore, the OWA problem still persists.

6.2.2.4 OWL 2 to Extended Ecore

The other way round, to use an OWL 2 CDM and to transform it to an extended Ecore model is also conceivable. This is under the assumption that it is the same extended Ecore language as detailed in 6.2.2.3. This architecture would allow mappings between CDM concepts and process artefacts, using closed world facts, and providing MDA and EMF compatibility. However, the OWL 2 semantics are not preserved.

6.2.2.5 OWL 2 to Extended Ontological Ecore

Another conceivable solution is the usage of the Extended Ecore proposal, and to also include ontological concepts there, essentially resulting in including the whole semantic extent of the OWL 2 language. This way, a reasoner can be instantiated on the EMF model. This approach is able to cover a lot of functionality, such as CDM adaption during runtime, multiple typing, functional rules, and artefact modeling. However, as the original Ecore model has now shifted to essentially being an OWL 2 model, the limits of the OWL 2 model apply. This includes the constraint to being an open world model, and in consequence the inability to perform evaluation of a large number of required constraints. Although these constraints are available for modeling on M1 level, they can never be executed on M0 level due to their incompatibility with the OWA. This also has an impact on the evaluation of data lifecycle aspects.

6.2.3 Parallel Language Architectures

Of the transformation-based architectures, many exhibit the problem of semantic loss during the transformation, essentially not considering populated concepts of the source language that are not scoped by the target language. This can be overcome by extending the target language to also encompass essential concepts of the source language. However, if this is practiced in an extensive manner, fundamental semantics of the target model shift from the original semantics to those of the source model, retaining all problems the source model originally had.

What these architectures all have in common is that they involve multiple CDMs on M1 level, but only one real implementation of the CDM on M0 level. In order to overcome the challenges of transformation-based architectures, an architecture can be defined that is based on two CDMs in two different languages, and also maintains two distinct, but highly interrelated, SMs on M0 level. Using this approach, two SMs, based on two distinct paradigms, can be used to represent information of one single system, combining the merits of the specific modeling technologies.

6.2.3.1 Ecore and OWL 2 in Parallel

One solution to do this would be to host in parallel an Ecore and OWL 2 model. This involves two separate CDMs, representing the production-oriented, and the knowledge-oriented aspect of the domain, and two corresponding SMs. While the Ecore-side of the system's representation is responsible for closed world checks such as consistency checking and production-oriented aspects, the OWL 2 side accomplishes the knowledge capture and reasoning part. However, a plain Ecore and OWL 2 approach does not cater to specific functionality not scoped by both languages, such as the modeling of sophisticated constraints, artefact modeling, defining functional rules, and maintaining a lifecycle aspect on data.

6.2.3.2 Extended Ecore and OWL 2 in Parallel

Consequently, in order to support all required functions, an extension of Ecore catering towards this custom functionality, and an OWL 2 model hosted in parallel are required.

6.2.4 Summary of Language Architecture Alternatives

Table 6.8 summarizes the functional coverage of each discussed language design, and provides a comparison.

The standalone language architectures are only able to realize the functionality that is directly supported by the language. The transformation-based designs all suffer the problem of semantic loss, also failing to address all requirements. This problem can be mitigated by a design where two languages are employed in conjunction. While Ecore and OWL 2 are not able to support all required functionality, an extended Ecore language with OWL 2 in parallel is able to cover all required aspects. Consequently, this design is selected to be pursued further.

Table 6.8: Functional comparison of language architecture alternatives

	Ecore only	ORM 2 only	OWL 2 only	ORM 2 to Ecore	Ecore to OWL 2	Extended Ecore to OWL 2	OWL 2 to Extended Ecore	OWL 2 to Extended Ontological Ecore	Ecore and OWL 2 in parallel	Extended Ecore and OWL 2 in parallel
Project-specific adaption of CDM			●		●	●		●	●	●
Disjoint reasoning			●		●	●		●	●	●
Artefact modeling and CDM mapping						●	●	●		●
Constraint modeling		●		●						●
Closed world fact support	●	●		●			●		●	●
Functional rules								●		●
Multiple explicit characterization mechanisms			●		●	●		●	●	●
Data lifecycle aspects										●
Reasoning capability			●		●	●		●	●	●
MDA and EMF compatibility	●			●			●	●	●	●

6.3 SCDML Design

The Semantic Conceptual Data Modeling Language (SCDML) picks up features from both Ecore and ORM 2 and integrates them consistently. Also, a bridge for mapping SCDML concepts to OWL 2 concepts is provided. In addition, functionality dedicated towards supporting functional aspects not covered by any of the three languages is introduced to SCDML, leading to the architecture as outlined in Figure 6.2.

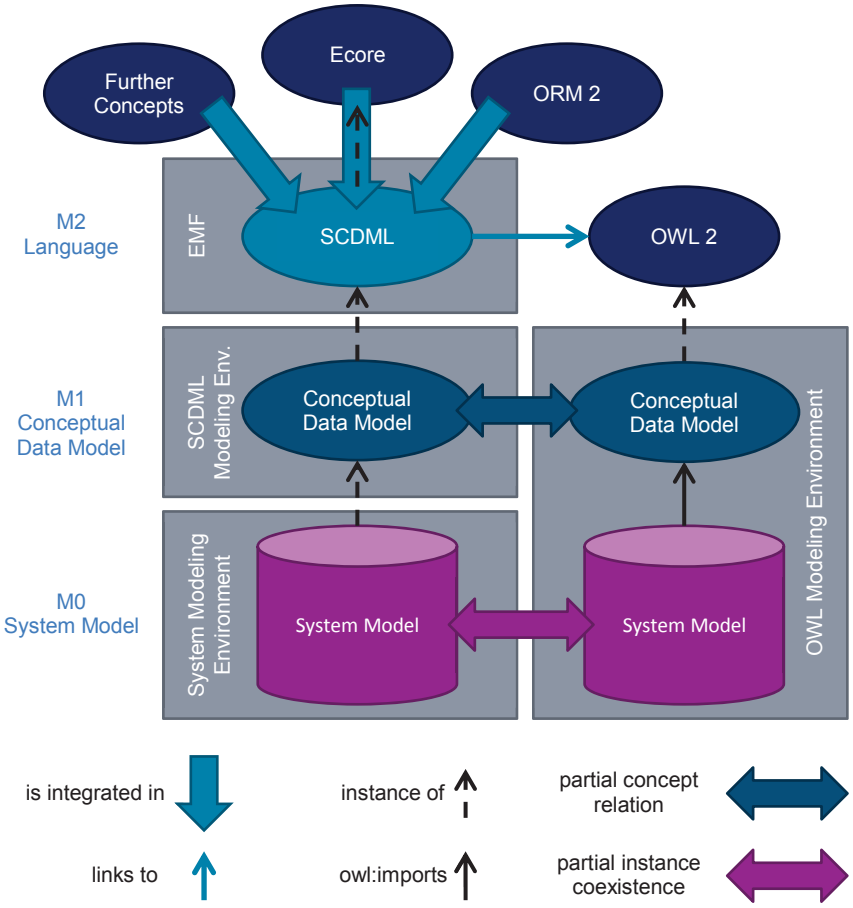


Figure 6.2: SCDML architecture

A characteristic of this design is that the CDM is not hosted in one CDM alone, but in two separate CDMs, based on two different modeling principles, forming a virtual CDM. As a consequence of this separation, the whole semantics of the CDM can only be grasped when regarding both the ontological, and the object-oriented CDM. Of those two, the latter contains information of where the two CDMs relate.

Each concept of the SCDML language is an instance of a concept of the Ecore language. SCDML consists of three top-level packages (Figure 6.3).

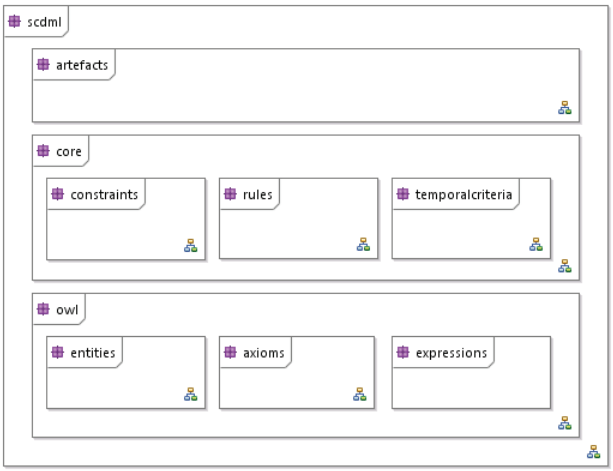


Figure 6.3: SCDML package structure

The *core* package supplies the data structures necessary for modeling a CDM's main data concepts and can be seen as a direct derivative of the Ecore language, with many parallels. Furthermore, this package scopes constraints, rules, and temporal aspects.

The *owl* package supplies the concepts necessary to model OWL classes that are related to *SClasses* of the core package using the concept of the *AbstractSemanticClass*.

The third top-level package is the artefacts package that supplies the ability to model process artefacts on PDM level and to relate them to concepts of the CDM.

The concepts of these packages are all contained in one *Model* (Figure 6.4)

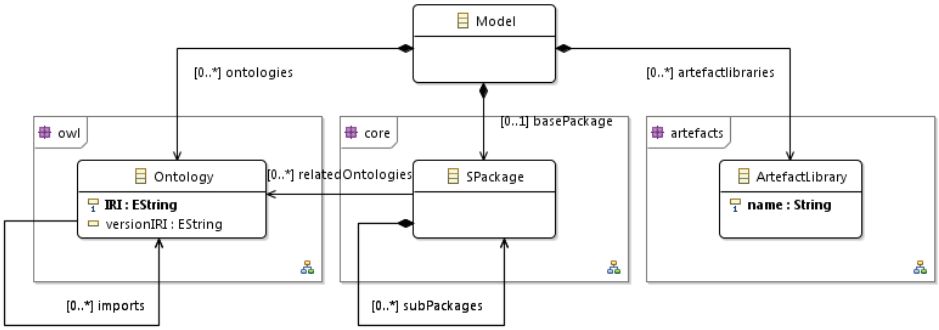


Figure 6.4: SCDML model

In the following sections, central concepts of the SCDML language will be detailed, explaining how they are designed, what functionality they provide, and how this relates to the required functionality. While the description of the concepts in this chapter is rather abstract, not going into detail of any application, Chapter 8 explains in more detail how these concepts are applied to modeling a concrete CDM, giving a more concrete idea about the motivation behind them.

6.3.1 Core Model: Modeling Overall Data Structures

The *core model* (Figure 6.5) provides the conceptual structures for modeling the central concepts of a CDM, such as *SPackages*, *SClasses*, *SReferences*, and *SAttributes*. It has a structure very similar to the Ecore model (The Eclipse Foundation, 2016c), providing the possibility to easily move from an Ecore-based representation of a CDM, to an SCDML-based one. Furthermore, compatibility to EMOF is provided by this approach.

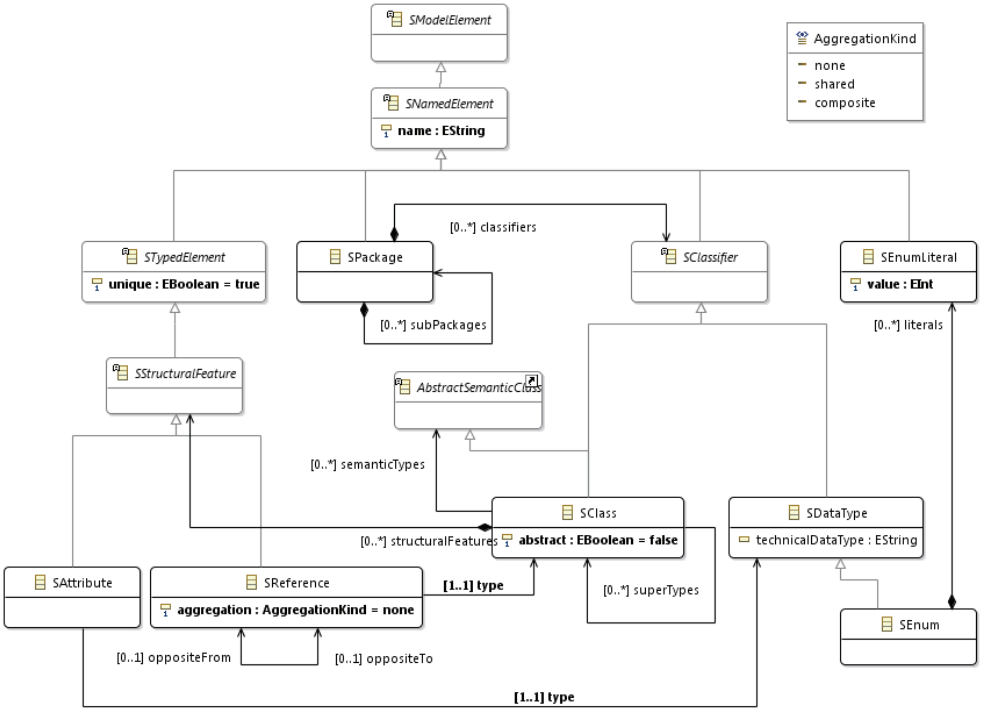


Figure 6.5: SCDML core package

Instead of extending the Ecore model, the core model mimics it in the majority of constructs. A difference occurs with the handling of feature cardinalities, which is realized by constraints instead, enabling an assignment of *temporal criteria* (see 6.3.3) to cardinalities. Also, technical model properties that are not of relevance to conceptual modeling are left out in SCDML. The core semantics are identical to Ecore, also adhering to the same naming scheme, in terms of *SPackages*, *SClasses*, *SReferences*, *SAttributes*, etc. The purpose of the core model is to capture the domain structure that is relevant for being supported by the usual software-based engineering support activities, supporting code generation.

The root of the core model is given by the *SPackage*, acting as a partitioning element. It contains other *SPackages* via the *subPackages* references, as well as *SClasses* and *SDataTypes* via the *classifiers* reference. *SClasses* can have any number of *SStructuralFeatures*, which may either be *SReferences* or *SAttributes*. *SReferences* are typed by another *SClass*, while *SAttributes* are typed by an *SDataType*.

6.3.2 Constraints: Defining Valid Model Populations

A significant number of constraints available in ORM 2 are also incorporated into SCDML (Figure 6.6).

Constraints are used to define the area of SM populations that adhere to rules or statements derived from the context in which the SM is situated, that constitute consistent and legitimate models. Constraints in this context may, for example, define the minimum required values for a given attribute, logical dependencies between two references, or a maximum amount of instances of a given class that may exist at one time. If these constraints are violated in an SM, the SM is regarded as not correct. Without constraints, any technically possible SM population may be defined, which might not constitute a valid model in terms of the internal rules or conventions of the model's context. Constraints in SCDML are grouped into three categories:

ClassHostedConstraints are hosted by and of relevance to an *SClass*. This includes, for example, the *ClassMultiplicityConstraint*, which specifies that only a limited number of instances for a specific class may exist, or the *ForbiddenClassConstraint*, that specifies that, at a given point in time, no instances of this class may exist.

FeatureHostedConstraints are hosted by and of relevance to *SStructuralFeatures*, i.e. *SAttributes* and *SReferences*. The *RingConstraint* deals with constraining reflexive *EReferences*, e.g. enforcing *acyclicity*, *reflexivity*, or *transitivity*. The *FeatureMultiplicityConstraint* can be used to specify that a specific feature can only exhibit a limited number of values across the entire model. The *FeatureCardinalityConstraint* is used to specify *lowerBounds* and *upperBounds* of *EStructuralFeatures* and has been evolved to a class instead of an attribute in order to enable referencing. The *ForbiddenFeatureConstraint* can be used to state that a specific feature cannot be set for a given period of time.

PackageHostedConstraints are contained in an *SPackage* and are used to constrain multiple features. This includes the *SubsetConstraint* for defining subsets of *EReferences*, the *ValueComparisonConstraint* for comparing numerical values of applicable *EAttributes*, and *SetComparisonConstraints* such as *Equality*, *Exclusion*, *ExlusiveOr*, and *InclusiveOr*. These constrains have a mode associated with their definition that can be used to specify of the constraint shall merely compare if the feature *is set*, or if it should compare actual *populations* set in the feature.

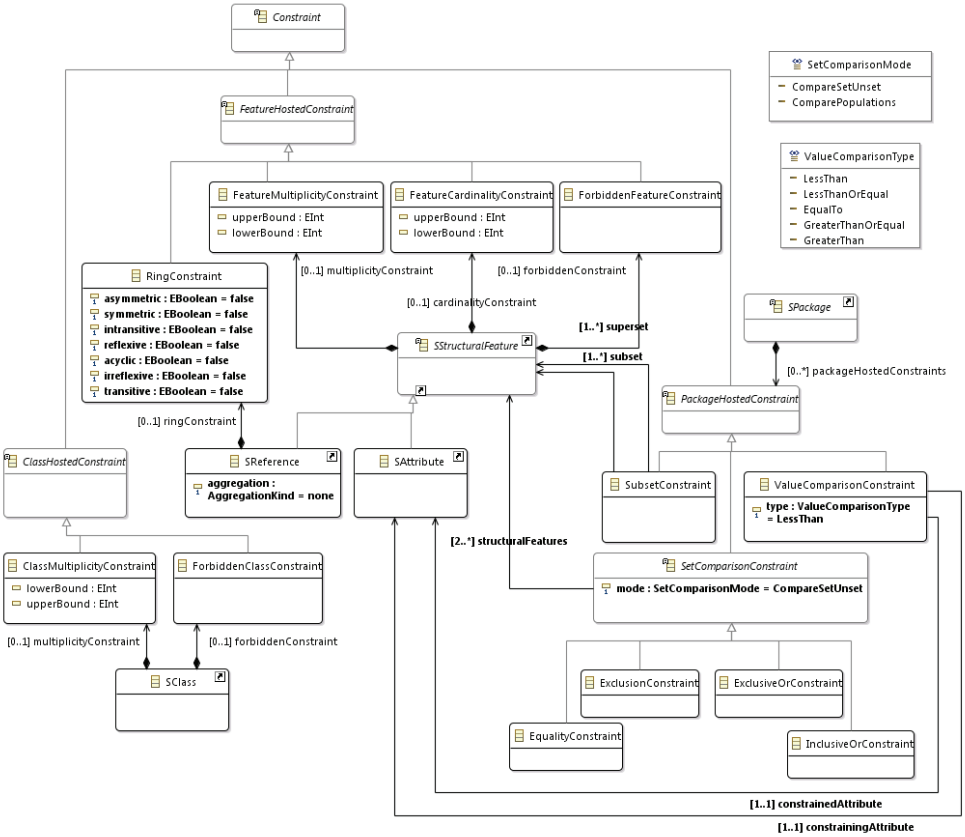


Figure 6.6: SCDML constraints package

6.3.3 Temporal Criteria: Assigning Lifecycle Aspects

For realizing the modeling of lifecycle aspects to data, the concept of *TemporalCriteria* is introduced.

In a given SM describing a system, the data represented by it may be formed differently at different points in the system's lifecycle. For example, in the beginning of a system's design, its description in the SM may be rather generic, with only a brief description of the system's constituents existing. As the system design gets more elaborated along its lifecycle, more details are required to be provided in the SM, such as extensive descriptions about the purpose of the system's constituents, behavior descriptions, and detailed descriptions of the system's interfaces.

This is realized with the concept of *TemporalCriteria*. A *TemporalCriterion* forms a point in time where a given set of *Constraints* applies. This point in time forms a milestone at which the data present in the SM has to have a specified extent and condition.

Each *Constraint* can be valid for an arbitrary amount of *TemporalCriteria*. If a constraint does not have any *TemporalCriteria* associated with it, it is valid at any point in time. If it has *TemporalCriteria,* the constraint only applies to these criteria.

TemporalCriteria are hosted in an *SPackage* and can be related with other *TemporalCriteria* by forming a hierarchy (Figure 6.7). This has been done to allow disciplines to have their own time concepts different from those globally applying to the system.

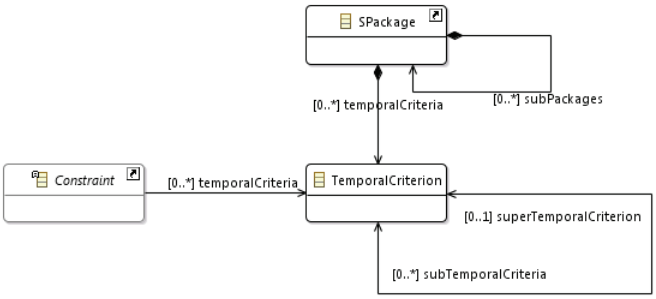


Figure 6.7: SCDML temporalcriteria package

The concept of modeling temporal aspects of data is quite simplified, only providing an instant or rather milestone-based consideration of aspects. It should not be confused with more elaborate, genuine time-modeling, as is provided by concepts such as the Time Ontology (W3C, 2017).

6.3.4 Rules: Modeling Functional Model Aspects

SCDML has two kinds of rules incorporated, that are used for two different purposes. On the one hand, for specifying consistency checks not scoped by any of the *Constraints*, OCL constraints (OMG, 2014b) can be incorporated into an SCDML-based CDM (Figure 6.8). The OCL statements covered by the *rules* package cover the standard extent of the OCL language. As such, all statements that can be expressed normally in OCL can be integrated in SCDML-based CDMs.

On the other hand, *FunctionalRules* (also Figure 6.8) can be modeled that describe functional aspects between concepts of the CDM. These functional aspects involve,

for example, the implications of the existence of a specific instance towards the existence of other instances, or the impact specific values of one instance have on the values of another instance. The *FunctionalRules* of SCDML do not intend to provide a generic description of business rules as is offered by other rule languages such as SBVR (Bollen, 2008), or FORML 2 (Halpin & Wijbenga, 2010). Instead, this concept is tailored towards providing the functionality required by the concepts described in 10-23 (ESA, 2011a). Concrete examples for *FunctionalRules* are described in 8.5.2, when a concrete CDM is detailed.

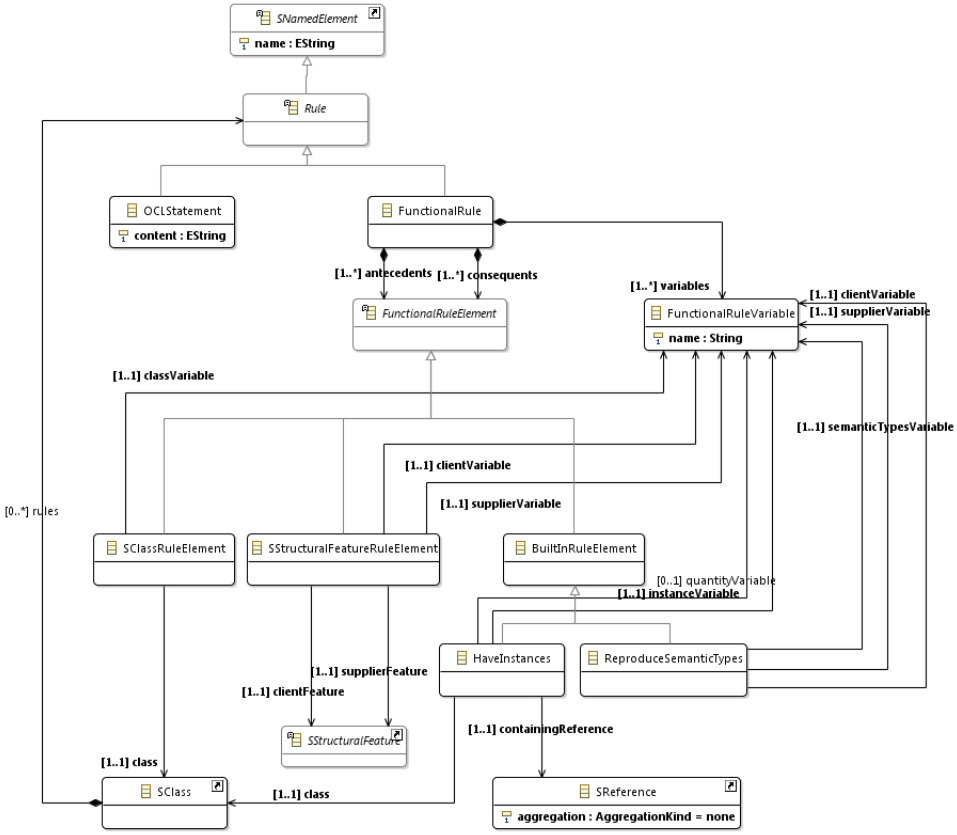


Figure 6.8: SCDML rules package

6.3.5 Aligning Technical Typing with Domain Typing

SCDML supports two kinds of typing mechanisms. On the one hand, the usual notion of super-typing is supported, defining the technical type for any *SClass*. On the other hand, the concept of *semanticType* is supported. This concept can be used to define the actual domain-specific meaning for an *SClass*, independent of its technical aspects. The *semanticType* of an *SClass* can be either an *SClass* itself, or an *OWLClass*. An *SClass* can have no *semanticType* at all, or have multiple *semanticTypes*, allowing multiple instantiation of either object-oriented or ontological concepts onto the instance standing behind a single *SClass*. SCDML typing mechanisms are illustrated in Figure 6.9.

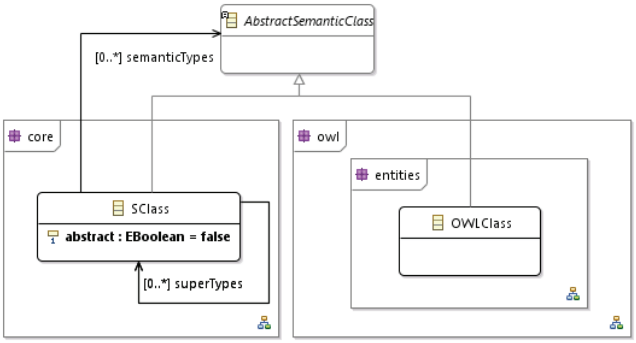


Figure 6.9: SCDML AbstractSemanticClass

The motivation behind this concept is to provide an arbitrary number of typing relations for a given element in the system. As described earlier in 2.4.8, a domain object is often in reality not only described by one type, but by multiple types. This combination of types leads to a proper semantic description of the domain object that cannot be provided with only a single typing relation.

6.3.6 Mapping Discipline Data and Process Artefacts

For being able to relate abstract artefacts from the process and concrete engineering data with each other, the concept of *ManagedArtefact* is introduced. With this concept, all data described on system level can be allocated to artefacts of the surrounding engineering process, as described in 2.4.4. These *ManagedArtefacts* can be contained in an *ArtefactLibrary* and map to *SClasses* or *SPackages*. Artefacts can also exhibit dependencies between each other, but this is not mandatory (Figure 6.10).

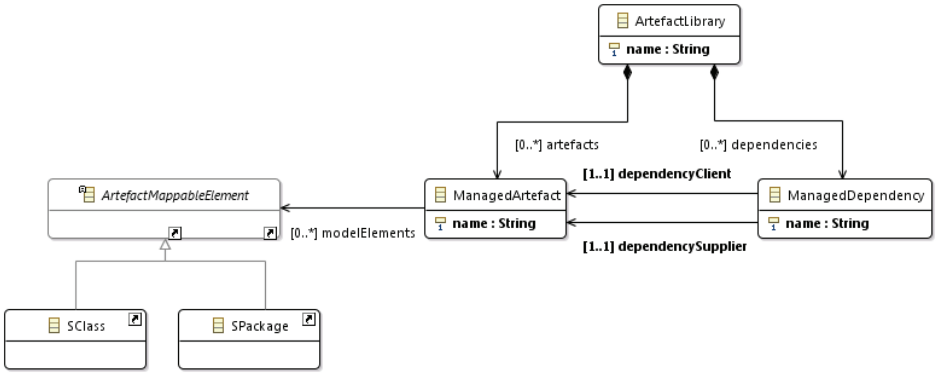


Figure 6.10: SCDML artefacts package

6.3.7 Mapping Object-Oriented to Ontological Descriptions

For realizing the mapping of classes in the object-oriented CDM to those of the ontological CDM, *Ontologies* and *OWLClasses* have been introduced into SCDML that are identified by their *IRI*. These *IRIs* can be used to relate the concepts mentioned in the SCDML model to those of their original *Ontology* (Figure 6.11). The OWL model is then used to capture the knowledge-focused-side of the domain, to realize reasoning aspects, and to realize the runtime-adaption of modeled concepts.

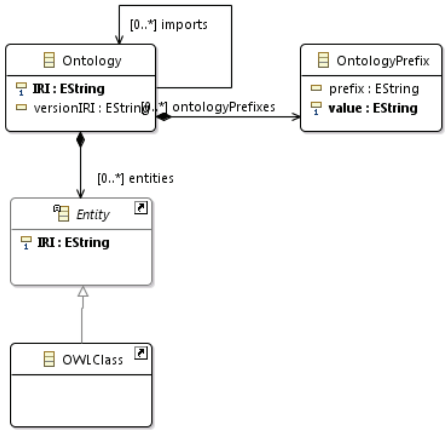


Figure 6.11: SCDML ontological aspects

6.3.8 Realization of Required Functions

The functions identified in 6.1.7 as required are all considered in one or the other concept of the SCDML language. A mapping of what function is realized with which concept is provided in Table 6.9.

Table 6.9: Realization of required functions with SCDML

Language Function	Realized by
Project-specific adaption of CDM	OWL integration
Disjoint reasoning	OWL integration
Artefact modeling and CDM mapping	artefacts package
Constraint modeling	constraints package
Closed world fact support	Reliance on Ecore
Functional rules	rules package
Multiple explicit characterization mechanisms	AbstractSemanticClass
Data lifecycle aspects	temporalcriteria package
Reasoning capability	OWL integration
MDA and EMF compatibility	Reliance on Ecore

6.4 Differentiation from Existing Work

A number of other approaches for relating object-oriented models with ontologies exist. While these approaches exhibit similarities at first glance, each existing approach differs significantly to the approach proposed in this thesis.

6.4.1 Mooop

Mooop (Merging OWL and Object-Oriented Programming) is an approach proposed by Frenzel (2010) for providing an implementation of OWL-based ontologies, hoping to support function aspects of ontologically defined data. For this purpose, an approach is developed that maps OWL TBox concepts to Java classes and OWL ABox concepts to the according objects.

This work relies solely on Java and does not consider any model-based aspects such as the MDA or MOF. Furthermore, the resulting models rely on information from the ontology with no additional information, such as packages, operations, or variables being supplied by an object-oriented model, only considering the semantics of OWL. The authors emphasize on software development aspects and do not consider the domain aspects of the resulting model.

6.4.2 The TwoUse Approach

The TwoUse Approach was proposed by Silva Parreiras (2011), pursuing the goal of using ontologies to reason about the design of software systems. For this purpose, an integration of Ecore and OWL was implemented, based on the OMG's ODM (OMG, 2014c). Using this approach, an Ecore model can be transformed to an OWL ontology, where a reasoner can then be applied to make visible new information on the design of the software.

The proposed TwoUse Approach is unidirectional and can only support the direction from Ecore to OWL, with no way of bringing information back into the Ecore model. The used mapping chooses the most obvious approach of transforming the concepts that can be transformed easily due to similar semantics. *EClasses* get transformed to *OWLClasses*, *EReferences* to *ObjectProperties*, *EEnums* to *DataOneOf restrictions*, etc. As a consequence, only information can be used in this process that is covered by both the Ecore and OWL language, with concepts such as *class disjointness*, *concept equivalency*, and the majority of *restrictions* not able to be specified. Similarly to Mooop, this work only considers the activity of software design and does not consider the application of the model that is managed by the software.

6.4.3 M3 Integration Bridge

Aßman, et al. (2013), take a position towards integrating the concepts of both Ecore and OWL on M3 level in a hybrid language, for the purpose of enabling language users familiar with only one of the languages to slightly annotate their concepts with concepts from the other language.

This approach integrates both languages, but suffers from deficiencies in its model design. For example, *EReferences* are considered to be subclasses of *ObjectProperties*. Although, in fact, the semantics of both concepts are quite different, as *ObjectProperties* form global definitions of references without being assigned to a *Class*, and *EReferences* being the assignment to an *EClass* and the definition of a reference at the same time. In addition, *eOpposites* to an *EReference* are treated as not related at all to

the *inverse* property of *ObjectProperties* in OWL, having potential information duplication, but without highlighting that this is actually equivalent information.

6.4.4 Integrating object-oriented and Ontological Models

Puleston, et al. (2008) developed a concept that enables integrating selected concepts from an OWL ontology into a Java model, with the intent of enabling reasoning and providing a dynamic adaption of specific model parts, based on information that is supplied to the model on instance level.

The developed approach does not consider MDA aspects and relies solely on Java code as model for the software-part of the system. Furthermore, the model uses a central Java-class around which the ontological aspects are situated, without the ability for ontological knowledge to be used independently from the Java classes. The authors talk about temporal aspects of the model, not in terms of constraint applicability, but with the idea in mind that the same type of data can take different values if it is collected at different points in time, e.g. for representing a series of measurements.

6.4.5 Adjustable OWL to Ecore Transformation

Rahmani, Oberle, and Dahms (2010) also propose a transformation-based approach for implementing an OWL ontology with the help of Ecore. The transformation is adjustable in its scope, with simple transformations being possible that only transform the OWL ABox to Ecore, the transformation of the ABox and TBox to Ecore, or the transformation of both to a model that uses Ecore and OCL.

However, this transformation suffers the same problem as many others, where information is not adequately transformed that is covered on the OWL side of the model, but not being able to be represented on the Ecore-side of the model. For example, instances are always distinct, while Individuals could be identical on the OWL side. On the Ecore side, no information can be modeled that is not scoped by the TBox on OWL side. Furthermore, the capability to use multiple super-types for *Individuals* is not retained by the transformation, as each *EObject* always has exactly one type. *Equivalent Classes* cannot be represented adequately, as one main class has to be chosen in the transformation. Furthermore, a semantic loss occurs for datatype properties that involve some kind of data range, as this is always mapped to an *EAttribute* of type *EString*.

6.4.6 Ecore-Based ORM 2 Implementation

The work that led to SCDML also explored the path of using ORM 2 as CDM syntax, and to implement it using Ecore. This was demonstrated in previous research (Hennig, et al., 2016a). The approach explored the idea and the implications of implementing a conceptual-focused language in the Ecore context. This research highlighted that, although ORM 2 puts more emphasis on relations on CDM level, this advantage is alleviated when going into object-oriented implementation, where the semantics become identical to those of an Ecore-based CDM. Furthermore, using *Fact Types* of arity greater than two did not yield large benefits to the expressiveness of the underlying CDM. Additionally, the ORM 2 syntax being different to the established syntax of e.g. Ecore and UML introduced additional complexity to the modeling process. However, the richness of constraints available in ORM 2, that were translated to OCL constraints in the implementation model, were identified to be of benefit for improving the consistency of the SM, as significantly more concepts to constrain the described domain data become available.

6.4.7 Semantic MOF

Semantic MOF (SMOF) (OMG, 2013) is a specification that is part of MOF (OMG, 2015a). SMOF describes a concept for integrating multiple classification and dynamic reclassification into MOF, as this is a functionality that is crucial to OWL, but not scoped at all in MOF-based models. However, the SMOF specification only considers these two aspects, and does not consider how any other ontological functionality can be integrated with MOF. Currently, the concepts defined in SMOF are not supported by EMF.

6.5 Conclusions on Language Design

This chapter provided an analysis of the fundamentals of the three languages Ecore, ORM 2, and OWL 2 in order to derive a suitable language design, capable of fulfilling the defined needs. Based on this analysis, the SCDML language is designed, able to meet the defined needs.

An evaluation regarding the suitability of SCDML to describe a CDM in the space system design context will be given in Chapter 8, and an overall evaluation to the whole approach will be given later in Chapter 9.

7 The Semantic Conceptual Data Modeling Procedure

This chapter focuses on the procedural aspects of defining CDMs, highlighting generally applicable principles, and in the end giving detailed instructions for translating identified information into an SCDML-based CDM. For this purpose, currently employed methodologies related to producing models of a subject of interest are examined. Based on this analysis, a design for a new procedure is proposed that involves a number prescribed steps in order to derive a model from underlying engineering data, answering the third research question:

(RQ3) What is an appropriate procedure for systematically specifying engineering data?

Similar to chapter 6, an analysis of existing procedures is performed at first. While the examined procedures are all published, them being compared in this context is new. The most significant contribution comes in 7.3, where a new procedure is derived that fulfils all of the requirements related to methodological aspects specified earlier in 5.1.

7.1 Survey of Existing Procedures

A number of procedures to derive a model of things or systems in the real world exist. While some of these procedures are rather generic, others provide detailed steps exactly prescribing how to proceed.

The analysis performed in this section picks up on an analysis of modeling methodologies performed earlier (Hennig, et al., 2016b). The analysis at hand is more aligned towards the industrial needs in a CDM, and encompasses a broader scope of procedures, also going into more detail on several aspects.

7.1.1 Software-Driven Procedures

The domain of software design has spawned a number of approaches that prescribe how to develop software systems. Most notable representations of this domain include the Waterfall Model (Royce, 1970), the V-Model (Forsberg & Mooz, 1991), and the Spiral Model (Boehm, 1986). These models focus a lot on the overall approach of how to develop software, not specifically highlighting the software's model, while staying on a very abstract level. As such, these procedures are not applicable to the use case at hand and are not considered further in the evaluation of applicable methodological candidates to support the design of SCDML-based CDMs.

7.1.2 Requirement-Driven Approaches

The architecture that was developed in the course of the EGS-CC project (ESA, 2013a) puts an extensive and detailed CDM at its center. As such, a significant amount of effort has been put into specifying and validating the EGS-CC CDM. In order to realize this, requirements were formulated for what the CDM shall be capable to represent, its internal relations, and what functionalities it shall enable. After the CDM was designed, a mix between validation and verification was performed where actual sample data, selected according to the requirements, was modeled using the CDM. This produced a sample population that could be used to demonstrate that the CDM is able to accurately and completely represent required data.

As the EGS-CC CDM is strongly related to the 10-23 CDM and to RangeDB, this approach is seen as the representative from the RangeDB/10-23 domain.

In that methodological approach to CDM design, specification and verification activities are well covered, but activities such as how to perform the actual CDM design are not considered. This includes the modeling of core constructs, as well as the derivation of constraints. Exhaustiveness of the CDM to be produced can be ensured under the assumption that the requirements are exhaustive, but not based on a sample definition of engineering data.

7.1.3 Methodologies from Fact Based Modeling

The approach of FBM puts a large emphasis on using structured processes to derive CDMs. Such processes have been around for quite some time, including the CSDP (Halpin & Morgan, 2008) for producing models in ORM 2 syntax, as well as the NIAM (Leung & Nijssen, 1998) and CogNIAM (CogNIAM.eu, 2015) methodologies. These approaches all rely on deriving elementary facts from the data to be represented, and utilize these for designing the CDM using a number of prescribed steps.

As representative example from this group, the CSDP has been selected due to its excellent state of publication. The methodology comes along with detailed modeling instructions, with knowledge acquisition and model derivation forming integral parts of the process. However, the methodology is focused on producing models relying on the syntax and semantics of FBM-based models and does not involve CDM validation.

7.1.4 Methodologies for Ontology Design

In ontology engineering, methodological approaches to the design of the model or rather the ontology also play an important role, with the motivation of evolving the modeling of knowledge "from an art into an engineering discipline" (Studer, et al., 1998). In this context, a variety of methodologies have surfaced over the years, most notably METHONTOLOGY (Fernández, et al., 1997), OTKM (Sure, et al., 2004), and the NeOn Methodology (Suárez-Figueroa, 2010).

Being one of the more up-to-date methodologies, the NeOn Methodology stands as representative example. While it discusses aspects such as ontology management activities, ontology development, and support activities, no consideration is given to detailed design decisions or procedures. Furthermore, verification or validation activities are not considered extensively.

7.1.5 Procedure Survey Conclusion

The survey made visible that none of the currently existing procedures are able to fully provide the required functionality. The overall functional coverage in respect to defined needs is summed up in Table 7.1.

Table 7.1: Summary of data modeling procedure analysis

Procedure Feature	Derived from REQ	Supported by		
		RangeDB/10-23 Methodology	Conceptual Schema Design Procedure	NeOn Methodology
Overall Process	2-1	not apart from requirements and verification	yes	yes
Model Derivation Procedure	2-2	no	only into ORM 2 model syntax	no
Constraint Exhaustiveness Ensuring Procedure	2-3	no	no	no
CDM Validation Procedure	2-4	yes	no	no

7.2 Procedure Design Discussion

Four features that have to be supported can be derived from the requirements defined previously in section 5.1.

Regarding a detailed model derivation procedure, this is a feature neither fulfilled by the methodology revolving around 10-23, nor by the NeOn Methodology. The CSDP (Halpin & Morgan, 2008) supports this activity in principle, but is tailored towards producing CDMs in a different syntax. Consequently, this methodology will be used and adapted in order to fit the needs exactly as defined. Concepts missing in the existing methodology, but required by the SCDML are newly developed accordingly.

Regarding constraint exhaustiveness, none of the analyzed procedures provide the required functionality. Consequently, new aspects, tailored towards the constraints existing in SCDML, need to be developed. Furthermore, the SCDMP loans and adapts several procedural aspects from the methodology associated with FAMOUS (Valera, 2014), which was excluded from the overall analysis due to an inadequate overall publication situation.

As CDM validation procedure, the approach used in the EGS-CC project (ESA, 2013a) is simplified. This adaption involves de-coupling it from requirements, and directly using the facts that served as source for the CDM's derivation as base data for validation. The central principle of the existing approach, where an agreed upon set of data is used to design a CDM, is preserved and even strengthened in the SCDMP.

The SCDMP requires picking up concepts from existing methodologies, and combining them with new, specifically developed, procedural concepts. In order to achieve a consistent integration, the overall process required by REQ-1-1 is defined from scratch.

7.3 SCDMP Design

The Semantic Conceptual Data Modeling Procedure (SCDMP) deals with producing an SCDML-based CDM. As such, defining the relations of an *SClass* and a corresponding *OWLClass* is scoped, however modeling of the ontological partition of the model, beyond the classes connecting to the SCDML-based partition, is not treated by the SCDMP.

As stated earlier, the SCDMP picks up on a variety of existing characteristics from existing procedural approaches towards conceptual modeling, being the CSDP (Halpin & Morgan, 2008), and the FAMOUS methodology (Valera, 2014). Consequently, this should be regarded more of an integration, adaption, and reconfiguration of already existing concepts.

7.3.1 SCDMP Overview

The overall procedure is divided into seven main steps that each involve a number of sub-steps. The overall process is outlined in Figure 7.1.

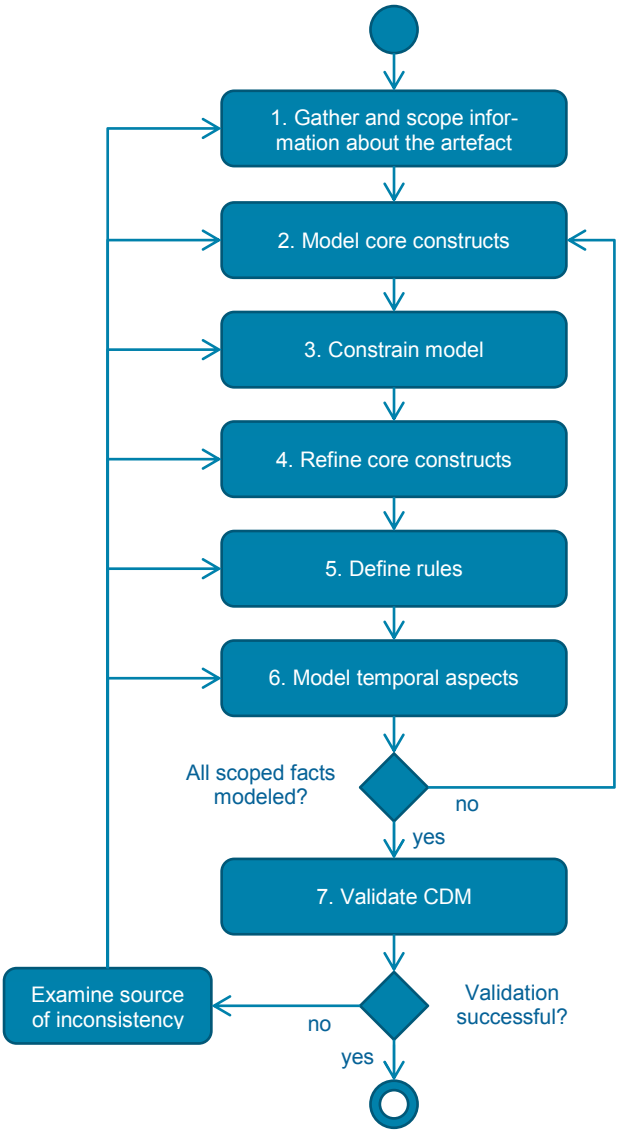


Figure 7.1: SCDMP Overall Process

The process covers gathering domain information relevant to the CDM, up to CDM verification, including the iteration that has to be performed in the event of unsuccessful CDM validation. A detailed description of each step and its sub-steps is provided in section 7.3.3.

7.3.2 SCDMP Key Principles

Before going into the detailed description of procedure steps, a number of key principles have to be detailed that form central pillars of the SCDMP and occur inside many of the main steps. These include decomposing domain information into elemental facts, using structured processes to derive constraints, and following a given specification and evaluation approach.

7.3.2.1 Exact Scoping and Validation Guidelines

One key principle of the SCDMP is the reliance on clear scoping of the information that shall be represented in the CDM a priori to its design. To accomplish this, a process is pursued where, from the descriptive source of the artefact to be modeled, the information that is necessary to be represented in the CDM is explicitly selected. This ensures that no unnecessary information is put into the CDM, and serves as the basis for validation later on.

For validation, the facts that have been derived from the selected information are modeled using an application implementing the CDM, ensuring that the required facts can indeed be represented. Should this validation fail for specific facts that are not able to be correctly represented, or not able to be represented at all, an iteration has to be performed on the CDM and its implementation.

7.3.2.2 Principle of Factual Decomposition

For deriving the facts from selected information, a specific approach where the information is decomposed into *elementary facts* is proposed.

An *elementary fact* is a statement about an object. The most basic elementary fact is of unary or Boolean type, such as *MagSat flies*, asserting that a particular object has a specific property (Halpin & Morgan, 2008). Most frequently, relationships involve two objects, e.g. *MagSat orbits Earth*, stating that two objects participate in a relationship together (Halpin & Morgan, 2008). The statement *MagSat surveyed the South Atlantic Anomaly on the 6th of December 2016* would be an example of a ternary fact. Facts can exhibit any arity, however arities greater than three are rather uncommon (Halpin & Morgan, 2008).

The term elementary indicates that the fact cannot be split into smaller parts of information without losing a portion of its original information in the process. Elementary facts do not contain logical connectives such as *NOT*, *AND*, *OR*, or *IF* (Halpin & Morgan, 2008). An example for a fact that is not yet fully reduced to an elementary one would be *MagSat was launched with Ariane 5 in the year 2016*. This fact can be split further into two facts, being *MagSat was launched with Ariane 5,* and *MagSat was launched in the year 2016*. Once both of these facts are combined again, no information loss occurred, which is a reliable sign towards the fact not being elementary.

7.3.2.3 Translation of Elementary Facts into Model Elements

For going from elementary facts about the domain to be modeled to concrete model elements, a derivation process can be used (Hennig, et al., 2016b). This process takes the elementary facts in scope and transforms them into *SClasses*, *SAttributes*, and *SReferences*. For this purpose, as an initial step, several similar facts are written down, for example:

MagSat contains Battery.
MagSat contains Star Tracker.
Star Tracker contains Star Tracker Electronics.
Star Tracker contains Star Tracker Sensor.

In a second step, the fact is split into constant and variable parts. The constant part stays identical for all facts, while the variable part may vary between examined facts:

MagSat	contains	Battery.
MagSat	contains	Star Tracker.
Star Tracker	contains	Star Tracker Electronics.
Star Tracker	contains	Star Tracker Sensor.
variable part	*constant part*	*variable part*

The variable parts of the fact denote different objects that play a role in the domain to be modeled. These objects translate to *SClasses*, which should be created in the course of the procedure. The constant part of the fact denotes a role a class plays in some fact, that can either be an *SAttribute* or an *SReference*. If an object plays a role with another object captured by the procedure, the fact is modeled using an *SReference*. If the object does play a role with something else that is not a first order object per se, such as a numeric value, a name, or some text, then it translates to an *SAttribute*. However there are exceptions, where in some cases these things might require to be treated as first order objects, e.g. when dates are modeled and several classes might require to be evaluated if they play a role in the same date. In the example at hand, the fact is translated the following way in the model:

MagSat	contains	Battery.
MagSat	contains	Star Tracker.
Star Tracker	contains	Star Tracker Electronics.
Star Tracker	contains	Star Tracker Sensor.
variable part	*constant part*	*variable part*
System Element	contains	System Element

7.3.2.4 Constraint Derivation and Exhaustiveness

For refining the model and providing it with sufficient semantic strength, processes to derive a number of constraints are provided. If all the processes are performed for each fact, all applicable constraints of the domain are captured in the model in the end. This systematic approach ensures that no constraint is overlooked and was already proposed in earlier works (Hennig, et al., 2016b). In this section, the approach is extended, supporting the derivation of more constraints, with additional methodological possibilities.

7.3.2.5 CDM Patterns

In the course of numerous CDM modeling activities over the last years (ESA, 2012a; Eisenmann & Cazenave, 2014; Fischer, et al., 2014) numerous patterns that surface time and again in CDMs have been identified.

Product Structure Pattern

As key pattern in this context, where technical systems are the objects of interest, the *Product Structure Pattern* has been proven to be extremely helpful. In this case, a central structure that represents key building blocks of the system is used around which the rest of the model revolves. Usually, *System Elements* form a sometimes more, sometimes less strict hierarchy. Data in the model can always be attributed to belonging to exactly one *System Element* that is part of the *Product Structure*, hinting at the second important pattern, the *System Element Aspect*.

System Element Aspect

The *System Element Aspect* forms a part of information about a *System Element* which supplies viewpoint-specific information about it. This can be information such as a description of its behavior, a requirement, or physical data about the element.

Element-Port-Interface Pattern

Another aspect is the *Element-Port-Interface Pattern* that is frequently used when something goes in and out of objects. These inputs and outputs can take a number of forms, for example information, physical substances, or electrical signals. The pattern

involves modeling the *System Elements* that are considered by the flow, modeling their ports, and modeling the interfaces between two ports For facts of arity greater than two, an addition class representing the fact has to be created in the CDM.

Container-SubElement-Pattern

Another recurring structure is the *Container-SubElement-Pattern*. In this pattern, a concept with the role of container contains a number of other concepts, which in turn form a hierarchy. One kind of container exists that usually contains one type of elements, but there may also be cases where several types of elements are contained by a single container.

7.3.2.6 Naming Conventions

The following naming conventions have been defined for models based on the SCDMP. Most of these patterns should not come as a surprise, as these have been established throughout numerous programming and modeling publications.

- The names for *SClasses* are defined using camel case with an initial capital letter, or rather upper camel case. This leads to names such as. *ProductTree*, *DiscreteModel*, or *ElectricalInterface*. Although an *SClass* often represents a number of instances in the SM later on, the name of the *SClass* should be formulated in singular.
- The names for *SAttribues* and *SReferences* are defined using lower camel case, e.g. *subElements* or *attributes*.
- *SAttributes* with type Boolean should not have an "is" in front of the attribute's name, e.g. not *isLogicalElement*, but *logicalElement*.
- *SReferences* that may reference numerous *SClasses*, based on the defined constraints, shall reflect the plurality in their name, e.g. *subElements*, *transitions*, or *constraints*.

7.3.3 SCDMP Detailed Steps

This section describes the detailed sub-steps of the SCDMP.

7.3.3.1 Gather and Scope Information about the Artefact

The initial main step deals with identifying the domain artefact to be modeled, gathering information about it, and scoping the information. The key starting point for the procedure is selecting an artefact to be modeled, which is then translated into elementary facts (Figure 7.2).

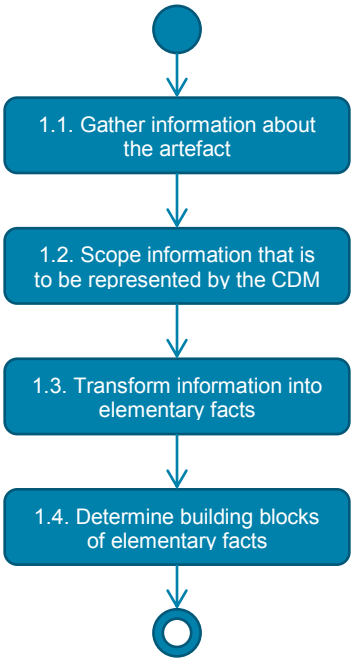


Figure 7.2: SCDMP information gathering process

Gather information about the artefact

The nature of this step depends significantly on the artefact to be modeled. If the artefact is formally described, e.g. in a specification document, this documentation can provide the required input information. If no formal specification about the artefact is available, a description based on prose text can also be utilized. The CDM of an artefact can also be reverse-engineered from an existing model, of which the meta-model is not accessible.

Scope information that is to be represented by the CDM

In many cases, not all information about the artefact in its documentation is required to be present in its representation in the CDM. As such, after sufficient documentation about the artefact has been gathered, the information of relevance is to be selected from the artefact's documentation. This is an important step, as the information selected will serve as basis for validation later on.

Transform information into elementary facts

Information about the artefact is to be transformed into elementary facts, according to the procedure described in 7.3.2.3. This is done by performing the following steps:

- Collect and write down a fact contained in the scoped source description
- Split the fact into smaller facts
- Check if the meaning of the smaller facts is identical to the meaning of the original fact.

If a loss of meaning occurred, the original fact was already elementary and cannot be split. If the meaning of the split facts does not change, the original fact was not yet elementary. In this case, the procedure can be repeated for each of the split up facts.

Determine building blocks of elementary facts

After the information has been transformed into elementary facts, their building blocks in terms of constant parts and variable parts have to be identified, as explained in 7.3.2.3.

7.3.3.2 Model Core Constructs

This procedure step involves translating the information represented by a selected elementary fact into the CDM, based on SCDML syntax (Figure 7.3).

Create SPackage for the artefact

The initial step to be performed involves creating an *SPackage* for the artefact to be modeled. Later on, it may become necessary to create *sub-SPackages* if the artefact is found to be rather complex and elaborate. As a rule of thumb, after an artefact consists of more than 12 *SClasses*, it should not be contained by one package alone, but be distributed among several *sub-SPackages*.

Assert SClasses to the model and add descriptions

The variable parts of the elementary facts have to be examined regarding whether they denote first-order objects in the domain. A first-order object is one that is required to be referred to by other objects. If this is the case, an *SClass* has to be created for the class of objects represented by the variable part of the fact.

Assert SAtrributes

For objects in the fact that are not required to be referenced to by other objects, an *SAttribute* may be created. This involves aspects such as:

- Names occurring in the fact
- Integer quantities occurring in the fact
- Floating point properties occurring in the fact
- Date and time occurring in the fact
- String-based statements occurring in the fact

The constant part of the fact becomes the name of the *SAttribute*, while the object in its variable part has to be examined regarding its type, and the according type modeled as the type of the *SAttribute*.

Assert SReferences

For *SReferences*, the constant part forms the reference itself, or rather drives its name. As the elementary fact stands, the first variable part forms the owner of the *SReference*, while the second variable part forms its type.

Identify SReference opposites

While elementary facts come with a default reading direction, the inverse reading direction of all facts should be examined. In order to do this, the fact is turned around, by switching the variable parts and rephrasing the constant part accordingly. If the inverse reading direction is of relevance to the domain-view of the artefact, it is to be added to the CDM and marked as being opposite to the original *SReference*.

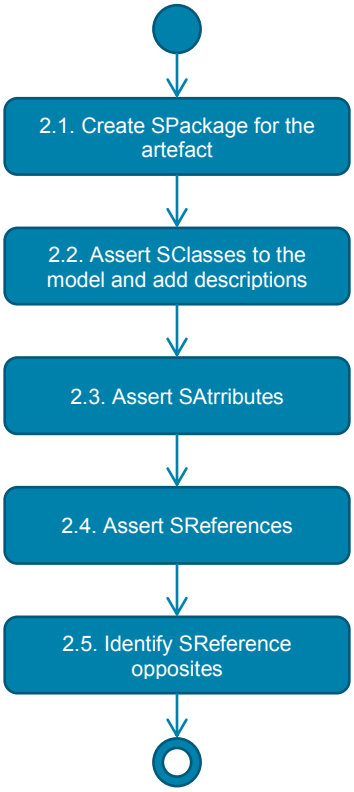


Figure 7.3: SCDMP core structure modeling process

7.3.3.3 Constrain Model

In order to give the required semantic accuracy, a range of constraints can be modeled, also directly derived from the artefact's documentation (Figure 7.4).

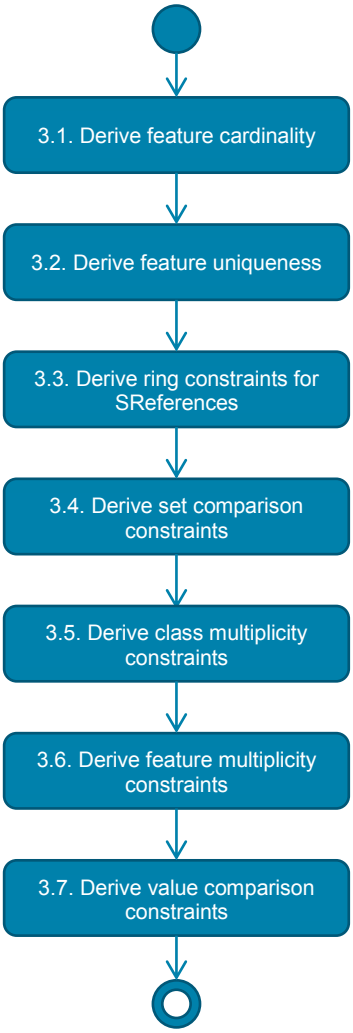


Figure 7.4: SCDMP procedure for deriving constraints

Derive feature cardinality

Up to now, only the features themselves have been defined, not their *cardinality*. In order to derive the cardinality in terms of *lower* and *upper bound*, a process can be followed. For each fact or rather *SStructuralFeature*, the question should be asked if it is necessary that the feature always has at least a number of values. If it can be ascertained from the artefact's documentation or from another source, such as a domain expert, that this is indeed the case, a *FeatureCardinalityConstraint* has to be introduced with the *lowerBound* set accordingly. In order to derive the *upperBound* of the feature, it should be asked if the feature is limited by having a maximum number of values. If so, an *upperBound* representing this maximum should be set. The complete control flow to derive the constraint and its bounds is illustrated in Figure 7.5.

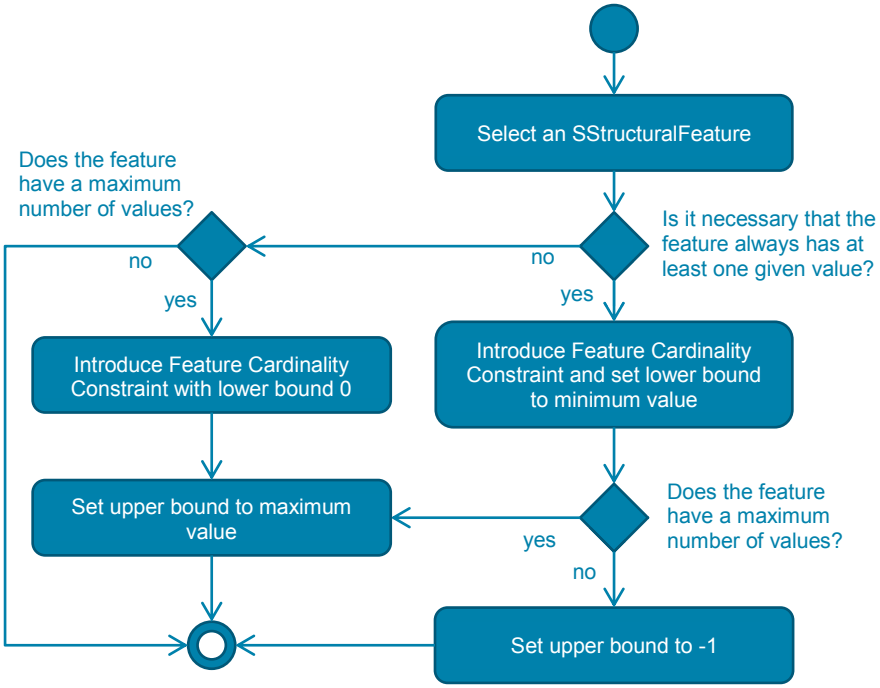


Figure 7.5: SCDMP FeatureCardinalityConstraint derivation procedure

Derive feature uniqueness

Uniqueness of *SStructuralFeatures* is applicable to any *SAttribute* or *SReference* that has a *FeatureCardinalityConstraint* with an *upperBound* greater than *1*. As such, each *SStructuralFeature* to which these properties apply has to be examined regarding uniqueness. In order to do so, a hypothetical fact, consisting of an exact copy of the fact in question is to be produced. Then, the question should be asked if it is possible that a second, identical fact can coexist in the model without invalidating it. If the answer to the question is yes, the *SStructuralFeature* representing the fact may be *nonunique*, resulting the setting of the *unique* property to *false*. If the answer is no, the fact is *unique* and the *unique* property be set to *true* (Figure 7.6).

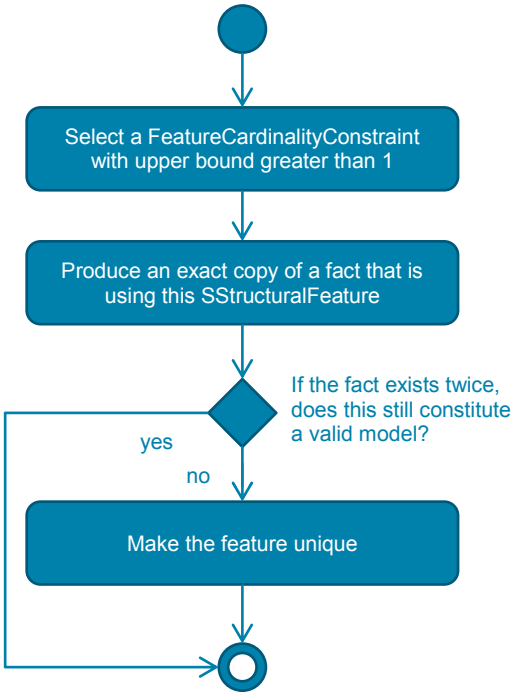


Figure 7.6: SCDMP procedure for determining feature uniqueness

Derive ring constraints for SReferences

SReferences contained by an *SClass* that is also their *type* may have a *RingConstraint* associated with them. This constraint states that not all populations for this *SReference* form a valid model. A *RingConstraint* can have a variety of characteristics that can be determined using a specific process. In order to do so, the fact has to be examined again. The first object in the fact is denoted as *A*, the second as *B*. The reference

lying behind the constant part of the fact is denoted as *R*. Consequently, a series of questions should be asked in order to determine the exact properties of the *RingConstraint*. For example:

- *Reflexivity*: Is it necessary that, if A Rs B, then A also Rs A? If the answer is yes, then the *RingConstraint* is *reflexive*. *Reflexivity* can be combined with additional characteristics, them being *symmetric*, and *transitive*
- *Symmetry*: Is it necessary that if A Rs B, then B also Rs A? If yes, then the constraint is *symmetric*, with additional possible characteristics being *irreflexive*, *intransitive*, *reflexive*, and *transitive*.

All necessary questions and possible *RingConstraint* properties are listed in Table 7.2.

Table 7.2: Summary of Ring Constraint Properties

Property	Question	Additional Properties
Reflexivity	Is it necessary that, if A Rs B, then A also Rs A?	Symmetry Transitivity
Symmetry	Is it necessary that if A Rs B, then B also Rs A?	Irreflexivity Intransitivity Reflexivity Transitivity
Transitivity	Is it necessary that if A Rs B and B Rs C, then A also Rs C?	Irreflexivity Asymmetry Acyclicity Reflexivity Symmetry
Acyclicity	If A Rs B and B Rs C, is it forbidden that C Rs A?	Transitivity Intransitivity
Irreflexivity	Is it forbidden that A Rs A?	Symmetry Transitivity
Asymmetry	Is it forbidden that if A Rs B, then B also Rs A?	Transitivity Intransitivity
Intransitivity	Is it forbidden that, if A Rs B and B Rs C, that A also Rs C?	Asymmetry Acyclicity Symmetry

Table 7.3 summarizes all sensible combinations of *RingConstraint* properties. As evident from the table, the relations are symmetric, resulting in the sequence in which the examination is done not influencing the outcome.

Table 7.3: Valid combinations of Ring Constraint properties

	Reflexivity	Symmetry	Transitivity	Acyclicity	Irreflexivity	Asymmetry	Intransitivity
Reflexivity		•	•				
Symmetry	•		•		•		•
Transitivity	•	•		•	•	•	
Acyclicity			•				•
Irreflexivity		•	•				
Asymmetry			•				•
Intransitivity		•		•		•	

Derive set comparison constraints

Another important constraint group is given by *SetComparisonConstraints*. *SetComparisonConstraints* compare different *SStructuralFeatures*. Here, two modes can be distinguished, where the constraint merely compares if the features are *set or unset*, or if the constraint should also compare the *values* of the features. The *SetComparison Constraint* applies to the following model constellations:

- Two or more optional *SReferences* of same owner and the same type (applicable for *SetUnset* type and *ComparePopulations* type)
- Two or more optional *SAttributes* of same owner and the same type (applicable for *SetUnset* type and *ComparePopulations* type)
- Between two or more optional *SStructuralFeatures* of the same owner (*SetUnset* type)

As *SetComparisonConstraints* come in different forms, such as *Subset Constraint*, *EqualityConstraint*, or *ExclusionConstraint*, a process exists in order to determine their exact nature. A simple process applies to *Set ComparisonConstraints* between two *SStructuralFeatures*. For more than two comparisons, the process becomes more complex.

For this process, a truth table needs to be created containing the facts from the sample population of the two *SStructuralFeatures* that should be compared (Table 7.4). The table contains four facts, where in the first row, both facts are true regarding the

SStructuralFeatures. In the second row, values are only true for the first feature, in the third row values for the second feature, and in the fourth row no values exist for both features, evaluating to false.

Table 7.4: Fact combinations for derivation of SetComparisonConstraints

Fact Population	Fact X	Fact Y
X ∧ Y	true	true
X ∧ ¬Y	true	false
¬X ∧ Y	false	true
¬X ∧ ¬Y	false	false

Now, all possible combinations of facts are played through and the question is asked which of these combinations is possible at all. This is realized using a second truth table (Table 7.5). In this table, T denotes true or valid combinations of the facts in the first column, while F denotes that this is false or not possible.

Table 7.5: Truth table for deriving SetComparisonConstraints

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
X ∧ Y	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
X ∧ ¬Y	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
¬X ∧ Y	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
¬X ∧ ¬Y	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Answer											-	-	-	-	-	-
Consequence	No constraint	Inclusive-or	X superset and Y subset	Mandatory up (card >0)	X subset and Y superset	Mandatory down (card >0)	Equality	Mandatory both (card >0)	Exclusion constraint	Exclusive -Or Constraint	Not applicable					

In the initial column, it is expected that all four facts can co-exist at the same time, i.e. may be valid at the same time and thus denote a valid model. Consequently, no constraint is required. For the second combination of facts, the first fact is valid, as well as the second and the third. However, there may not be any situation where none of the facts are true, leading to the necessity to specify an *InclusiveOrConstraint*. This procedure allows deducing the *InclusiveOr-*, *Subset-*, *Mandatory-*, *Equality-*, *Exclusion-*, and *ExlcusiveOr-Constraint*.

The last six columns continue the logical chain, but are not applicable, as they constrain the model in a way where it cannot be sensibly populated. All combinations of facts given by columns K to P would not enable a population of facts in the model.

Derive class multiplicity constraints

An additional type of constraint is formed by the *Class Multiplicity Constraint*. This constraint specifies that for one *SClass*, only a specific number of instances may exist in the model. In many cases, if the multiplicity is restricted, it is restricted to one. If only one object can exist for an *SClass* at the same time it is to be determined from the documentation, or, if this cannot be done, through domain experts. The procedure is defined in Figure 7.7 and has to be executed for every single *SClass*.

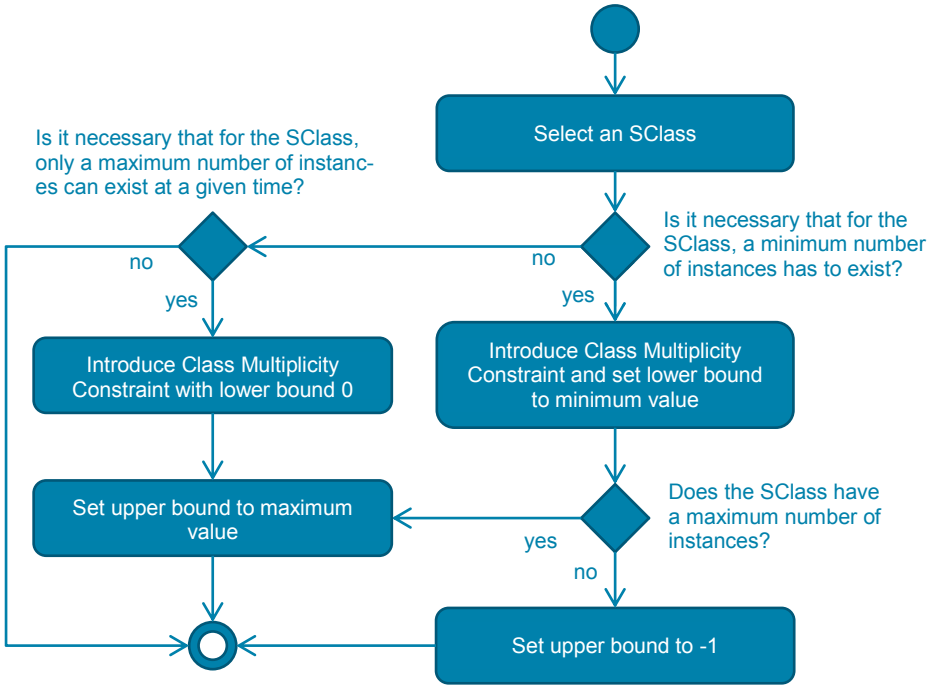


Figure 7.7: SCDMP procedure for deriving ClassMultiplicityConstraints

Derive feature multiplicity constraints: In addition to class multiplicity, *SStructuralFeatures* can also have *MultiplicityConstraints*. In this case, only a given number of instances may exhibit a value for this feature. For example, only one *Person* may play the role of *isMagSatHeadProjectManager* at any time, requiring the Boolean *SAttribute* to have a *FeatureMultiplicityConstraint* with *upperBound* 1. The procedure to derive this is analogue to the one described in Figure 7.7.

Derive value comparison constraints: Between two *EAttributes* that have a type that can be numerically compared, a *ValueComparisonConstraint* may be defined. These constraints can be used, for example, to determine if an integer value is equal to another integer value, if a date is greater than another date, or if a floating point value is less than or equal to another floating point value. For this purpose, the compare operands *greater than, greather than or equal, equal to, less than or equal, and less than* are provided.

7.3.3.4 Refine Core Constructs

For refining the concepts modeled until now, the steps in Figure 7.8 can be taken.

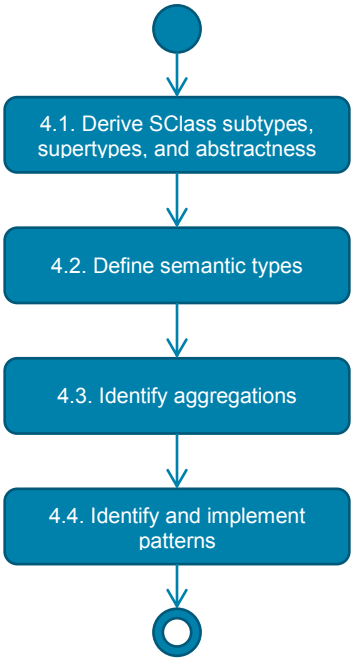


Figure 7.8: SCDM core structure refinement process

Derive SClass subtypes, supertypes, and abstractness

In SCDML, *SClasses* may also form taxonomies. Defining class taxonomies is an art in itself and depends on numerous factors, especially on the experience of the modeler and the concrete problem at hand. Consequently, only very rough guidelines can be provided for this activity, leaving a good deal of creative freedom to the modeler. When unsure whether a certain *subtype* or *supertype* should be defined, the guidelines may be consulted. The procedure for superclassing is described in Figure 7.9. In principle, the procedure has to be iterated over each set of *SClasses*. However, for practical reasons, it is more effective to quickly go over all *SClasses* and to spot those that have similarly named *SStructuralFeatures*. If the features are also identical in their semantics, a common superclass may be defined.

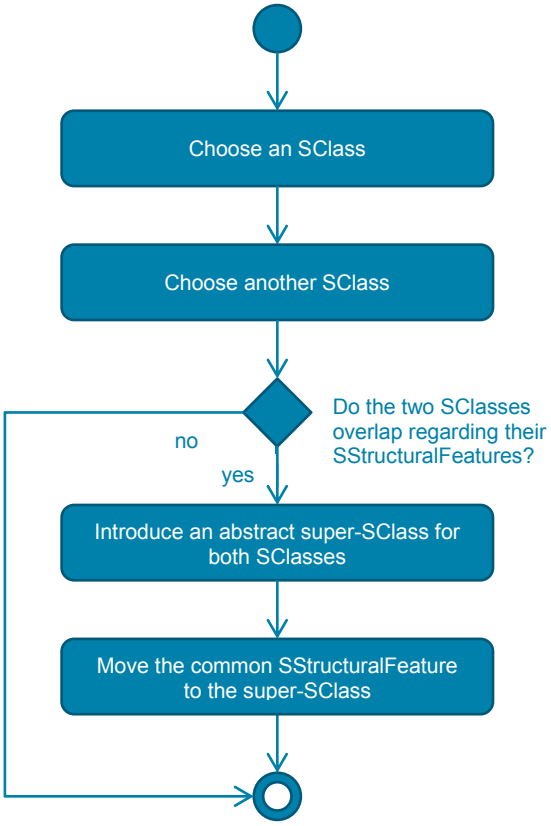


Figure 7.9: SCDMP procedure for determining applicable supertypes

For determining subclassing, the procedure described in Figure 7.10 can be followed. Each *SClass* is examined whether it has an optional *SStructural Feature*, i.e. which applies in cases where there is a feature without *Feature CardinalityConstraint*, or in cases where there is one with a *lowerBound* of 0. If this is the case, it can be examined whether this optional feature is only applicable to a subset of the potential *SClass* population. If this is the case, a *sub-SClass* should be introduced, and the feature moved towards it.

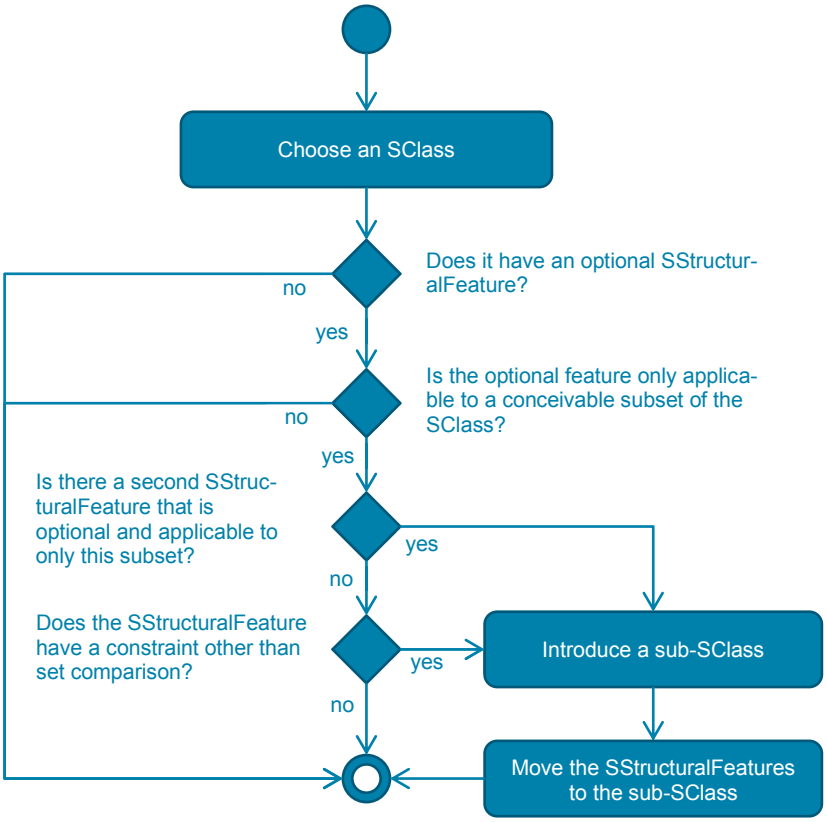


Figure 7.10: SCDMP procedure for determining applicable subtypes

Define semantic types

The *semanticTypes* take on two roles. On the one hand, they form the bridge to the ontology-based part of the CDM, while on the other hand they define the aspects a model element can have. If a description of the *SClass* also exists in the ontology world (not directly scoped by this procedure), then it should be added as its *semantic*

Type. Any aspect-based information that is to be added to the *SClass* should be done so by using a *semanticTypes* reference to another *SClass* defining the aspect.

Identify aggregations

SClasses can form hierarchies in terms of *composition* or *aggregation* relationships. For determining if this is necessary, the artefact documentation should be consulted again. If a hierarchy of elements becomes evident in the documentation, the *SReference* behind the constant part of the fact involving the hierarchy has to be marked as either being *shared* or *composite*. For determining which one, the process described in Figure 7.11 can be used.

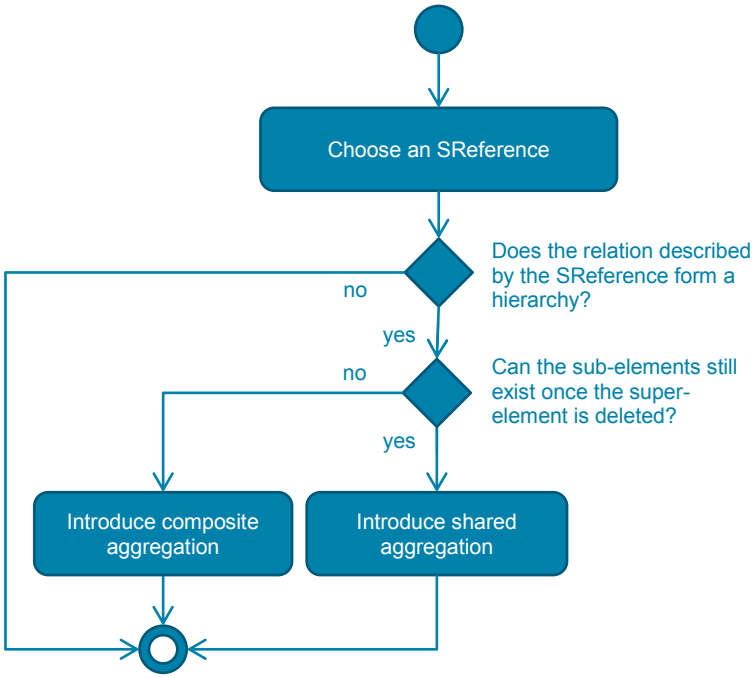


Figure 7.11: SCDMP procedure for determining SReference hierarchies

Identify and implement patterns

As last sub-step, after several *SClasses* have been created, it may become evident that one of the patterns described in 7.3.2.5 has been implicitly used, or that constructs almost matching this pattern have been modeled. If this is the case the pattern should be implemented accordingly.

7.3.3.5 Define Rules

SCDML and the SCDMP permit the modeling of two kinds of *Rules* (Figure 7.12). On the one hand, *OCL statements* can be modeled that are used as consistency constraints in the traditional sense. On the other hand, SCDML allows the modeling of *FunctionalRules*. The purpose of these rules is to represent functional dependencies between different model elements, such as the necessity for a specific class to exist, based on given conditions, or the necessity for certain values to hold. The difference to *OCL constraints* is that, while an *OCL constraint* mainly checks if the required conditions hold, the *FunctionalRules* enforce the condition.

As the modeling of rules is extremely dependent on the use case, no detailed instructions are given here. This will be detailed in 8.5.2.

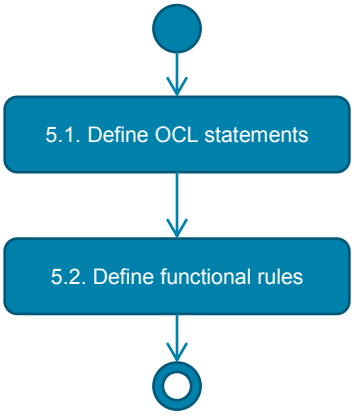


Figure 7.12: SCDMP rule definition procedure

7.3.3.6 Model Temporal Aspects

For modeling temporal aspects of the underlying domain data, the concept of *TemporalCriteria* is offered that represent certain milestones of the domain's underlying process. After temporal criteria have been derived, constraints are allocated to them stating at which point in time a constraint is valid, i.e. has to be evaluated. The process of introducing temporal aspects to a CDM is outlined in Figure 7.13.

Model Temporal Criteria

For modeling *TemporalCriteria*, the documentation of the artefact is to be consulted again. However, as the artefact itself may not have enough information as to grasp it in context of the domain, other domain documentation focused on the engineering process might become necessary to be evaluated. In general, a *TemporalCriterion*

represents a milestone, standing in the middle or at the end of a given phase in the artefact's lifecycle. After these milestones have been captured as *TemporalCriteria*, their relation to other *TemporalCriteria* that serve as *super-* or *sub-criteria* has to be examined. Due to the nature of two or more engineering processes often being very different, it is difficult to properly formalize a process that can be used for deriving *TemporalCriteria*. Instead, a couple of hints or rather heuristics can be given, pointing to where they can commonly be found:

- The most general level of *TemporalCriteria* is represented by the milestones of the overall engineering process that is used to design the system. There, typical milestones such as end of specification, finalization of design, start of production, or end of testing serve as *TemporalCriteria*.
- The milestones of the discipline-specific processes involved in the system's design form a second detailing level to *TemporalCriteria*. They are usually related to the *TemporalCriteria* of the overall process, as they occur within it. This relation can be expressed using the *superTemporalCriterion* reference.
- If discipline-specific processes also involve a number of sub-processes that have their own lifecycle considerations, further levels of decomposition can be introduced for *TemporalCriteria*.

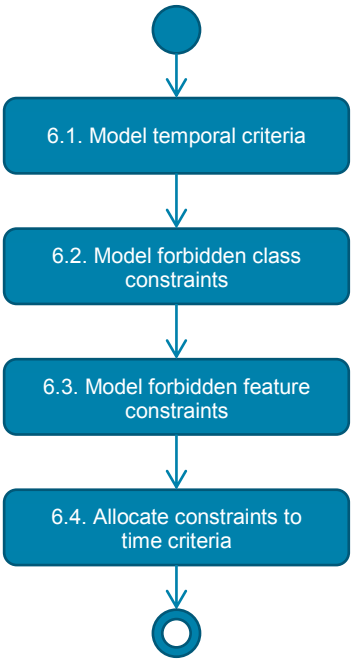


Figure 7.13: SCDMP temporal aspects modeling procedure

Model Forbidden Class Constraints

After the lifecycle aspects of the artefact and its process have been understood and *TemporalCriteria* are modeled, *ForbiddenClassConstraints* may be added to the CDM. A *ForbiddenClassConstraint* may be used when, in selected phases of the artefact's lifecycle, having information represented by one of its *SClasses* is explicitly excluded. More specifically, if the *ForbiddenClassConstraint* applies to an *SClass* for a given *TemporalCriterion*, this means that no instances of this *SClass* may exist at that point.

Model Forbidden Feature Constraints

A *ForbiddenFeatureConstraint* may also be modeled, working analogous to the *ForbiddenClassConstraint*. The *ForbiddenFeatureConstraint* states that a given *SStructuralFeature*, i.e. an *SAttribute* or an *SReference*, may not have any value set at that point.

Allocate constraints to time criteria

In order to complete the modeling of *TemporalCriteria*, each *Constraint* is required to be evaluated regarding its applicability to one or more *TemporalCriteria*. If the *Constraint* is valid at each point of the artefact's lifecycle, no *TemporalCriteria* have to be allocated.

7.3.3.7 Validate CDM

Validating the CDM is done using a two-step approach. Initially, it is ensured that the CDM is able to represent all scoped facts. After this, it is examined whether the consistency checking enabled by the CDM is sufficient. The procedure to do this is given in Figure 7.14.

Model previously scoped facts

For validating the CDM, the facts derived from the information about the artefact to be modeled have to be put into the model, more specifically into the application implementing the CDM. Consequently, this step can only be performed after an initial version of the application has been deployed, or at least a prototypical generic implementation of the CDM is made available for validation purposes.

If a selected fact can be correctly represented in the SM, this part of the CDM is considered validated. If this is not possible, an examination has to be performed as to why this is the case, zooming in on the modeling error in the CDM. After the error has been corrected and a new implementation of the CDM is available, another validation run can commence, re-trying putting the fact into the modeling application. As soon as all initially selected facts are able to be represented by the modeling application, the validation step is successful.

Model artificially created inconsistent facts

In the second validation step, inconsistent facts are deliberately created and put into the model. This step is used to ensure that the provided data structures are not only sufficient to accommodate the scoped information, but also suitable to store it consistently. If all deliberately modeled inconsistent facts can be identified through the constraints introduced earlier, the CDM is successfully validated.

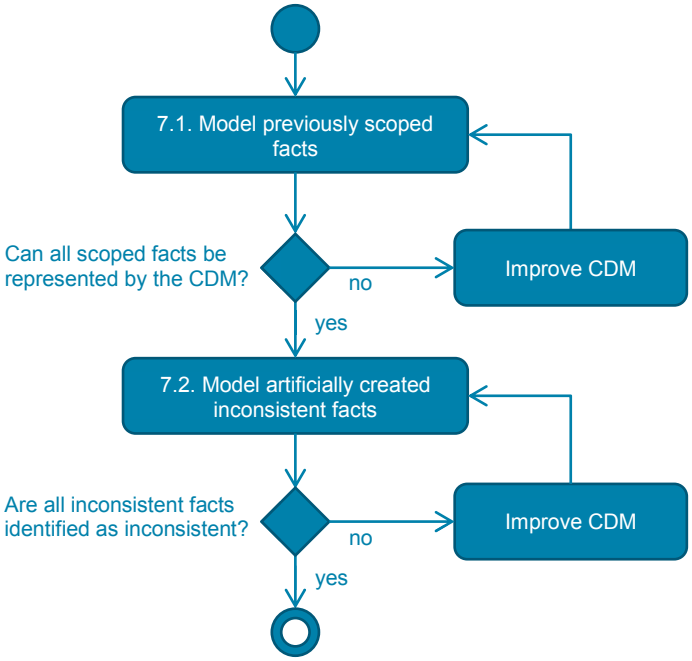


Figure 7.14: SCDMP validation procedure

7.4 Concluding on Procedure Design

This chapter discussed the design of a procedure used to produce CDMs in the context of space system design. This procedure supports central requirements identified earlier on the derivation of data structures, such as a guided overall process, procedures to ensure that the CDM is defined exhaustively, and guidelines to validate the produced CDM.

The motivation behind providing these detailed procedures lies in moving the CDM closer to the actual engineering process which it should support in the end, and in ensuring that the data of this process is adequately represented in the required scope

and detail. A validation and application of this procedure is provided in the following chapter.

8 The Model-Based Space System Engineering CDM

This chapter describes the design of the Model-Based Space System Engineering Conceptual Data Model (MBSE CDM), modeled in SCDML, and derived using SCDMP. This includes an outline of the CDM's origin, the improvements made on existing data structures by applying SCDML and the SCDMP, and the extension with new concepts derived from the earlier defined requirements. As such, this chapter answers the fourth research question:

(RQ4) What is an appropriate structure and content of the system model specification in order to meet defined needs?

The MBSE CDM picks up on the concepts defined for the original 10-23 data model (ESA, 2011a). Furthermore, the improvements made on the 10-23 CDM in follow-on activities (Fischer, et al., 2014; Eisenmann & Cazenave, 2014), and other related projects (ESA, 2013a) are considered as being additions to the original 10-23 CDM. The description of the MBSE CDM content is spread across three areas in respect to the original 10-23 CDM, being concepts that are improved from 10-23, concepts that are confirmed as being already fully sufficient in 10-23, and concepts that are newly introduced in the MBSE CDM.

Evaluation of the MBSE CDM is performed in Chapter 9, where it is used to model a concrete SM on M0 level, solving concrete engineering problems.

8.1 MBSE CDM Architecture

As outlined in the SCDML architecture in 6.3, the MBSE CDM is made up of two constituent CDMs, based on two distinct languages, forming one virtual CDM. Virtual in this context means that the MBSE CDM does not exist as one distinct entity, but that it is defined by combining its constituent CDMs, which reference each other. One constituent, the MBSE Object-Oriented CDM, is based on SCDML, instantiating the SCDML language constructs. Consequently, this part of the CDM offers object-

oriented semantics in its SM. The other constituent of the MBSE CDM, made up by the MBSE Ontology, instantiating concepts from OWL 2, offers ontological semantics in its respective SM. While the MBSE OO-Model is modeled in the modeling environment offered by SCDML, the MBSE Ontology is created within an ontological modeling tool.

This architecture is given in Figure 8.1 and forms a concrete instantiation of the generic design outlined in Figure 6.2 in Chapter 6. As such, it contains the concrete names of the CDMs outlined in this chapter, along with their instantiations that will be detailed later in Chapter 9.

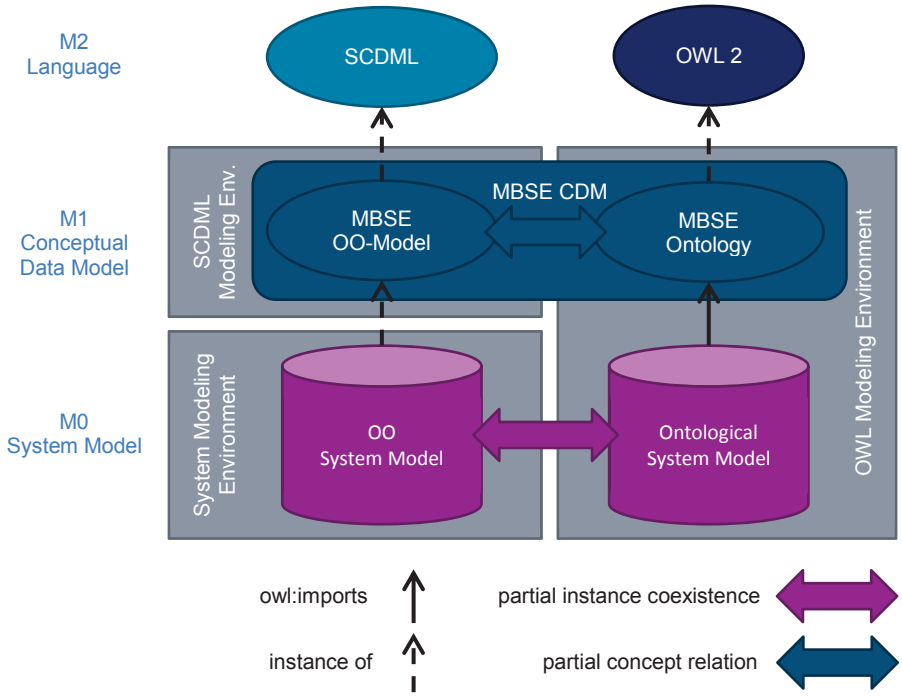


Figure 8.1: MBSE CDM constituents and relation to M2 and M0 levels

The MBSE CDM is designed for addressing concrete requirements, derived from concrete engineering problems in the MBSE context (see section 5.1). Out of the requirements addressing M0 level functionality, a specific modeling approach (object-oriented or ontological) may be more suited to address a requirement. As such, different concepts in the MBSE CDM focused on addressing specific requirements reside within one of the two constituent CDMs, based on the modeling approach that is most suitable to solve a given problem. Table 8.1 gives an outline of the main building

blocks of the MBSE CDM and in which technical domain they are situated. There are cases where a concept utilizes descriptions from both the object-oriented, and the ontological domain. Why a specific CDM concept was allocated to a specific CDM constituent is explained throughout this chapter.

Table 8.1: MBSE CDM main concepts allocation

Concept	MBSE OO-Model	MBSE Ontology
Specification	•	
Product Structure	•	•
Functional Design		•
Operational Design	•	
Topological Design	•	•
Engineering Activity Support		•
Physical Properties		•
Verification	•	•

8.2 MBSE Ontology Characteristics

The MBSE Ontology does not consist of a single ontology, but of several ontologies. This section gives an overview on the general architecture of the MBSE ontology, and provides statistics to get an impression of its complexity.

8.2.1 Constituent Ontologies

What is called MBSE Ontology more accurately consists of a number of constituent ontologies. The main ontology named *MBSE* imports ontologies describing different aspects of space system design, such as *Topological Design, Functional Design*, or *Verification*. These constituent ontologies are used for thematically partitioning the MBSE ontology, and for providing reusability of thematic concepts. Also, this approach enables independent configuration control by different stakeholders or owners of the MBSE Ontology constituents. This architecture, together with the ontologies' respective prefixes, is shown in Figure 8.2.

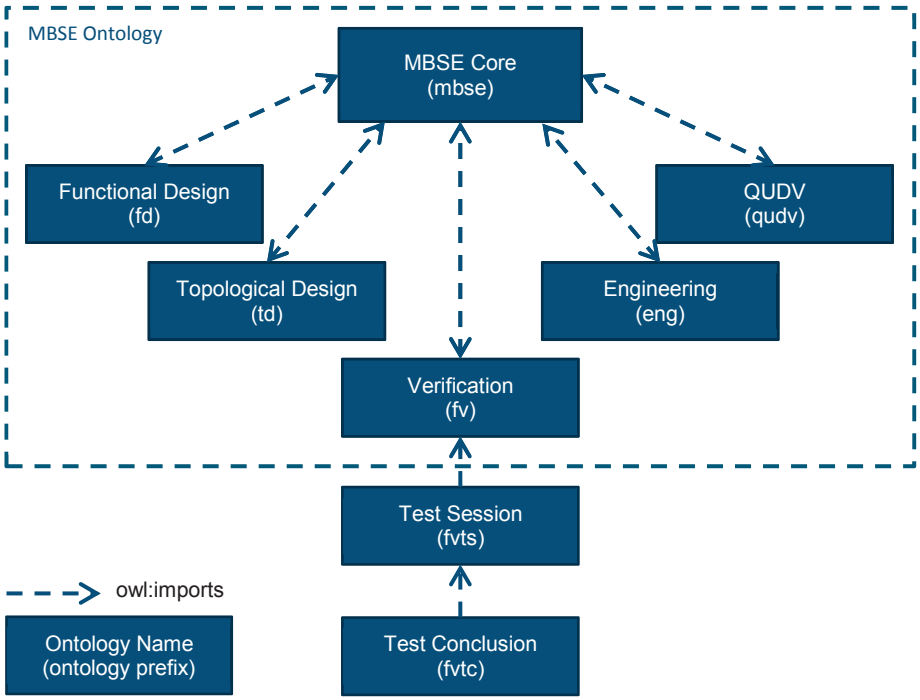


Figure 8.2: MBSE Ontology Constituents

The *MBSE Core* ontology imports all of the constituent ontologies in order to enable usage of the concepts defined in them. However, the constituent ontologies also utilize concepts defined in other constituents, so them importing the *MBSE Core* ontology is also required.

The *MBSE Core* ontology contains central concepts, such as the Product Structure, and concepts for the definition of properties. These properties are refined by concepts from the *QUDV* ontology, which contains information about physical quantities and units. The *Functional Design* ontology contains the concepts used to describe functional aspects of the system, while the *Topological Design* ontology can be used to describe interfaces between elements of the *Product Structure*. The *Verification* ontology describes generic verification-related concepts, while its sub-ontologies refine one of four means of verification in space engineering, the test. Due to this specificity, these are not considered a core part of the *MBSE* Ontology. The *Engineering* ontology contains concepts supporting a range of engineering activities, such as identification of critical elements, identification of single points of failure, and examination of physical effect interactions.

8.2.2 Ontology Metrics

In order to give a sense of size and complexity, the statistics in Table 8.2 are provided for the MBSE Ontology. These metrics encompass all concepts of the main ontology and of its imported constituents.

Table 8.2: MBSE Ontology Metrics

	mbse	eng	fd	qudv	td	ver	fvts	fvtc
Axiom count	647	563	55	34	143	111	146	128
Logical axiom count	349	378	27	17	72	54	58	93
Declaration axioms count	286	162	27	16	71	48	88	35
Class count	206	136	8	3	65	35	33	38
Object property count	42	26	10	5	5	7	1	0
Data property count	9	1	0	0	4	5	55	9
Individual count	50	65	9	9	0	10	0	0

8.2.3 Description Syntax

The MBSE Ontology and its sub-ontologies are documented in OWL 2's Manchester Syntax (W3C, 2012c) in this chapter and subsequent chapters. In contrast to OWL 2's standard functional syntax that relies on axiomatic specification, the Manchester Syntax is based upon a hierarchical representation of properties of the OWL 2 language's first order entities, resulting on an overall better readability for users.

8.3 CDM Concepts Improved from 10-23

This section contains a description of concepts that are already present in 10-23, but are improved considerably in the MBSE CDM regarding their extent, expressiveness, or alignment to the underlying engineering process.

8.3.1 Product Structure

The *Product Structure* is a central concept in 10-23, defining the building blocks that make up the system. In addition, the *Product Structure* gives these building blocks a lifecycle notion, determining if it is an element *as specified*, *as configured*, or *as built*.

The notion of a *Product Structure* that describes a system along its lifecycle started in 10-23 with the concepts of *ElementDefinition*, *ElementUsage*, *ElementOccurrence*, and *ElementRealization* and evolved over numerous related and follow-on projects. In this thesis, the SCDMP was used to augment the *Product Structure* further, ensuring an exact fit to occurring engineering processes, introducing important constraints, and assigning a genuine lifecycle notion to its concepts.

8.3.1.1 Derivation of Product Tree Main Concepts

The *Product Structure* starts at the *Product Tree* that forms a breakdown of the system into subsystems, components, and other constituents. The *Product Tree* describes elements *as specified*, providing a description about how often a component occurs in the system, the components' configuration numbers, and several other aspects.

In order to produce the concepts to represent the *Product Tree* in the MBSE CDM, the SCDMP is used on an actual *Product Tree* document, deriving the model from the elementary facts occurring within.

Step 1.1 of the SCDMP involves selecting an artefact to be modeled, which is the *Product Tree* in the case at hand. Step 1.2 is about scoping the information to be modeled, which is the example data of the *Product Tree of the MagSat* spacecraft that has been presented earlier in Table 3.1. This will serve as source data for CDM derivation. The next step 1.3 involves transforming the information contained in the sample into elementary facts. The following facts are selected for this purpose:

The MagSat has the sub element Electrical Power System.
The MagSat has the sub element Data Handling System.
The Electrical Power System has the sub element Battery.
The Battery has the Config Item No 1200.
The Electrical Power System has the Config Item No 1000.
The Battery has the abbreviation BAT.
The On-Board Computer has the abbreviation OBC.
The Battery occurs 1 time in the Electrical Power System.
For the Pipework 1 set occurs per Cold Gas Propulsion System.

The last fact is not yet a genuine elementary fact as it can be split further. It requires additional transformation, yielding:

The Pipework occurs once per Cold Gas Propulsion System
The Pipework occurs as set.

Also, the information that the Pipework is inside the *Cold Gas Propulsion System* is already present. Consequently, this part is excluded from the fact, yielding:

The Pipework occurs 1 time.

The same is done for the other similar fact:

The Battery occurs 1 time.

Furthermore, in discussion between modeling expert and discipline expert during the modeling activity, the fact is reformulated. The idea behind is that the fact is phrased in a way where the fact behaves like a property of the first building block, yielding:

The Pipework has a multiplicity of 1.
The Battery has a multiplicity of 1.

For step 1.4 of the SCDMP, the facts have to be sorted into similar facts and have to be split up into constant and variable parts.

The MagSat	has the sub element	Electr. Power Syst.
The MagSat	has the sub element	Data Handling Syst.
The Electr. Power Syst.	has the sub element	Battery.
variable part	*constant part*	*variable part*
Element	has sub element	Element

For the second type of fact, the following can be said:

The Battery	has the Config Item No	1200.
The Electr. Power Syst.	has the Config Item No	1000.
variable part	*constant part*	*variable part*
Element	has the Config Item No	String

For the third type of fact, the following can be said:

The Battery	has the abbreviation	BAT.
The On-Board Computer	has the abbreviation	OBC.
variable part	*constant part*	*variable Part*
Element	has the abbreviation	String

Next fact:

The Battery	has a multiplicity of	1.
The Pipework	has a multiplicity of	1.
variable part	*constant part*	*variable part*
Element	has a multiplicity of	Integer

This concludes the first main activity of the SCDMP that deals with gathering and scoping information about the artefact. At this point, the core facts that make up the *MagSat's Product Structure* are identified. As a next step, these facts have to be formalized in a CDM. In the case at hand, the CDM will be based on SCDML syntax, which is generated by following the SCDMP. As such, the SCDMP's second main activity deals with putting this information into an SCDML-based model.

8.3.1.2 Modeling of Product Tree Core Constructs

For this purpose, the SCDMP prescribes creating an *SPackage* for the artefact in question (step 2.1). In this case an *SPackage* called *productstructure* is created. In that package, the modeling of the classes making up the artefact will be pursued. As an initial step, the fact type saying that an Element has a configuration item number is modeled. Initially, the Element is modeled as an *SClass* (step 2.2), and, as each *Element* seems to have a name, a name *SAttribute* is introduced. Furthermore, the fact that the *Element* has a *configurationItemNumber* based on a *String* is modeled as an *SAttribute* (step 2.3). This yields a model as outlined in Figure 8.3.

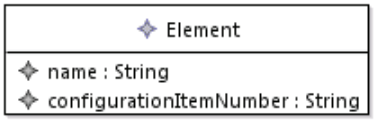


Figure 8.3: Product Tree CDM-part state 1

As a next step, the remaining facts are added to the model. The fact that *Elements* may exhibit a *multiplicity* that is counted in full numbers is translated to an *SAttribute* with type *Integer*. The fact that an element may occur as a *set* is translated to a Boolean *SAttribute*. Also, the fact that an element may be *abbreviated* is added, yielding the following model (Figure 8.4):

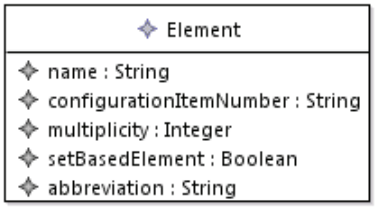


Figure 8.4: Product Tree CDM-part state 2

Consequently, the fact that *Elements* can have a number of other *Elements* of the *Product Tree* in a hierarchy is put into the model, resulting in an *SReference* from an *Element* to another *Element* (step 2.4), as both concepts playing the elementary fact are of type *Element (*Figure 8.5*)*.

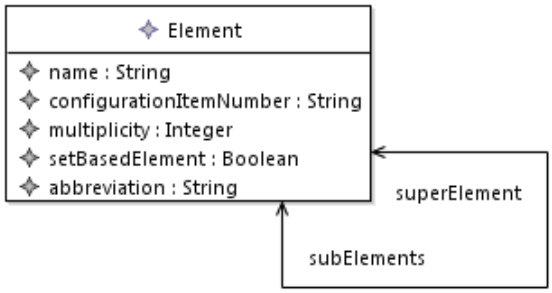


Figure 8.5: Product Tree CDM-part state 3

As of now, all collected facts have been put into the CDM. As a final step of this activity, in accordance with the subject matter expert, *SReferences* are examined for *opposites* (step 2.5). In order to do this, the expert in the domain to be modeled has to be consulted regarding the views usually taken in the domain. This means that the subject matter expert is asked whether any of the identified facts with its identified reading direction also make sense when formulated in the other direction. In this case, this would mean that

Electrical Power Syst.	has sub element	Battery.
Battery	has super element	Electr. Power Syst.

For the fact in question, the subject matter expert says that this makes sense and is actually a vital reading direction in the *Product Tree*, so the fact representing this *SReference* is modeled and made an *opposite* to the existing *subElements SReference*.

Cases may arise where the modeling expert is able to identify facts that the subject matter expert may not have recognized. One of these facts is that all of the *Elements* mentioned above are part of the artefact that is focused by the model, i.e. the *Product Tree* itself, e.g.:

The MagSat Product Tree	has the element	MagSat.
The MagSat Product Tree	has the element	Electr. Power Syst.
The MagSat Product Tree	has the element	Battery.

These facts are checked with the subject matter expert and put into the CDM. As it is evident from the fact in question that the *ProductTree* can have more than one *Ele-*

ment, one of the naming conventions prescribed in 7.3.2.6 is implemented by adjusting the *elements SReference* to being formulated in plural (Figure 8.6).

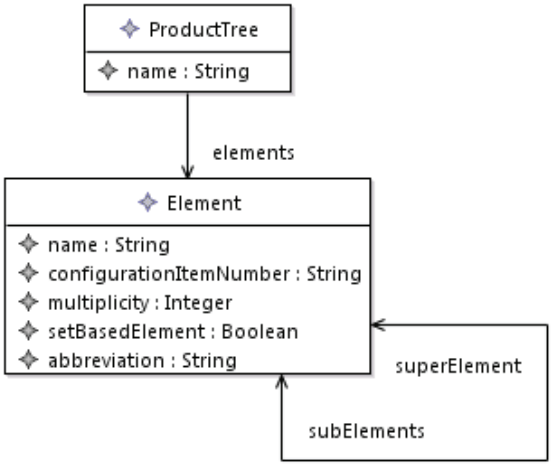


Figure 8.6: Product Tree CDM-part state 4

8.3.1.3 Derivation of Product Tree Constraints

The next major activity of the SCDMP deals with introducing constraints to the modelled structures. Initially, the cardinalities of features are to be derived using a number of questions (step 3.1). In this case, the *SAttributes* of the *Element* are treated first. The first question to be asked is *How many configurationItemNumbers can an Element have at the same time?* The *Product Tree's* documentation (see Table 3.1) has one line per concrete instance, mapping to one field in per instance and *Config Item No* column, indicating that there can be at most one *configurationItemNumber* per *Element*.

The next question revolves around whether the *Element* has to have a *configurationItemNumber*. The Product Tree's documentation at hand has a value for each *Element*. Consequently, the assumption can be made that each *Element* is always required to have a *configurationItemNumber*, in respect to the previously scoped set of information available about the artefact. This leads to a *FeatureCardinalityConstraint* with *upperBound 1* and *lowerBound 1*, resulting in the fact that each *Element* always has to have exactly one *configurationItemNumber*.

The same process is followed for the *multiplicity*, yielding the same result. For the *setBasedElement* Boolean *SAttribute*, the same applies. From the *Product Tree's* docu-

mentation it can be ascertained that an abbreviation exists in many cases, but that the field may also be empty. However, there is never more than one abbreviation, yielding a *FeatureCardinality Constraint* on the abbreviation *SAttribute* with *upperBound 1* and *lowerBound 0*, making it optional. Also, the *name SAttribute* of both the *Element* and the *Product Tree* is constrained to be required and maximum *1*.

The same process is pursued for the elements *SReference*. The *ProductTree* can obviously contain more than one *Element*, as there are multiple *Elements*, i.e. rows appearing in its documentation. Directly, it cannot be determined if it would be okay for the *ProductTree* to contain no elements at all, so the subject matter expert is consulted. As he states that this might be possible, a *FeatureCardinalityConstraint* with *upperBound -1* (representing infinity) and *lowerBound 0* is modeled. For the *Element*, the question arises whether the same element can have multiple *subElements*. This can be confirmed based on information in the *ProductTree*. Also, the hierarchies end at some level, making it necessary to leave the *subElements* reference empty, leading to a *FeatureCardinalityConstraint* of *0..-1*. For the *superElement*, it is determined that there can be at most one *superElement* and that this is optional, as the *MagSat* itself does not have a *superElement*. This leads to a constraint of *0..1 (*Figure 8.7*)*.

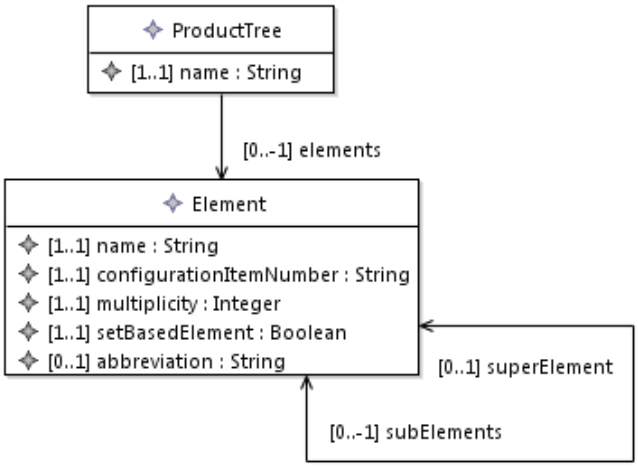


Figure 8.7: Product Tree CDM-part state 5

For each *SStructuralFeature* with a *FeatureCardinalityConstraint* that has an *upperBound greater than 1*, *uniqueness* can be determined (step 3.2). *Uniqueness* means that the same fact may not occur twice at the same time. For instance, this would mean that the *Electrical Power System* contains the exact same *Battery* twice. This, and the

uniqueness of *ProductTree contains Element* is checked with the subject matter expert. Based on the answer, both *SReferences* are made unique.

Elementary facts that involve the same *SClasses*, i.e. *SReferences* with the same *SClass* as owner and type, are applicable to *RingConstraints*. For this purpose step 3.3 using the process described in 7.3.3.3 is pursued, resulting in a populated ring constraint table (Table 8.3).

Table 8.3: Derivation of Ring Constraints

Property	Question	Answer	Additional Properties
Reflexivity	Is it necessary that, if the EPS has as subElement the BAT, then it also has as subElement the EPS?	no	Symmetry Transitivity
Symmetry	Is it necessary that if the EPS has as subElement the BAT, then the BAT also has as subElement the EPS?	no	Irreflexivity Intransitivity Reflexivity Transitivity
Transitivity	Is it necessary that, if the MagSat has as subElement the EPS and the EPS has as subElement the BAT, then the MagSat also has as subElement the BAT?	no	Irreflexivity Asymmetry Acyclicity Reflexivity Symmetry
Acyclicity	If the MagSat has as subElement the EPS and the EPS has as subElement the BAT, is it forbidden that the BAT has as subElement the MagSat?	yes	Transitivity Intransitivity
Irreflexivity	Is it forbidden that the EPS has as subElement the EPS?	yes (but not required due to acyclicity)	Symmetry Transitivity
Asymmetry	Is it forbidden that if the EPS has as subElement the BAT, then the BAT also has as subElement the EPS?	yes (but not required due to acyclicity)	Transitivity Intransitivity
Intransitivity	Is it forbidden that, if the MagSat has as subElement the EPS and the EPS has as subElement the BAT, then the MagSat also has as subElement the BAT?	yes	Asymmetry Acyclicity Symmetry

Consequently, the *subElements SReference* becomes applicable to a *Ring Constraint* that is *acyclic* and *intransitive*. On the one hand, the *Elements* of the *ProductTree* should not form any cycles, as there is supposed to be a hierarchy with a defined *Element* at the top, usually the system itself, and a given set of leaf *Elements* at the very bottom. Also, the consideration of *Elements* is intransitive, as the *subElements*

relation in this case is always considered directly, with an interest in only the next lower level elements.

For optional *SStructuralFeatures*, *SetComparisonConstraints* are applicable. The only spot this currently applies to is at the *ElementDefinition* between *abbreviation* and *subElements*. Intuition might state that there is very likely no connection between these two, but in order to confirm this, the prescribed process for deriving *SetComparisonConstraints* is pursued (step 3.4). Initially, hypothetical facts are derived.

> Fact X: The Electrical Power Syst. has the abbreviation EPS.
> Fact Y: The Electrical Power Syst. has as subElement the Battery.

By populating the schema given by Table 7.4, sample facts for deriving set comparison constraints are created (Table 8.4):

Table 8.4: Fact Derivation for Set Comparison Constraints

Fact Population	Fact X	Fact Y
X ∧ Y	Electrical Power System has the abbreviation EPS	Electrical Power System has as subElement the Battery
X ∧ ¬Y	Electrical Power System has the abbreviation EPS	-
¬X ∧ Y	-	Electrical Power System has as subElement the Battery
¬X ∧ ¬Y	-	-

With these combinations, the following questions, based on Table 8.4, are asked:

- Is fact combination 1 a valid combination, i.e. can the *Electrical Power System* have an *abbreviation* and a *subElement* at the same time?
- Is fact combination 2 valid, i.e. can the *Electrical Power System* have an *abbreviation* while having no *subElements*?
- Is fact combination 3 valid, i.e. can the *Electrical Power System* have *subElements* while having no *abbreviation*?
- Is fact combination 4 valid, i.e. can the *Electrical Power System* have no *abbreviation* and no *subElements*?

The answers to those questions are all yes, as provided by the subject matter expert, which implies that there is no constraint whatsoever between the two *SStructuralFeatures*. This makes the only valid fact combination the one in column A of Table 8.5.

Table 8.5: Evaluation Table for Set Comparison Constraints

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
X ∧ Y	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
X ∧ ¬Y	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
¬X ∧ Y	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
¬X ∧ ¬Y	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
Answer	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	-	-	-	-	-	-
Consequence	No constraint	Inclusive-or	X superset and Y subset	Mandatory up (card >0)	X subset and Y superset	Mandatory down (card >0)	Equality	Mandatory both (card >0)	Exclusion constraint	Exclusive -Or Constraint	Not applicable					

The next type of constraint to be modeled is the *ClassMultiplicityConstraint* (step 3.5). This kind of constraint implies that there is a maximum amount of objects for a specific class that can exist. The *Product Tree's* documentation suggests that it can hold multiple *Elements*, so there is no such constraint on the *Element SClass*. However, there might be a scenario where there is only one *ProductTree* in the whole project, which is checked with the subject matter expert. As he negates this, no constraint is modeled.

The same can be done with *SStructuralFeatures* with the *FeatureMultiplicity Constraint* (step 3.6). In this case, only a specific amount of objects would be able to exhibit a value for the respective feature. From the documentation, it can be derived that the *abbreviation* of an *Element* is unique, meaning that it is not possible that two or more *Element* have an identical *abbreviation*. This is modeled in the CDM for the *abbreviation SAttribute* (not shown in the diagram).

The last constraint to be modeled is the *ValueComparisonConstraint* (step 3.7) that can exist between comparable *SAttributes*. As there is only current one *SAttribute* that is numerically comparable, this does not apply, so no constraint is derived.

After modeling the mentioned constraints, the CDM looks as detailed in Figure 8.8:

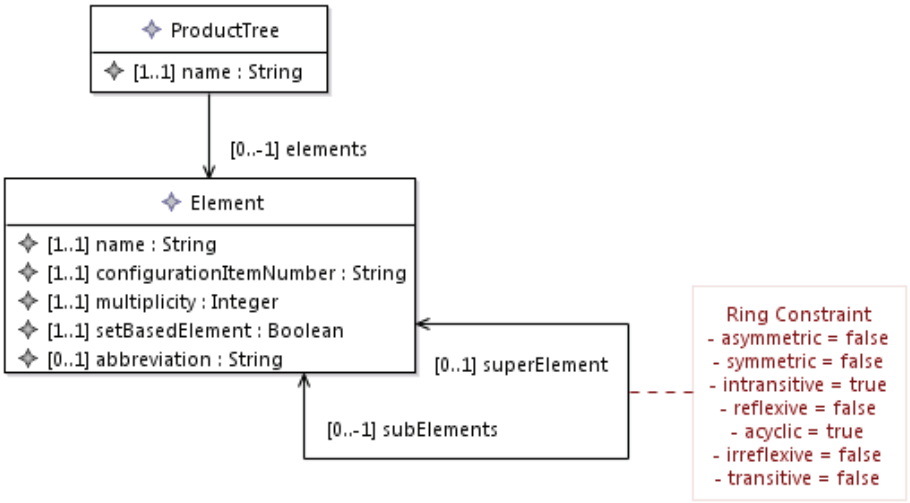


Figure 8.8: Product Tree CDM-part state 6

8.3.1.4 Product Tree Core Construct Refinement

For the next major activity of the SCDMP, class *supertypes* and *subtypes* have to be identified (step 4.1). In this case, this falls more to the modeling expert, as no other information is available that would allow identifying any class taxonomies. As the modeling expert knows that there will be other trees like the *ProductTree*, and similar concepts as the one described by the *Element SClass*, these concepts are abstracted. Furthermore, the naming is adjusted where the *Element* becomes an *ElementDefinition*. However, as there will never be an instance of the *SystemTree*, only the *ProductTree* and other concrete trees, this *SClass* is made abstract, which is shown as an italic styling of *SClass* names in Figure 8.9.

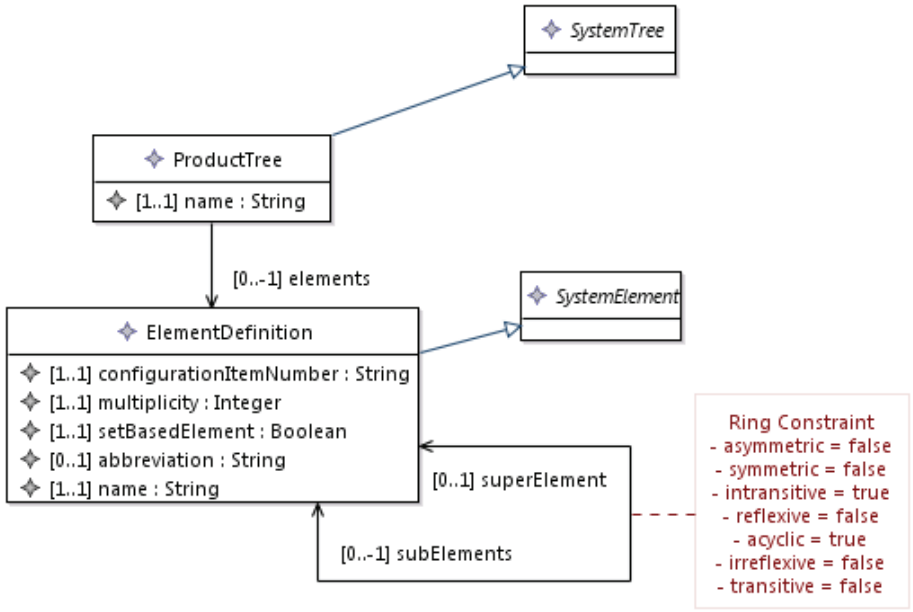


Figure 8.9: Product Tree CDM-part state 7

Defining *semanticTypes* (step 4.2) will be revisited later on.

It is also known from the discipline expert that all *SystemTrees* will have a name, and that all *SystemElements* will have a name. As these two properties are identical, they are abstracted to a more generic concept, forming the *NamedElement*.

Step 4.3 prescribes that each *SReference* is to be examined whether it forms an aggregation, i.e. if it forms a hierarchical structure. For the *subElements* reference, this is already evident from the *ProductTree's* documentation, as the elements there are clearly hierarchical. To determine which kind of *aggregation* is necessary, the subject matter expert is asked if a given *ElementDefinition* can still exist if its *superElement* no longer does. In this case, this is confirmed, so the *subElements* reference will have an *aggregation* of kind *shared* (empty diamond in Figure 8.10).

For the *ProductTree*, that contains numerous *ElementDefinitions*, the same question is asked. There, the subject matter expert answers that it would not make sense to have any *ElementDefinitions* still remaining in the model, once a spacecraft's *ProductTree* itself was deleted. This results in a *composite* aggregation, as represented by the filled diamond in Figure 8.10.

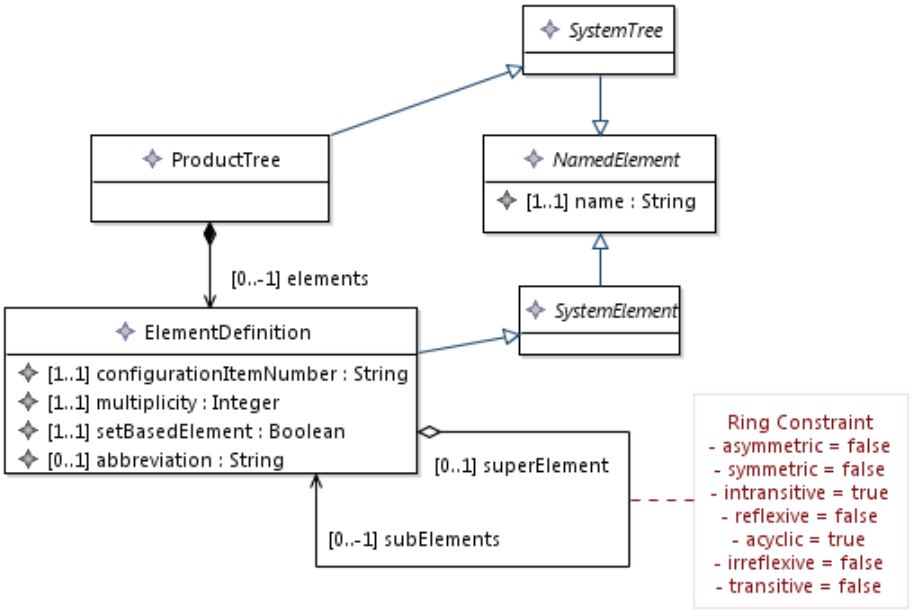


Figure 8.10: Product Tree CDM-part state 8

As the *Product Structure* is a pattern on its own, the step to identify and make explicit patterns is omitted (step 4.4).

In this early stage of the modeling effort, no rules can be modeled (steps 5.1 and 5.2), as most of the elements required for these rules are missing. This subject will be revisited in a later stage of modeling the *Product Structure*.

8.3.1.5 Product Tree Temporal Criteria

For introducing lifecycle aspects to the *Product Tree*, the SCDMP provides a process for deriving *Temporal Criteria* (step 6.1). For *Temporal Criteria* on this level, documentation for the ESA system engineering process (ESA, 2009a) is consulted, as that process defines the overall lifecycle of a project in this context. Consequently, the main milestones in this process are introduced as *Temporal Criteria*.

As a next step (6.2), *ForbiddenConstraints* should be allocated to the modeled concepts, stating that specific concepts or features are explicitly excluded at defined points in the project's or rather model's lifecycle. As the *Product Tree* stands in the very beginning of the engineering effort, it is not excluded by a *ForbiddenConstraint*.

The same is true for all of its *SStructuralFeatures*. However these kinds of constraints are revisited later on.

8.3.1.6 Product Tree CDM Validation

The last main activity is validation. For this purpose, facts from the sample population, in this case the *Product Tree's* documentation, are to be put into the model, or rather an instantiation of the CDM, as outlined in Figure 8.11.

	CI Number	Abbreviation	Multiplicity	isSet
Element Definition MagSat	0000		1	false
Element Definition Electrical Power System	1000	EPS	1	false
Element Definition Battery	1200	BAT	1	false
Element Definition Power Control and Distribution Unit	1100	PCDU	1	false
Element Definition Attitude and Orbit Control System	4000	AOCS	1	false
Element Definition Cold Gas Propulsion System	4100	CGPS	2	false
Element Definition Pipework	4170		1	true

Figure 8.11: Instantiated Product Tree validation data

As the information of the documentation can in fact be represented in the CDM, the first step of its validation is successful. Now, an additional step is taken where deliberately inconsistent facts are modeled. In this case, the *Battery* is modeled to also contain the *EPS*, forming a *cycle* and being *symmetric*, violating the defined *RingConstraint* on the *subElements SReference*. As this gets flagged through the constraint, the modeled part of the CDM is validated.

8.3.1.7 Product Structure Decomposition Levels

Decomposition levels of the *Product Structure* are defined in the MBSE Ontology. 10-23 prescribes several levels of hierarchical decomposition of the system (ESA, 2011a), where each level has specific semantics. These levels are all defined as a subclass of the *SystemLevelElement* and encompass:

```
mbse:SystemLevelElement
    mbse:Segment
    mbse:System
    mbse:Subsystem
    mbse:SubsystemSet
    mbse:Assembly
    mbse:Equipment
    mbse:Component
    mbse:Part
    mbse:Module
```

In the above extract from the MBSE Ontology, the system levels are shown in their appearing hierarchy in 10-23 (ESA, 2011a). The *Segment* is the most top-level concept in which the actual *System* is contained. The *System* may have a number of *Subsystems*, which may be decomposed into *SubsystemSets*. In a *Subsystem*, the elements that form an "integrated set of parts and components" that "accomplishes a specific function" (ESA, 2012b) are defined as *Equipments*.

Equipments may be made up of several *Components*, which form a "set of materials, assembled according to defined and controlled processes, which cannot be disassembled without destroying its capability and which performs a simple function that can be evaluated against expected performance requirements." (ESA, 2012b)

For the level of *Component*, a distinction has to be made. A *Component* is regarded as a *Part*, if it exhibits no electronic or electrical characteristics (ESA, 2012b), i.e. if it is a purely mechanical *Component*. If a *SystemElement* is a *Component*, but also a piece of software, it becomes a *Module* (ESA, 2011a).

This additional information is also supplied in the ontology:

```
Class: mbse:Subsystem
    EquivalentTo:
        mbse:isDirectlyContainedByElement some mbse:System
    SubClassOf:
        mbse:SystemLevelElement

Class: mbse:Part
    EquivalentTo:
        mbse:Component
         and (not (td:hasConnector some td:Connector))
    SubClassOf:
        mbse:Component

Class: mbse:Module
    EquivalentTo:
        mbse:Component
         and mbse:SoftwareElement
    SubClassOf:
        mbse:Component
```

8.3.1.8 Remaining Concepts of the Product Structure

With a similar approach, the remaining concepts of the *Product Structure* can be modeled, putting the *ProductTree* and the *ElementDefinition* into the larger context. This is shown in Figure 8.12.

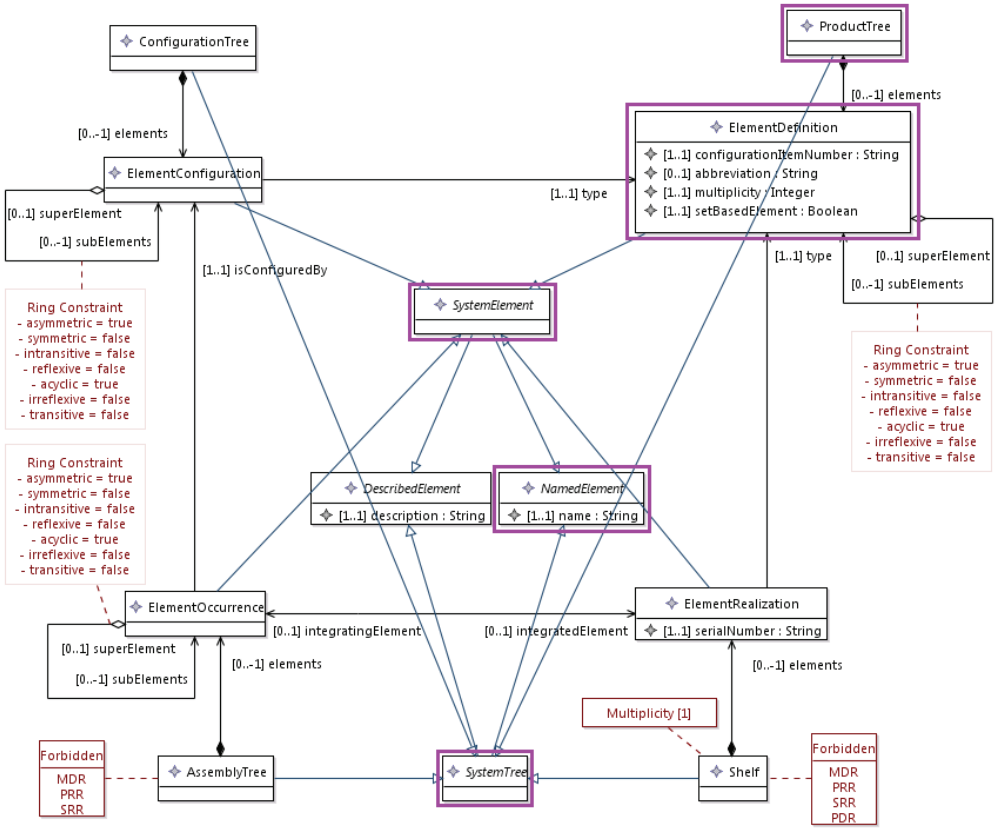


Figure 8.12: Product Structure CDM

The model of the *Product Structure* is now complete, with the existing concepts (emphasized in purple) being complemented by the following new concepts:

The *ConfigurationTree* contains *ElementConfigurations* which are representing *SystemElements* as configured. These elements can be used to exactly configure the system in terms of relation of elements to each other, and to define system variants. *ElementConfigurations* are typed by an *ElementDefinition*. For each configuration of the system, one *ConfigurationTree* may exist.

The *AssemblyTree* consists of *ElementOccurrences* and is used to represent each concrete instantiation of the system that will be built. For instance, if the *MagSat* mission consists of a constellation of three identical satellites, there would be one *ConfigurationTree*, and three *AssemblyTrees*, one for each concrete spacecraft. In the beginning of a project's lifecycle while the mission concept and first design are elabo-

rated, the *AssemblyTree* is not required. Consequently, it is deactivated via a *ForbiddenClassConstraint* that applies for the early milestones of the system's design cycle.

The *AssemblyTree* serves as an anchor for *ElementRealizations* stored in the *Shelf*. *ElementRealizations* are concrete, tangible components *as built*, and thus exhibit a *serial number* for identification. As such, the *Shelf* represents all elements *as built* currently available, ready to be tested or integrated. Each *ElementOccurrence* may integrate one *ElementRealization*, meaning that, for example, the produced *Star Tracker Sensor Head* with serial number *msc_str01_a841* is *integrated* into position *Star Tracker Sensor Head X*, defined by the according *ElementConfiguration*. Originating from the underlying engineering process, a *ClassMultiplicityConstraint* has been defined for the *Shelf*, stating that there can only be one *Shelf*, in contrast to the other *SystemTrees*. Also, the *Shelf* is restricted with a *ForbiddenClassConstraint* for early design phases.

8.3.2 Physical Properties

In 10-23, the concept of *EngineeringDataCategories* was introduced as a runtime-loadable library containing various properties These properties and their categories are exchangeable during runtime, as the motivation of this construct is to provide the possibility for project-specific data structure adaption without requiring to deliver a new application based on an updated CDM, supporting the practice of tailoring. The properties contained inside these categories may be common enumeration or string properties, but also properties based on a physical quantity.

For the MBSE CDM, these categories and properties were moved to the ontology-side of the CDM. In 10-23, for being changeable during runtime, properties were required to be described through emulating a class/instance structure, offering description of both type and object on M0 level. In the MBSE Ontology, both the dynamic and the instantiation aspect can be realized simpler, by offering a genuine class structure that is changeable during runtime. These classes are then instantiated via *Individuals*.

In the MBSE Ontology, categories are realized as subclasses of the *EngineeringPropertyElement*.

```
Class: mbse:EngineeringPropertyElement
    SubClassOf:
        mbse:SystemEngineeringThing

Class: mbse:MissionDesignElement
    SubClassOf:
        mbse:EngineeringPropertyElement
```

```
Class: mbse:MassBudgetElement
    SubClassOf:
        mbse:MissionDesignElement,
        mbse:hasMass exactly 1 mbse:ElementMass
    DisjointWith:
        mbse:SoftwareElement

Class: mbse:ThermalPropertyElement
    SubClassOf:
        mbse:EngineeringPropertyElement

Class: mbse:OperationalTemperatureRangeElement
    SubClassOf:
        mbse:ThermalPropertyElement,
        mbse:hasMaxNonOperatingTemperature exactly 1
          mbse:TemperatureValueProperty,
        mbse:hasMaxOperatingTemperature exactly 1
          mbse:TemperatureValueProperty,
        mbse:hasMinNonOperatingTemperature exactly 1
          mbse:TemperatureValueProperty,
        mbse:hasMinOperatingTemperature exactly 1
          mbse:TemperatureValueProperty
```

The logical consistency of categories is ensured using *disjoints*. For instance, the *MassBudgetElement* is disjoint with any *SoftwareElements*. Also, categories meant for specific system levels (e.g. *System* or *Subsystem*) are made *disjoint* with the *System-Levels* to which they may not apply.

Physical properties are defined using the concept of *ValueProperty* that also utilizes SysML's QUDV model, similar to 10-23. This is realized with the object property *isBasedOnQuantity* that requires a QUDV *QuantityKind*, such as mass, length, or electrical potential difference, etc.

```
Class: mbse:ValuePropertyThing
    SubClassOf:
        mbse:SystemEngineeringThing

Class: mbse:RealQuantityProperty
    SubClassOf:
        mbse:ValuePropertyThing,
        qudv:isBasedOnQuantity exactly 1 qudv:QuantityKind,
        mbse:hasValue exactly 1 xsd:double
```

For defining these *properties*, the *RealQuantity* class is refined with, for instance, *ThermalValueProperty*, forming the set of thermal-relevant properties, such as *TemperatureValueProperties*.

```
Class: mbse:ThermalValueProperty
    SubClassOf:
        mbse:RealQuantityProperty
```

```
Class: mbse:TemperatureValueProperty
    SubClassOf:
        mbse:ThermalValueProperty,
        qudv:isBasedOnQuantity value qudv:temperatureQK
```

In addition, the property concept was enhanced to support uncertainties engineering. On the one hand, this includes support for specifying the maturity status of a property, determining if it is based on an assumption, or if its value is actually confirmed by some kind of analysis. On the other hand, this includes support for margin-based properties that are commonly used when the property's value has significant uncertainty attached to it. These concepts are realized with the concepts of *KeyParameter* and *MarginBasedProperty*, respectively.

```
Class: mbse:KeyParameter
    SubClassOf:
        mbse:ValuePropertyThing,
        mbse:hasMaturityStatus exactly 1
          mbse:KeyParameterMaturityStatus

Class: mbse:KeyParameterMaturityStatus
    SubClassOf:
        mbse:ValuePropertyThing,
        { mbse:CustomerAssumption, mbse:CustomerConfirmed,
          mbse:TeamAssumption, mbse:TeamConfirmed, mbse:Unknown}

Class: mbse:MarginBasedProperty
    SubClassOf:
        mbse:RealQuantityProperty,
        mbse:hasMargin exactly 1 xsd:double,
        mbse:hasNominalValue exactly 1 xsd:double
```

There may be properties that are both *KeyParameters* and *MarginBasedProperties*, as is the case with many *MassProperties*:

```
Class: mbse:MassValueProperty
    SubClassOf:
        mbse:KeyParameter,
        mbse:MarginBasedProperty,
        mbse:RealQuantityProperty

Class: mbse:ElementMass
    SubClassOf:
        mbse:MassValueProperty

Class: mbse:SubsystemMass
    SubClassOf:
        mbse:MassValueProperty

Class: mbse:SystemMass
    SubClassOf:
        mbse:MassValueProperty
```

8.3.3 Topological Design

Modeling a system's topology, including concepts such as its electrical interfaces or mechanical interfaces, is realized using concepts from the *topologicaldesign* package, such as the *Electrical Architecture* (Figure 8.13).

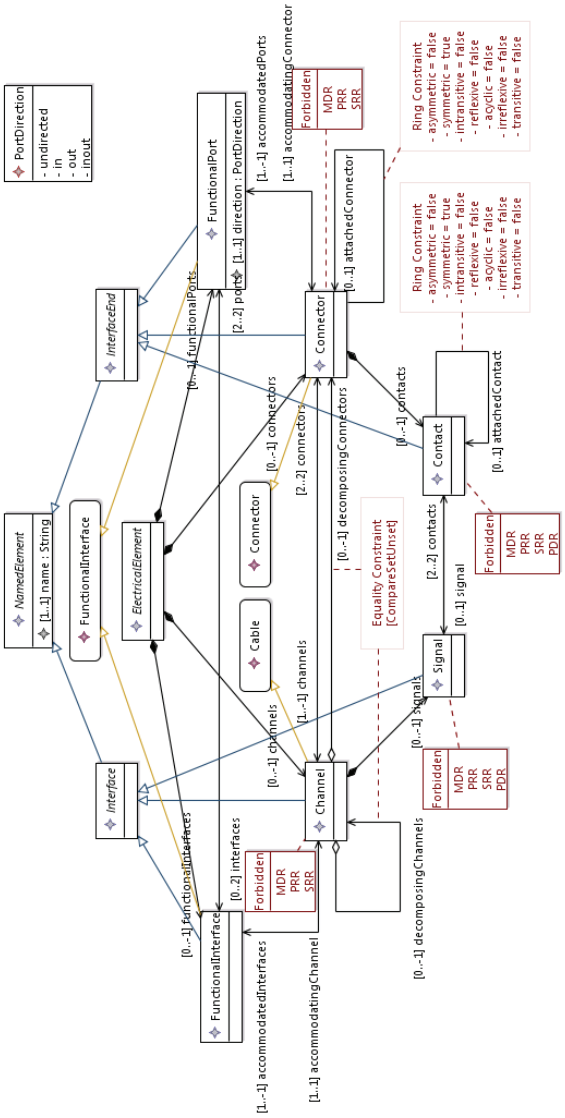


Figure 8.13: Part of MBSE CDM topologicaldesign package

This package is spread across both the MBSE OO-Model and the MBSE Ontology. The conceptual model in this regard was derived using diagrammatic artefact documentation in terms of the *Functional Electrical Interface Diagram*, and tabular artefact documentation with the *Harness Interface Control Document*. Employing the SCDMP yielded the model given in Figure 8.13.

The level of *FunctionalInterface* and *FunctionalPort* forms a functional view of what interfaces in general are used aboard the spacecraft, and about their purpose. The *Channel* and *Connector* level maps these functional concepts to a concrete hardware implementation via the *accommodatingChannel* and *accommodatingConnector SReferences*. *Channels* and *Connectors* are in turn broken down into concrete *Contacts* and *Signals*.

These main concepts are refined via the yellow *semanticType* relation, indicating a connection to the MBSE Ontology. While the central concepts such as *Channel* and *Connector* are defined on the object-oriented side of the CDM, the *semanticType* relation states that they are completed by concepts on the ontology side of the CDM. On the ontological side, the concrete physical properties of these concepts are defined, being tailorable to a specific project due to their ontological nature.

FunctionalInterface and *FunctionalPort* are both typed via the ontological *FunctionalInterface* class, offering a variety of subtypes for these interfaces. A selection of possible types is outlined below.

```
Class: td:FunctionalInterface
    SubClassOf:
        td:ElectricalArchitectureThing

Class: td:AnalogueInterface
    SubClassOf:
        td:FunctionalInterface

Class: td:AN1Interface
    SubClassOf:
        td:AnalogueInterface

Class: td:AN2Interface
    SubClassOf:
        td:AnalogueInterface

Class: td:HPCInterface
    SubClassOf:
        td:FunctionalInterface

Class: td:HPC1Interface
    SubClassOf:
        td:HPCInterface
```

```
Class: td:HPC2Interface
    SubClassOf:
        td:HPCInterface

Class: td:PowerInterface
    SubClassOf:
        td:FunctionalInterface

Class: td:LC1Interface
    SubClassOf:
        td:PowerInterface

Class: td:LC3Interface
    SubClassOf:
        td:PowerInterface

Class: td:LVPInterface
    SubClassOf:
        td:PowerInterface
```

On the level below, the concrete physical properties of *Cables* are detailed:

```
Class: td:Cable
    SubClassOf:
        td:ElectricalArchitectureThing,
        td:diameter exactly 1 mbse#Diameter,
        td:specificResistance exactly 1 mbse#SpecificResistance,
        td:specificWeight exactly 1 mbse#SpecificWeight,
        td:gauge exactly 1 xsd:integer,
        td:noOfCores exactly 1 xsd:integer,
        td:noOfShields exactly 1 xsd:integer,
        td:twisted exactly 1 xsd:Boolean

Class: td:TSPCable
    SubClassOf:
        td:Cable,
        td:gauge value 24,
        td:noOfCores value 2,
        td:noOfShields value 1,
        td:twisted value true
```

8.3.4 Functional Design

The concepts for *Functional Design* are based on an object-oriented description in 10-23, but were moved to the MBSE Ontology in this work. The reason for this repartition is to enable additional use cases that utilize the functional description of a system, which cannot be directly realized on the object-oriented side of the CDM. On the ontological side, however, the definition of functions can be utilized for automatically identifying issues in the system's design, such as single points of failure, as is demonstrated in the next chapter in section 9.4.2.2.

The definition of *Functional Design* is directly taken from the original concept in 10-23, which puts an emphasis on how this is realized in the space domain. The original model complemented with ontological aspects, enabling reasoning functionality. More specifically, the *functionaldesign* ontology contains concepts to describe functions, their type of redundancy, relations between functions, and an allocation of functions to *SystemElements*. The most essential concepts are:

```
Class: fd:Function
    SubClassOf:
        fd:FunctionalDesignThing,
        fd:containsFunction min 0 fd:Function,
        fd:containsInterfaceEnd min 0 fd:FunctionInterfaceEnd,
        fd:hasFunctionRedundancy max 1 fd:FunctionRedundancyType

ObjectProperty: fd:isPerformedBy
    InverseOf:
        fd:performsFunction

Class: fd:FunctionRedundancyType
    EquivalentTo:
        {fd:coldredundant , fd:hotredundant , fd:nonredundant}
    SubClassOf:
        fd:FunctionalDesignThing
```

8.3.5 Verification

The *verification* package deals with ways to verify the various requirements on the system. Figure 8.14 describes the main concepts of the CDM used for verification.

The activity of *Verification* can be performed using different approaches, including verification by *Test*, *Analysis*, *Inspection*, or *Review*. Out of these concepts, the first forms the most relevant concept and is detailed further. For this purpose, a *TestTask* is defined to *verify* a given set of *Requirements*, and implemented using a *TestSpecification*. This *TestSpecification* defines the general properties of the test to be performed, such as its *TestType*, the *itemUnderTest*, the configuration of the tested system, and used *TestFacilities* and *TestEnvironments*. For detailing a *TestSpecification*, a *TestProcedure* using a number of *Steps* is defined. This procedure is then executed in one or several *TestSessions*, which produce data represented by the *DataSession*, which is subsequently evaluated in the *TestEvaluation* for determining if the *TestSession* was successful or not.

The concept of Verification is complemented by further concepts on the ontological side of the CDM, enabling exploitation of available design and test data, which is detailed in sections 8.5.1.5 to 0.

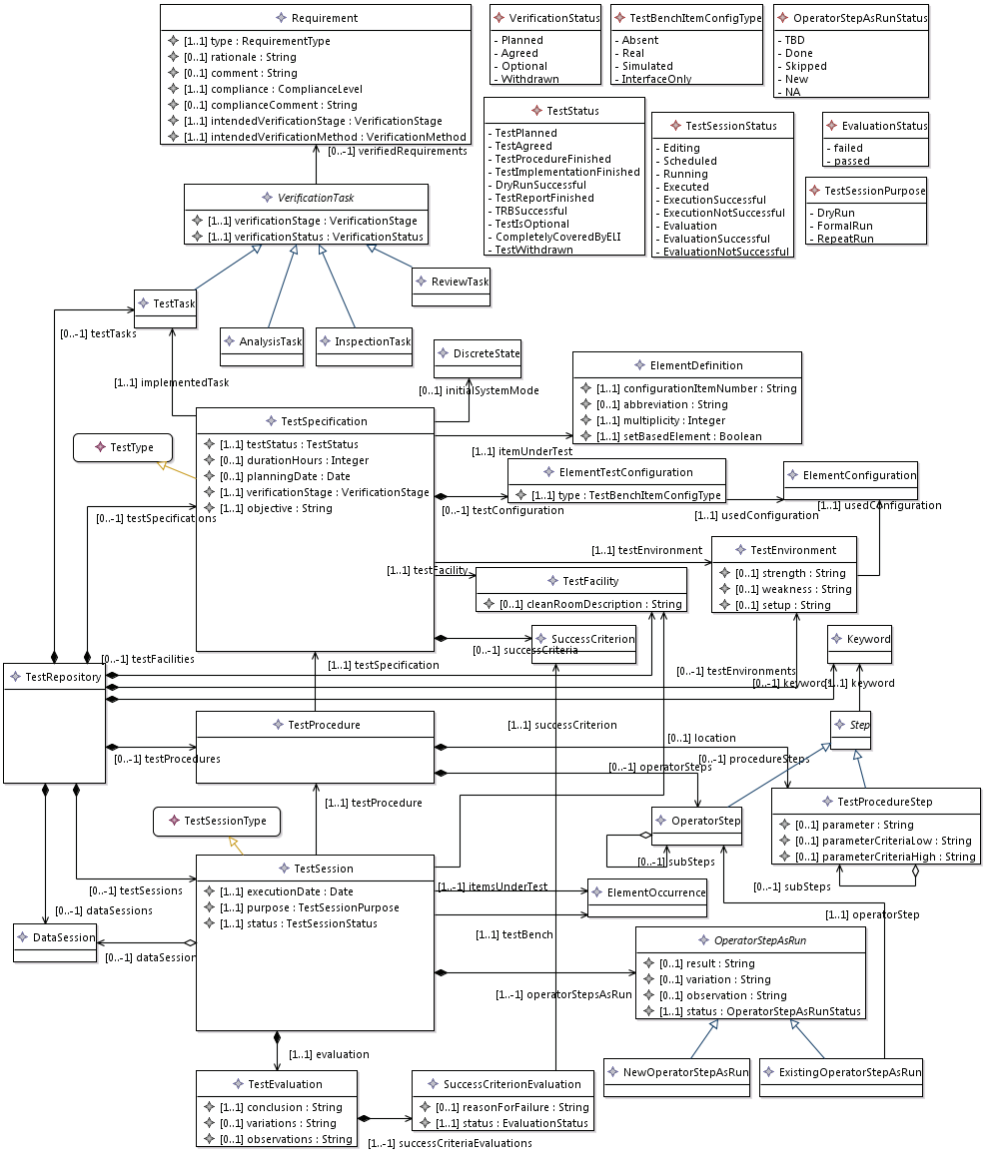


Figure 8.14: MBSE CDM verification package

8.4 CDM Concepts Confirmed from 10-23

There are also concepts in 10-23 that were confirmed as being correct and sufficient by use of the SCDMP. While these concepts were confirmed overall, they are improved at some points with *Constraints* or with *Temporal Criteria*, both of which are concepts not originally scoped by the 10-23 CDM.

8.4.1 Specification

Specification is done using *Requirements*, of which the semantics are specified in the *requirements* package. This package is directly adopted from 10-23 for the MBSE CDM. This was done after an actual *Spacecraft Design Specification* containing requirements was treated with the SCDMP, confirming that the structures defined in 10-23 were correct and sufficient. A slight addition was made by including two new constraints. This leads to the core structure for the requirements-related part of the CDM given in Figure 8.15.

This version of the requirements meta-model is evaluated to ensure that it is able to contain a set of representative sample data. For this purpose, the *MagSat Spacecraft Design Specification* outlined earlier in Figure 3.10 is modeled using a limited set of representative data taken from the document.

The repositories or requirement groups such as *Satellite Requirements or Launcher and launch environment compatibility* are represented by the *RequirementRepository SClass*. The requirements themselves, such as *In-orbit lifetime* and *Reliability* are represented by the *Requirement SClass*, including *name*, *description*, *identifier*, and the other attributes. *RequirementTypes*, as well as other properties of the requirements part of the MBSE CDM, are derived from the applicable process documentation (ESA, 2009b). The *ExlusiveOrConstraint* between *requirements* and *subRepositories* states that a *RequirementRepository* may either contain other repositories, or requirements. It can never contain both, but at least one of these two. The full set of data from Figure 3.10 can be modeled accordingly.

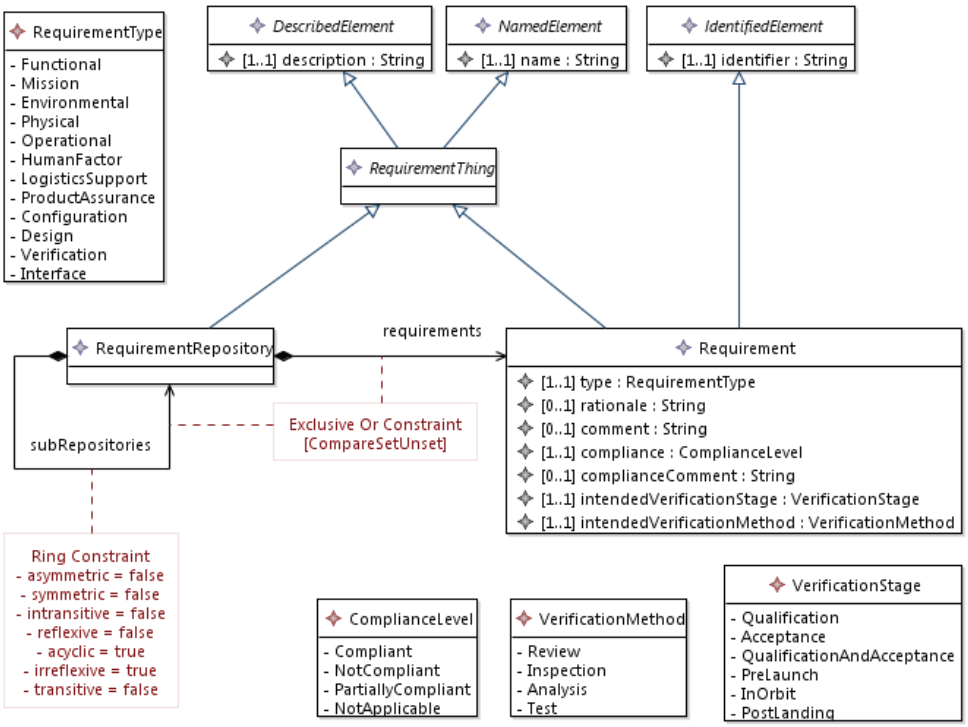


Figure 8.15: MBSE CDM requirements package

In addition to being compatible to the space engineering process driven by ESA, the data structures in the requirements part of the MBSE CDM are also compatible to other standards such as ReqIF (OMG, 2016), being realized over dedicated import and export interfaces.

8.4.2 Operational Design

8.4.2.1 Discrete Model

In order to describe the behavior of *SystemElements*, the *DiscreteModel* from 10-23 is also used in the MBSE CDM. It has been slightly extended by *ExclusionConstraints*, stating that a *sourceState* of a transition cannot be its *targetState* and that a *DiscreteState* cannot constraint itself, but must constrain other states. Additionally, *ForbiddenClassConstraints* were added for specifying behavior of any kind is not yet to be

defined in the mission definition stage of a project, and that highly detailed *DiscreteModels* with transitions are not required in the mission elaboration phase. *DiscreteStates*, however, may exist. This model is summarized in Figure 8.16

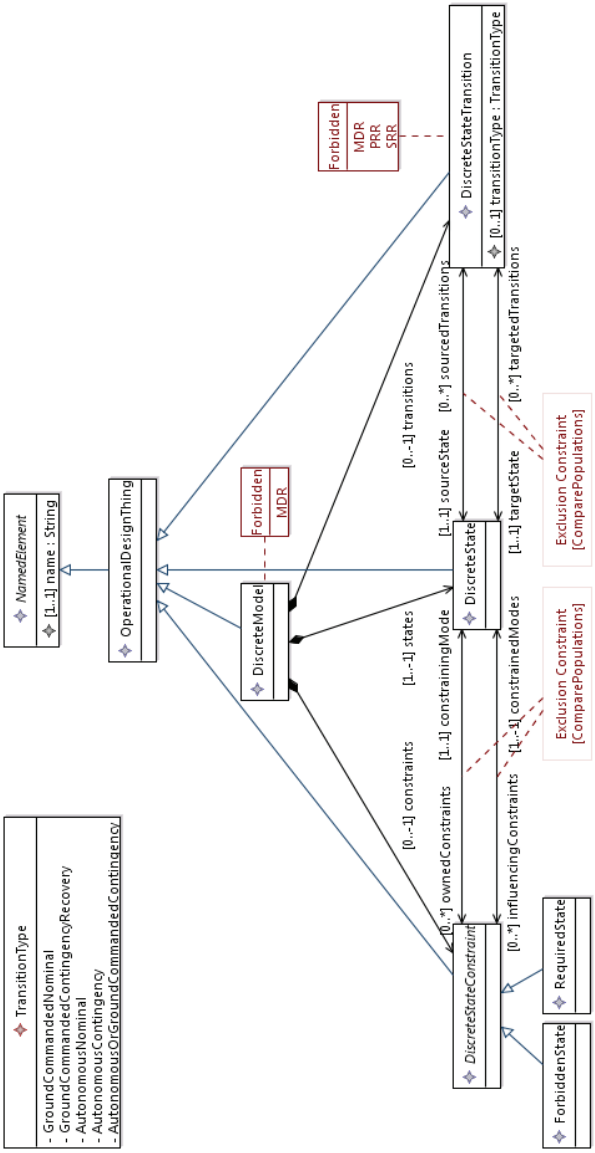


Figure 8.16: MBSE CDM discretemodel package

8.4.2.2 Operational Procedures

The concept of *OperationalProcedures* from 10-23 was also confirmed by applying the SCDMP, resulting in the following model without adaptions (Figure 8.17):

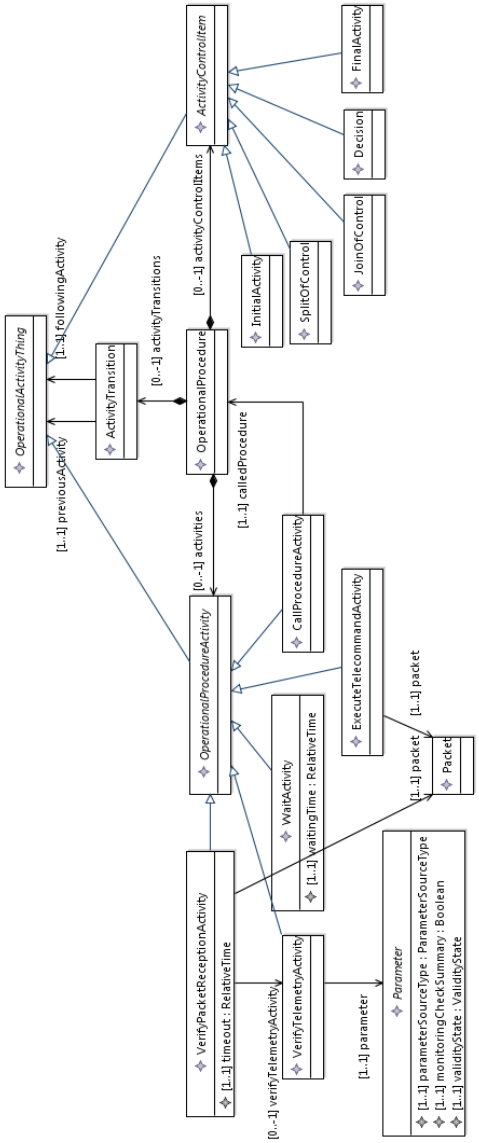


Figure 8.17: MBSE CDM operationalactivity package

8.4.2.3 Monitoring and Control

The *Monitoring and Control* model, defining *Packets*, *Parameters*, and other services used for space system command and control, is defined in the *monitoringandcontrol* package. The model remains unchanged from its original design (ESA, 2013a; Eisenmann & Cazenave, 2014) and is not documented further at this point.

8.5 CDM Concepts Newly Introduced

Several concepts have been introduced to the MBSE CDM that are out of scope of the original 10-23 specification. Primarily, these new concepts reside in the area of knowledge management and exploitation, enabling the automated execution of numerous engineering activities. A strong notion in this context is that the knowledge specified in the MBSE CDM required for performing these activities forms a kind of engineering knowledge base that can be applied to a system under design, enabling automated activity execution. Furthermore, it is intended that the knowledge base grows steadily with each project, as more and more operational engineering knowledge becomes formalized in the CDM.

An important differentiation to make at this point is that, again, the modeling process behind coming to these data structures involves at least two individuals. On the one hand, there is the subject matter expert who is the expert on a specific part of the system and the underlying engineering process. On the other hand there is the modeling expert who, in accordance with the discipline expert, is able to produce a model-based representation of the given process.

8.5.1 Engineering Activity Support and Knowledge Base

In order to cater to the requirements related to supporting engineering activities by providing support on SM level (see 5.1.3), the concept of the *Knowledge Base* is introduced, residing on the ontology-side of the MBSE CDM. It contains information for enabling the automated execution of several engineering activities in the context of space system design. The concrete nature of these activities, their motivation, and inherent challenges, are detailed in Chapter 9. This chapter gives an outlook on the conceptual structures defined that enable the execution of these activities in conjunction with a reasoner. For a more in-depth understanding, it is highly recommended to read Chapter 9 beforehand, and to come back to this section once the application has become familiar.

8.5.1.1 Domain Allocation

For getting an overview of discipline involvement in each defined *System Element*, a taxonomy of *DisciplineElements* is defined. Each *DisciplineElement* comes with numerous conditions that, should they apply, denotes some involvement of an engineering discipline, such as *Thermal Engineering*, in a specific system component.

For example, an element involving *Requirements Engineering* as a discipline is simply defined as an element that has a *Requirement*

```
Class: mbse:RequirementsEngineeringElement
    EquivalentTo:
        req:RequirementElement
    SubClassOf:
        mbse DisciplineElement
```

A *ThermalEngineeringElement* is defined by elements that have a strong relevance in *Thermal Engineering*, as is the case for *Thermistors* and *Heaters*. Additionally, each element that has the defined thermal properties associated with it is scoped by *Thermal Engineering*:

```
Class: mbse:ThermalEngineeringElement
    EquivalentTo:
        mbse:Heater,
        mbse:ThermalPropertyElement,
        mbse:Thermistor
    SubClassOf:
        mbse:DisciplineElement
```

For the case of the *Harness*, an element of interest to the discipline of *Harness Engineering*, a sub-discipline of electrical engineering, each element that has a *FunctionalInterface*, or a *Connector* is scoped:

```
Class: mbse:ElectricalEngineeringElement
    SubClassOf:
        mbse:DisciplineElement

Class: mbse HarnessEngineeringElement
    EquivalentTo:
        mbse:Harness,
        td:hasConnector some td:Connector,
        td:hasFunctionalPort some td:FunctionalPort
    SubClassOf:
        mbse:ElectricalEngineeringElement
```

8.5.1.2 Critical Elements

Concepts for supporting the activity of determining *CriticalElements* in the system have been added to the MBSE Ontology. These concepts are derived from the sample MagSat project, which followed the critical item control process as prescribed by (ESA, 2008e). As such, the provided model is also compatible with this approach. Based on this existing process, knowledge for deriving different categories of *CriticalElements* is provided, being C*ontaminationElements*, *LifeLimitedElements*, M*agneticCleanlinessElements*, *TechnologyCriticalElements*, and *SafetyCriticalElements*. These types of *CriticalElements,* along with their definitions, are derived from the documentation of the *Critical Items List* maintained in the underlying MagSat project.

TechnologyCriticalElements, if not explicitly stated so, are elements of which their design is not yet flight qualified, implying a *Technology Readiness Level* (TRL) (NASA, 2007) lower than or equal to *7*.

```
Class: eng:TechnologyCriticalElement
    SubClassOf:
        eng:CriticalElement

Class: eng:DesignNotQualifiedElement
    EquivalentTo:
        mbse:technologyReadinessLevel some xsd:integer[<= 7]
    SubClassOf:
        eng:TechnologyCriticalElement
```

The class of *SafetyCriticalElements* contains a variety of different kinds of elements. What they have in common is that in the event of failure, personnel are likely to be injured, necessitating a number of mitigation steps. For instance any component of type *Battery* is always a *SafetyCriticalElement,* as its failure will have a significant impact during mission, and may result in injury to personnel during testing.

```
Class: eng:SafetyCriticalBattery
    EquivalentTo:
        mbse:Battery
    SubClassOf:
        eng:SafetyCriticalElement,
        eng:hasFailureEffect value eng:InjuryToPersonnel,
        eng:hasFailureEffect value eng:LossOfSpacecraft,
        eng:hasFailureEffect value eng:ReleaseOfToxicMaterial,
        eng:hasFailureEffect value eng:RuptureOfCells,
        eng:hasRiskReductionMeasure value eng:CurrentLimitDevice,
        eng:hasRiskReductionMeasure value eng:DesignQualification,
        eng:hasRiskReductionMeasure value eng:DoubleSealingBarrier,
        eng:severityLevel value eng:Catastrophic_1S
```

The same is true for any component of type *PressureTank*, where a number of risk reduction measures have to be taken into consideration, such as a *leak before burst*

design, the process of using *dye to detect flaws*, and the rule to *limit pressurization and depressurization cycles* to a pre-defined, safe amount.

```
Class: eng:SafetyCriticalPressureTank
    EquivalentTo:
        mbse:PressureTank
    SubClassOf:
        eng:SafetyCriticalElement,
        eng:hasFailureEffect value eng:InjuryToPersonnel,
        eng:hasFailureEffect value eng:LossOfSpacecraft,
        eng:hasRiskReductionMeasure value eng:BurstTest,
        eng:hasRiskReductionMeasure value eng:DyePenetrantFlawDetection,
        eng:hasRiskReductionMeasure value eng:LeakBeforeBurstDesign,
        eng:hasRiskReductionMeasure value eng:LimitNumberOfCycles,
        eng:hasRiskReductionMeasure value eng:ProofPressureTest,
        eng:hasRiskReductionMeasure value eng:UltrasonicFlawDetection,
        eng:severityLevel value eng:Catastrophic_1S
```

Another class of *CriticalElements* is given by *LifeLimitedElements*, which denote components that have a limited lifespan. The *PressureTank* also falls into this category and is modeled using the following expressions:

```
Class: eng:LifeLimitedPressureTank
    EquivalentTo:
        mbse:PressureTank
    SubClassOf:
        eng:LifeLimitedElement,
        eng:hasFailureEffect value eng:FillVentCyclesLeadToMaterialWear,
        eng:hasRiskReductionMeasure value eng:IncludeDesignMargins,
        eng:hasRiskReductionMeasure value eng:LimitNumberOfCycles,
        eng:severityLevel value eng:Catastrophic_1S
```

Another class of component that has a limited lifespan is any *Battery*, of which failure effects, risk reduction measures, and failure severity level are defined using the following statements:

```
Class: eng:LifeLimitedBattery
    EquivalentTo:
        mbse:Battery
    SubClassOf:
        eng:LifeLimitedElement,
        eng:hasFailureEffect value
          eng:BatteryCapacityDegradationOverTime,
        eng:hasRiskReductionMeasure value eng:IncludeDesignMargins,
        eng:hasRiskReductionMeasure value eng:ObserveStorageConditions,
        eng:hasRiskReductionMeasure value
          eng:ReduceUsageOfFlightModelDuringTest,
        eng:severityLevel eng:value Major_3
```

Another class is defined by the *MagneticCleanlinessElement*. This criticality is not given per se to any element that belongs to its basic type, such as *Battery* or *Pressure-Tank*, but is based on whether the instance of this type is situated within certain conditions. For instance, the *MagneticCriticalBattery* is not always critical:

```
Class: eng:MagneticCriticalBattery
    EquivalentTo:
        mbse:Battery
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some mbse:MagneticInstrument)))
    SubClassOf:
        eng:MagneticCleanlinessElement,
        eng:hasFailureEffect value
          eng:OwnMagneticFieldMayCausePerfDegradOfMagnInstruments,
        eng:hasRiskReductionMeasure value eng:CompensationInDataProcessing
```

The same is true for the spacecraft, which also only exhibits magnetically critical properties if it contains at least one *MagneticInstrument*, resulting in risk reduction measures such prescribing the use of demagnetized tools, to keep these tools separate from magnetized tools, and magnetically clear working conditions:

```
Class: eng:MagneticCriticalSpacecraft
    EquivalentTo:
        mbse:System
         and (mbse:containsElement some mbse:MagneticInstrument)
    SubClassOf:
        eng:MagneticCleanlinessElement,
        eng:hasFailureEffect value
          eng:UseOfMagneticToolsMayMagnetizeMaterials,
        eng:hasRiskReductionMeasure value eng:ClearWorkingInstructions,
        eng:hasRiskReductionMeasure value eng:KeepDemagnetizedToolsSeparate,
        eng:hasRiskReductionMeasure value eng:UseOfDemagnetizedTools
```

8.5.1.3 Single Points of Failure

Aspects of the system's design that do not occur with some form of redundancy form single points of failure. In the MBSE CDM, the concepts are provided for deriving single points of failure of a system, based on a description of its functional breakdown. This means that *Functions* are allocated to *Element Configurations*, and *Functions* contain a description of their internal redundancy. A Function that is only realized by one kind of *ElementConfiguration*, which only occurs once aboard the spacecraft, becomes a *SingleFailure Function*. *ElementConfigurations* that perform such functions become *Single FailureElements*.

```
Class: eng:SinglePointOfFailure
    SubClassOf:
        eng:EngineeringActivityThing

Class: eng:SingleFailureFunction
    EquivalentTo:
        fd:NonredundantDefinedFunction
         and (fd:isPerformedBy max 1 mbse:ElementConfiguration)
    SubClassOf:
        eng:SinglePointOfFailure

Class: eng:SingleFailureElement
    EquivalentTo:
        mbse:ElementConfiguration
         and (fd:performsFunction some eng:SingleFailureFunction)
    SubClassOf:
        eng:SinglePointOfFailure
```

8.5.1.4 Physical Interactions

Components aboard a spacecraft interact with other components. While at some points this is desired, there are also a number of undesired interactions. For instance, a *Reaction Wheel* produces vibration that puts disturbance upon the *Accelerometers* on board. A *Battery* emits a local electromagnetic field that impacts the accuracy of *Magnetic Instruments*. The plume of propellant exiting a *Thruster* may impact the field of view of O*ptical Instruments*, etc.

The concepts outlined in this section are not intended to replace detailed discipline-specific analyses of the specific effects. The concepts should be used to identify areas in the system's design, where a problem has the potential to occur. If an actual problem exists, as well as how extensive the interaction of effects actually is, should then be determined by a separate, detailed, discipline-specific analysis. The purpose of the generic consideration of physical interactions is to scope required analyses on system level, and to trigger them over the given system engineering process.

The following physical effects that frequently occur aboard spacecraft and have to be considered in its design are included in the MBSE Ontology:

- Electromagnetic Compatibility (ESA, 2012c)
- Outgassing (ESA, 2011b)
- Propellant Plume (ESA, 2008c)
- Thermal (ESA, 2008a)
- Vibration (ESA, 2008b)
- Micro-Meteorites and Orbital Debris (ESA, 2012d)

In order to evaluate these effects, the concepts of *PhysicalInfluenceElement* and *PhysicalInfluencedElement* are introduced. The first describes the general definition of

these physical properties and contains the classes of *Emitting Element* and *SusceptibleElement*. These are in turn split up into *Thermal EmitingElement* and *ThermalSusceptibleElement*, *MagneticEmittingElement* and *MagneticSusceptibleElement*, etc. For this purpose, the following taxonomy is introduced:

```
eng:PhysicalInteractionThing
    eng:PhysicalInfluenceElement
        eng:EmittingElement
            eng:MagneticEmittingElement
            eng:OutgassingEmittingElement
            eng:PlumeEmittingElement
            eng:ThermalEmittingElement
            eng:VibrationEmittingElement
        eng:SusceptibleElement
            eng:MagneticSusceptibleElement
            eng:MMODSusceptibleElement
            eng:OutgassingSusceptibleElement
            eng:PlumeSusceptibleElement
            eng:ThermalSusceptibleElement
            eng:VibrationSusceptibleElement
```

The components are allocated to these classes by subclassing at least one of them, illustrated with the following examples:

```
Class: mbse:Battery
    SubClassOf:
        eng:MagneticEmittingElement
         and eng:MagneticSusceptibleElement
         and eng:ThermalEmittingElement
         and eng:ThermalSusceptibleElement

Class: mbse:OnBoardComputer
    SubClassOf:
        eng:MagneticEmittingElement
         and eng:MagneticSusceptibleElement
         and eng:OutgassingSusceptibleElement
         and eng:ThermalEmittingElement
         and eng:ThermalSusceptibleElement
         and eng:VibrationSusceptibleElement

Class: mbse:SBandAntenna
    SubClassOf:
        eng:MagneticEmittingElement
         and eng:MagneticSusceptibleElement
         and eng:VibrationSusceptibleElement
```

In order to evaluate the concrete influences aboard one spacecraft, the class of *PhysicalInfluencedElement* and its subclasses come into play.

```
Class: eng:PhysicalInfluencedElement
    SubClassOf:
        PhysicalInteractionThing

Class: eng:MagneticInfluencedElement
    EquivalentTo:
        eng:MagneticSusceptibleElement
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some
               eng:MagneticEmittingElement)))
    SubClassOf:
        eng:PhysicalInfluencedElement

Class: eng:OutgassingInfluencedElement
    EquivalentTo:
        eng:OutgassingSusceptibleElement
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some
               eng:OutgassingEmittingElement)))
    SubClassOf:
        eng:PhysicalInfluencedElement

Class: eng:PlumeInfluencedElement
    EquivalentTo:
        PlumeSusceptibleElement
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some
              eng:PlumeEmittingElement)))
    SubClassOf:
        eng:PhysicalInfluencedElement

Class: eng:ThermalInfluencedElement
    EquivalentTo:
        eng:ThermalSusceptibleElement
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some
               eng:ThermalEmittingElement)))
    SubClassOf:
        eng:PhysicalInfluencedElement

Class: eng:VibrationInfluencedElement
    EquivalentTo:
        eng:VibrationSusceptibleElement
         and (mbse:isContainedByElement some
            (mbse:System
             and (mbse:containsElement some
               eng:VibrationEmittingElement)))
    SubClassOf:
        eng:PhysicalInfluencedElement
```

8.5.1.5 Test Identification

Determining which tests have to be performed on which system component is dependent on the component's nature. Defining which tests have to be performed on which components is also supported by an automated process, using the concept of *TestRequiringElement*. This class has numerous subclasses that describe the different kinds of tests that may be necessary on a given component. For example, a component requiring an *Integrated System Test* (IST) is, in any case, the OBC, and every component that has been defined as being an *Equipment* from the system decomposition perspective:

```
Class: fv:ISTRequiringElement
    EquivalentTo:
        mbse:CentralSoftware or mbse:Equipment
    SubClassOf:
        fv:TestRequiringElement,
        fv:requiresTest value fv:IST
```

There is also the *Electric Integration Test* (ELI) that is required by any component that has a *Connector*:

```
Class: fv:ELIRequiringElement
    EquivalentTo:
        mbse:ElementDefinition
         and (td:hasConnector some td:Connector)
    SubClassOf:
        fv:TestRequiringElement,
        fv:requiresTest value fv:ELI
```

For other elements, test necessity is quite straightforward, e.g. each *On-Board Control Procedure* (OBCP) requires an *OBCP IST*:

```
Class: fv:OBCPISTRequiringElement
    EquivalentTo:
        op:OnBoardControlProcedure
    SubClassOf:
        fv:TestRequiringElement
```

8.5.1.6 Test Session Identification

Each test also has a specific configuration of components that are required to be present for the test to be executed. As the integration state of a satellite changes often, it is not always evident which tests may be performed at a given point in time.

While the general class of *TestCapableSystem* can be defined in the MBSE CDM, this class has to be refined on a per-project basis, as different satellite designs require different configurations for a specific test to be performed. This is shown in further detail in 9.5.2.2. Consequently, only the general class is defined at this point:

```
Class: ver:TestCapableSystem
    SubClassOf:
        Ver:VerificationThing
```

8.5.1.7 Test Conclusion

For determining if a conducted test was a success or failure, the data generated by the test is evaluated. In order to enable automated evaluation, these logs can be represented ontologically, and test success or failure can be determined using a reasoner. For this purpose, the following concepts taxonomy is provided:

```
fvts:AsRunLogThing
  fvts:AsRunLog
    fvts:AsRunLogElement
      fvts:EventReport
      fvts:Procedure
      fvts:ProcedureElement
        fvts:ArmPacket
        fvts:CheckInputArguments
        fvts:Cmd
          fvts:SatCmd
          fvts:SCOECmd
        fvts:Comment
        fvts:ControlElement
          fvts:CallSub
          fvts:ProcedureStart
          fvts:ProcedureEnd
          fvts:StepStart
          fvts:StepEnd
        fvts:EndVerify
          fvts:EndVerifyAnd
          fvts:EndVerifyOr
          fvts:EndVerifyPrint
        fvts:EndVerifyElement
        fvts:Exesub
        fvts:InitMessage
        fvts:OpRequest
        fvts:ReadPacket
        fvts:ReleasePacket
        fvts:Step
        fvts:WaitCycle
        fvts:WaitForMessage
```

EventReports can be classified into four kinds of *ThrownEvents*, where the severity is determined according to their *PUS type* and *PUS subtype* (ESA, 2003). In order to manage this data, the following conceptual structures are provided:

```
Class: fvtc:ThrownEvent
    SubClassOf:
        fvtc:TestConclusionThing

Class: fvtc:NormalEvent
    SubClassOf:
        fvtc:ThrownEvent,
        fvts:EventReport
         and (fvts:pusSubtype value 1)
         and (fvts:pusType value 5)

Class: fvtc:WarningEvent
    SubClassOf:
        fvtc:ThrownEvent,
        fvts:EventReport
         and (fvts:pusSubtype value 2)
         and (fvts:pusType value 5)

Class: fvtc:AnomalyEvent
    SubClassOf:
        fvtc:ThrownEvent,
        fvts:EventReport
         and ((fvts:pusSubtype value 3)
         and (fvts:pusType value 5))

Class: fvtc:CriticalEvent
    SubClassOf:
        fvtc:ThrownEvent,
        fvts:EventReport
         and ((fvts:pusSubtype value 4)
         and (fvts:pusType value 5))
```

These thrown events are further processed by classifying them into *ExpectedEvents* and *UnexpectedEvents*. However, as the conditions for belonging to one of these two classes are project-specific, only the general description can be given at this point:

```
Class: fvtc:ProcessedEvent
    SubClassOf:
        fvtc:TestConclusionThing

Class: fvtc:ExpectedEvent
    SubClassOf:
        fvtc:ProcessedEvent

Class: fvtc:UnexpectedEvent
    SubClassOf:
        fvtc:ProcessedEvent
```

As a significant part of the relevant data is system-specific, further detailing is performed in the application chapter in 9.5.2.3.

8.5.1.8 Engineering Heuristics

Another knowledge-based functionality is driven by the class of *Engineering HeuristicThings*. This concept provides information for identifying elements that fall into some conspicuity that is based on a pre-defined heuristic, derived from engineering experience. These heuristics all point to some area in the system design that is not yet sufficient, resulting in engineering work still to be done. While in the beginning of a project, these areas will be quite extensive, whereas towards the end these areas should be minimized or ideally fully eliminated. Applying these heuristics provides a continuously updated overview of not yet fully completed design elements that can be utilized within the system engineering process, highlighting elements that are of increased importance due to their lack of maturity.

One such heuristic is the *TenuousElement* that is the superclass of *SystemElements* that have some aspect in their design that is not yet sufficient. This can be due to a variety of reasons. For instance, the subclass of *AssumedParmeterElement* identifies all elements that still have parameters associated with them that are based on an assumption and are not yet confirmed by an analysis. The class of *MultiplePRElement* marks elements that have at least three *ProblemReports* associated with them, while the class of *MultipleRIDElement* does something similar for *ReviewItemDiscrepancies*. However, it comes with different thresholds, depending on whether it is a *CriticalElement*, or not.

```
Class: eng:TenuousElement
    SubClassOf:
        eng:EngineeringHeuristicThing

Class: eng:AssumedParameterElement
    EquivalentTo:
        mbse:ElementDefinition
         and (mbse:hasValueProperty some
          (mbse:KeyParameter and
           ((mbse:hasMaturityStatus value mbse:CustomerAssumption) or
             (mbse:hasMaturityStatus value mbse:TeamAssumption))))
    SubClassOf:
        eng:TenuousElement

Class: eng:MultiplePRElement
    EquivalentTo:
        mbse:ElementDefinition
         and (mbse:hasEngineeringAnnotation min 3 mbse:ProblemReport)
    SubClassOf:
        eng:TenuousElement
```

```
Class: eng:MultipleRIDElement
    EquivalentTo:
        mbse:ElementDefinition
         and eng:CriticalElement
         and (mbse:hasEngineeringAnnotation min 3
          mbse:ReviewItemDiscrepancy),
        mbse:ElementDefinition
         and (mbse:hasEngineeringAnnotation min 7
          mbse:ReviewItemDiscrepancy)
    SubClassOf:
        eng:TenuousElement
```

Another heuristic is given by the class of *TestingStressToBeMinimizedElement*, where the number of times an *ElementRealizion* is switched on during test is being tracked and compared to a threshold (in this case 70%) using a SWRL rule:

```
mbse:ElementRealization(?er) ^ ver:noOfTimesSwitchedOn(?er, ?nso) ^
ver:maxNoOfTimesSwitchedOn(?er, ?mnso) ^ swrlb:divide(?d, ?nso, ?mnso) ^
swrlb:greaterThan(?d, 0.7) -> eng:TestingStressToBeMinimizedElement(?er)
```

8.5.2 Rules

Rules are located on the object-oriented side of the MBSE CDM and utilize two of SCDML's concepts, *FunctionalRules* and *OCLStatements*.

8.5.2.1 OCL-Based Consistency Checks

The latter can be used for common consistency checking, implementing checks not covered by the built-in constraints. For example, in the *Product Structure*, it is important to check whether the type of an integrated *ElementRealization* is identical to the type of its configuring *ElementConfiguration*. For this purpose, the following statement is attached to the *ElementOccurrence*:

```
integratedElement.type=isConfiguredBy.type
```

OCL-based constraints are also necessary in the *discretemodel* package for ensuring correct ownership of transitions between states. Thus, for the *DiscreteStateTransition*, the following constraint is defined:

```
sourceState.oclContainer=oclContainer
```

8.5.2.2 Product Structure Functional Rules

For addressing the requirement of formalizing functional dependencies between model elements (see 5.1.1), several *FunctionalRules* are defined for the *ProductStructure*. These *FunctionalRules* have a large impact upon all *SystemElements* of the *ProductStructure*, as they declare how *SystemElements* from different *SystemTrees* relate to each other, and how they behave in respect to their *semanticTypes*.

For instance the following *FunctionalRule* is defined for the *ElementConfiguration*, describing the relation to its typing *ElementDefinition*.

```
ElementDefinition(?ed) ^ multiplicity(?ed, ?mult) -> scdml:haveInstances(?ec,
ElementConfiguration, ConfigurationTree::elements, ?mult) ^ type(?ec, ?ed)
```

This rule defines how *ElementConfigurations* are to be created. For each *ElementDefinition*, an instance in the variable *ec* is to be created, of type *ElementConfiguration*, in the containing feature elements, for a total of *multiplicity* times. In addition, the *type* feature of the *ec* is to be set to the *ed*.

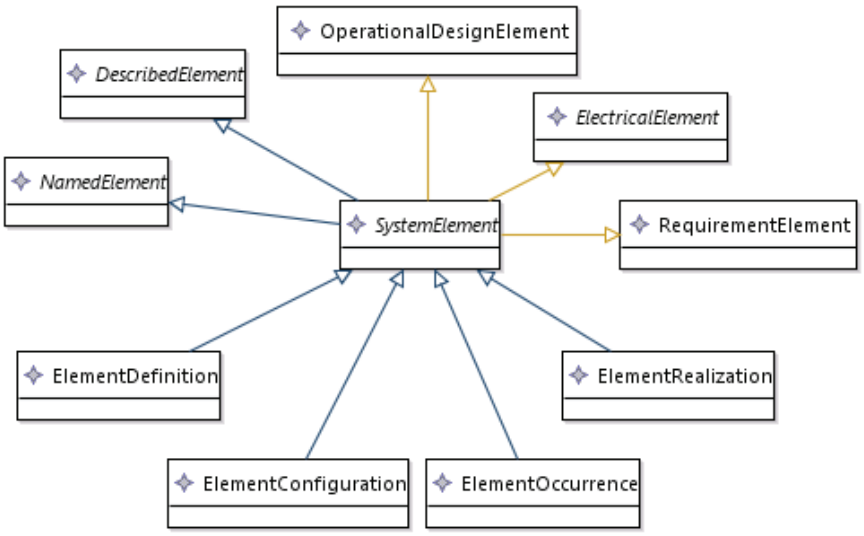


Figure 8.18: Selected semantic types for the SystemElement

Another aspect is the concept of *semanticTypes* in the *ProductStructure*. The *semanticType* reference in SCDML was introduced for enabling multiple instantiation for pre-defined types. Figure 8.18 describes in an exemplary manner three conceivable

instantiations of the *SystemElement*, such as *RequirementElement*, *OperationalDesignElement*, and *ElectricalElement*.

As the semantic types between the four *SystemElements* are strongly related, additional rules are introduced. For describing the relation between *semanticTypes* of the *ElementConfiguration* and the *ElementDefinition*, the following rule is introduced:

```
ElementConfiguration(?ec) ^ type(?ec, ?ed) ^ semanticTypes(?ed, ?st)
-> scdml:reproduceSemanticTypes(?st, ?ed, ?ec)
```

In this rule, an *ElementConfiguration* is examined regarding the *semanticTypes* of its typing *ElementDefinition*. If *semanticTypes* exist there, they are to be reproduced for the *ElementConfiguration* according to the defined rules behind the *scdml:reproduceSemanticTypes* function, essentially meaning a copy that also copies applicable references. In other words, any of the *semanticTypes* of an *ElementDefinition* will also be instantiated for all its related *ElementConfigurations*.

These rules are evaluated when any of the *SClasses* or *SStructuralFeatures* used in the rule are changed. In the first example, this would mean that the rule is enforced in cases where an *ElementDefinition* is newly created, or where its *multiplicity* is changed. In the second case, the rule would be newly evaluated once the *type* of an *ElementConfiguration* changes, or once the *semanticTypes* of its typing *ElementDefinition* change.

8.6 Conclusion on MBSE CDM Modeling

This chapter used the SCDMP to derive a CDM for designing space systems. For this purpose, the established 10-23 CDM was picked up, with some concepts confirmed as they are in the original model, others updated to fit current needs, and yet others newly introduced. The newly introduced concepts are mainly focused on enabling the automated execution of engineering activities using a reasoner, relying strongly on an ontological description of concepts.

The next chapter will focus on instantiating both the object-oriented and the ontological side of the MBSE CDM for the purpose of demonstrating its utility, applying it to a representative example.

9 Application of the SCDM Framework

This chapter applies the SCDM Framework (SCDMF) with SCDML as language, SCDMP as procedure, and the MBSE CDM as data model on the MagSat scenario (see 3.7). This application is performed for the following reasons:

- Demonstration of the applicability of the developed framework to a representative, previously executed, project.
- Identification of concrete technical benefits that result from the application of this framework.
- Closeout of the requirements identified earlier (see 5.1).
- Evaluation of the hypothesis that using the developed framework improves the utility of the SM.

In addition, this chapter elaborates on how the outlined benefits improve the overall product in terms of faster time to market, reduced cost, and increased system quality.

The demonstration utilizes the MagSat scenario, and details how engineering activities performed during the original spacecraft design process change when performed with the proposed framework. The MagSat scenario, derived from an actual project performed in the past, stands representative of a design of a typical earth observation satellite in terms of complexity, size, and documentation.

Some of the examples illustrated in this chapter are evolved from previous works, most notably Hennig, et al. (2016c). In this thesis, these examples are picked up, and elaborated, both in terms of size of the underlying ontology on M1 level, and in size and complexity of the system itself modeled on M0 level. Furthermore, the relation of the concepts residing on the ontological side of the model architecture is made to those on the object-oriented side.

9.1 Technical Demonstration Setup

The setup used for demonstration is a concrete implementation of the architecture described in 6.3 and picks up on Figure 6.2 from Chapter 6, resulting in the concrete realization detailed in Figure 9.1.

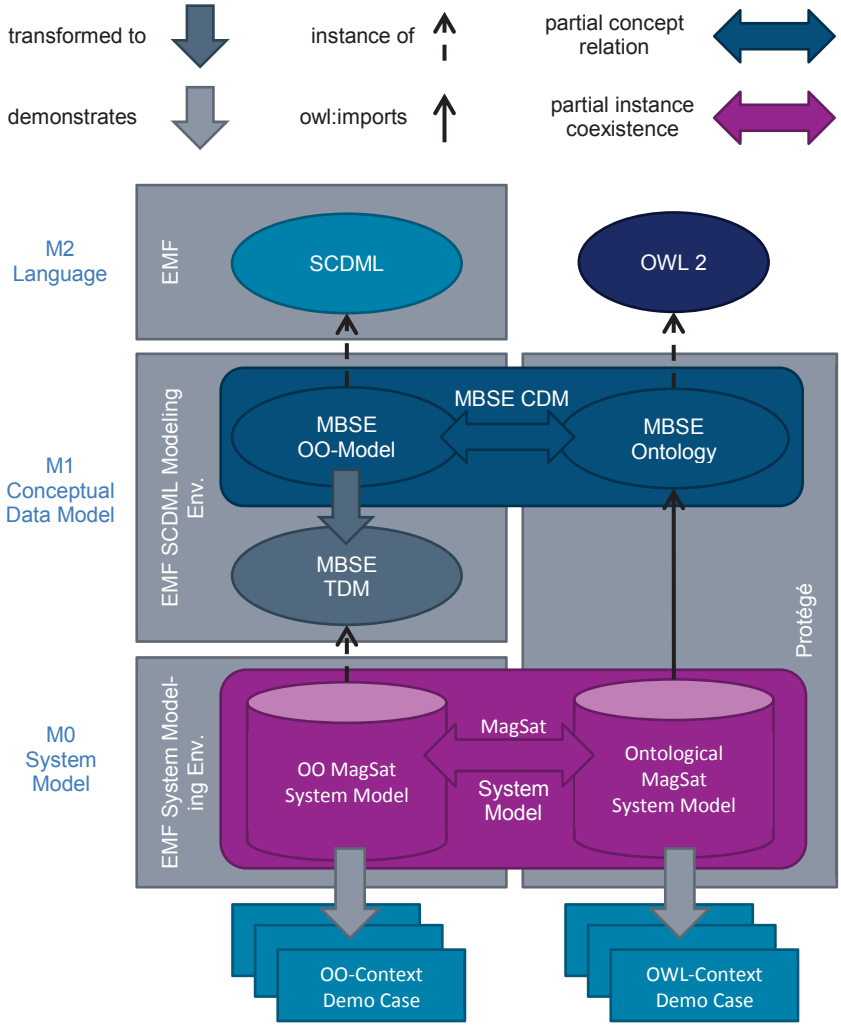


Figure 9.1: Technical demonstration setup

In comparison to the earlier figure, some detailing can be observed at numerous points. For example, the MBSE TDM has been introduced, containing technical aspects of the MBSE OO Model, from which it is derived, facilitating its EMF-based implementation. In addition, concrete technologies are shown in which the different models on M1 and M0 level reside. This includes EMF-based environments on the object-oriented side, and Protégé on the ontology side. Furthermore, until now generic models are made explicit, with the MBSE OO Model and Ontology on M1 level, and the MagSat System Models on M0 level.

9.1.1 Models, Activities, and Tools on Language/M2 Level

On language resp. M2 level, SCDML as described in Chapter 6 is defined using EMF, based on *Eclipse Mars service release 2* (The Eclipse Foundation, 2017a). Furthermore, the conceptual definition of the OWL 2 language also resides on the M2 level, and is taken as-is for this demonstration.

9.1.2 Models, Activities, and Tools on CDM/M1 Level

On the level below, the specification of engineering data in terms of CDMs takes place. This is realized using the MBSE CDM, more specifically using the MBSE object-oriented model and the MBSE Ontology. The MBSE OO Model is defined using the SCDMP as described in Chapter 7, based on SCDML, and occurs as detailed in Chapter 8. The MBSE Ontology also occurs as detailed in Chapter 8.

The MBSE Ontology is modeled using *Protégé 5.1.0* (Musen, 2015), while the MBSE OO Model is defined in an EMF-based environment that includes the plugins necessary to model SCDML-based models. For being able to instantiate the MBSE OO Model, it is mapped to Ecore in a Java-based transformation, forming the MBSE TDM, gaining an additional level of instantiation. This implementation approach has already been realized for implementing a CDM based on ORM 2 syntax in an earlier work (Hennig, et al., 2016a), which is extended and aligned at this point to support the implementation of SCDML-based CDMs.

9.1.3 Models, Activities, and Tools on SM/M0 Level

On SM or rather M0 level, the Object-Oriented MagSat SM instantiates concepts from the MBSE OO Model, also in an EMF-based environment. This environment deploys the plugins that utilize the code generated from the Ecore model that comes out of the transformation from the MBSE OO Model towards the MBSE TDM. At several points, data from the MagSat SM is illustrated using table-based representa-

tions. These representations are defined using the *Sirius* (The Eclipse Foundation, 2017b) modeling environment.

On the ontological side, the Ontological MagSat SM is instantiated with Individuals that are typed by concepts from the MBSE Ontology, also using *Protégé 5.1.0*. For this purpose, the MBSE Ontology, and indirectly also its sub-ontologies, are imported by the Ontological MagSat SM. Reasoning is done using *Pellet 2.2.0* and the OWL-DL reasoner of *SWRLTab 1.0.3*.

As both SMs describe their characteristic, complementary view of the MagSat system, information from both models has to be linked with each other in order to get a full system representation, forming a virtual MagSat SM. For this purpose, the instances of both SMs are related via the link defined on M1 level.

Both sides of the MagSat SM are then used to perform the demonstration cases detailed in 9.2. For the specific demonstration case detailed in 9.5.2.3, a separate ontology containing execution data is used. For the involved ontologies on M0 level, the metrics in Table 9.1 are compiled. As before, the metrics do not count concepts from the imported ontologies, such as the MBSE Ontology. The MagSat Ontology contains the design description of the spacecraft and other relevant data such as its current integration state. The MagSat AFT Ontology contains test execution data that was generated during one run of the MagSat AFT.

Table 9.1: MagSat Ontology and MagSat AFT Ontology Metrics

	MagSat	**MagSat AFT**
Axiom count	1431	250209
Logical axiom count	975	226791
Declaration axioms count	441	23418
Class count	3	0
Object property count	0	0
Data property count	0	0
Individual count	415	23295

The instance-level ontologies perform the imports given in Figure 9.2:

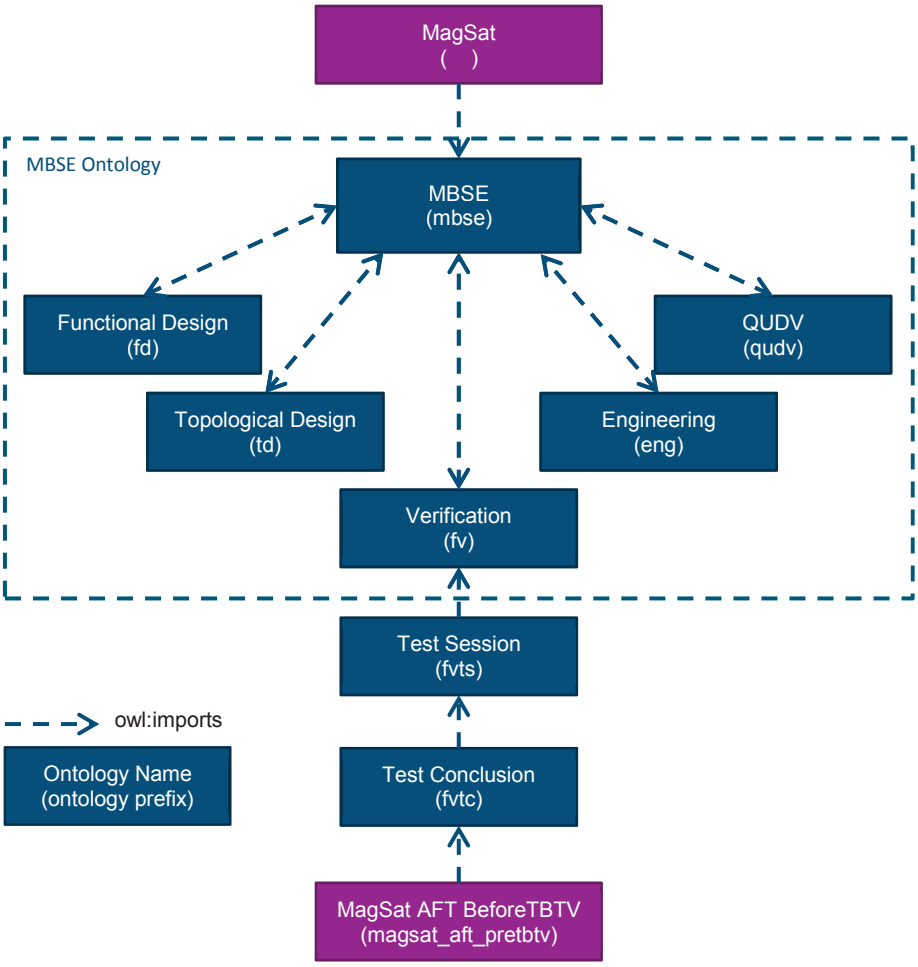


Figure 9.2: Ontology imports on M0 level

9.2 Overview on Demonstration Cases

The selected demonstration cases are shown at a given snapshot in the system's design, but their application is relevant throughout a significant portion of the whole design cycle, beginning at system solution exploration, going over system design, up

to system production. Figure 9.3 gives an overview of how each of the engineering activities of a demonstration case is located within the system's overall lifecycle.

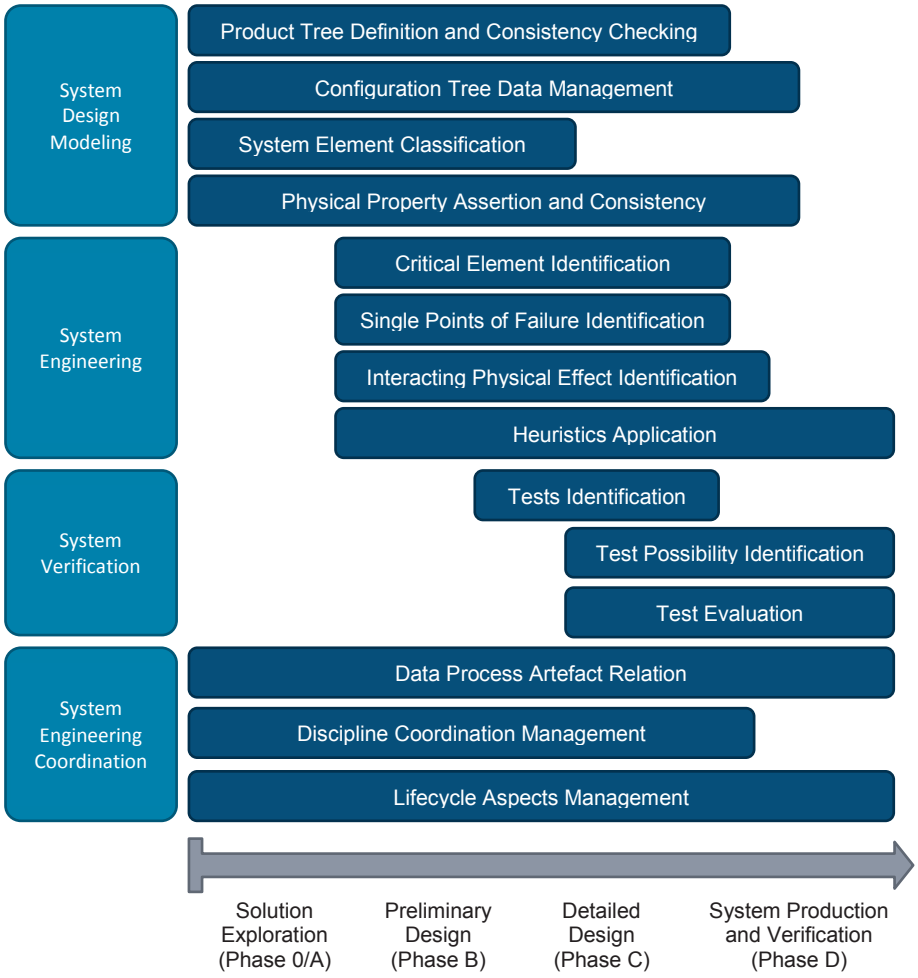


Figure 9.3: Overview on demonstration cases

9.3 System Design Modeling Demonstration Case

This demonstration case deals with the engineering activity of producing the main part of the digital representation of the system's design. This involves instantiating numerous concepts from the MBSE CDM, such as the MagSat's *Product Structure*, its *Operational Design*, its *Topological D*esign, and its physical properties. Also, providing project-specific adaptions, such as customizations of physical properties, or new sets of electrical connectors, is an important activity in this context. Another factor is the necessity for the SM to accurately represent the actual engineering data, being able to identify design inconsistencies, and helping in the identification of facts that are modeled, but are actually not valid in respect to the engineering domains' semantics.

9.3.1 Existing Challenges in Established Process

In the existing approach, the following set of shortcomings identified earlier (see 5.2.1) are of relevance to this engineering activity:

Data representation discrepancies

As the CDMs are mostly produced ad-hoc (see 2.4.1), considering the underlying engineering process, but not directly deriving the data structure from it, a discrepancy between the CDM and how the data is actually decomposed in the engineering process frequently occurs (see 5.2.1.2).

Possibility to produce inconsistent SMs

As outlined in 2.4.3, it is possible to produce inconsistent populations in respect to discipline data, but not inconsistent in accordance to the CDM, due to shortcomings in either the modeling technology or the CDM design process. This applies to the current modeling approaches (see 5.2.1.2 and 5.2.1.1).

Hard-coding of functional dependencies in implementation

Between selected concepts of the CDM, numerous functional dependencies may exist, dictating a specific behavior of the concepts' instances (see 2.4.7). The specification of these functional dependencies is done conceptually on CDM level, but is implicitly performed in the engineering application's program code (see 5.2.1.1).

Distribution of SystemElement description data across numerous SM areas

The multi-disciplinary environment of space engineering results in a number of discipline-specific type considerations of a single *SystemElement* (see 2.4.8). These typing relations are usually introduced via additional, manual type-like references,

scattering the typing of a *SystemElement* across numerous regions of the CDM, and resulting in a dedicated implementation per defined typing reference (see 5.2.1.1).

Trade-off required between semantic accuracy and effective tailoring

For enabling project-specific adaptions of parts of the CDM (see 2.4.2), generic data structures are introduced that allow side-loading during application runtime. This generic nature frequently enables the possibility to produce inconsistent SM populations (see 5.2.1.2).

9.3.2 SCDMF Application

Using SCDML, SCDMP, and the MBSE CDM results in the following SM of the MagSat spacecraft:

9.3.2.1 MagSat Product Tree Definition and Consistency

The structure of the *ProductTree* class of the MBSE CDM was derived using the SCDMP directly from existing engineering data (see 8.3.1), and results in the part of the SM as shown in Figure 9.4:

The MagSat *Product Tree* conforms closely to the underlying source data, offering fields for the name of the *element*, *CI number*, *abbreviation*, *multiplicity*, and if the element has a set-based nature.

On the *MagSat Product Tree*, a number of consistency checks may be performed. One of these checks enforces the *FeatureMultiplicityConstraint* for the *abbreviation* attribute specifying that all *abbreviations* need to be *unique* (see 8.3.1). This means that there may not be two elements in the given context that have identical abbreviations, being in fact a constraint of the actual engineering process. Thus, if two elements have an identical abbreviation as is provoked in Figure 9.5 where both the *Nadir Antenna* and the *Zenith Antenna* are abbreviated with *ANT*, this becomes marked as inconsistent in the SM.

	CI Number	Abbr.	Mult.	isSet
Element Definition MagSat	0000		1	false
Element Definition Electrical Power System	1000	EPS	1	false
Element Definition Power Control and Distribution Unit	1100	PCDU	1	false
Element Definition Battery	1200	BAT	1	false
Element Definition Solar Array +Y	1310	SAPY	1	false
Element Definition Solar Array +Y Aft Panel	1311	SAPYA	1	false
Element Definition Solar Array +Y Bow Panel	1312	SAPYB	1	false
Element Definition Solar Array -Y	1320	SAMY	1	false
Element Definition Solar Array -Y Aft Panel	1321	SAMYA	1	false
Element Definition Solar Array -Y Bow Panel	1322	SAMYB	1	false
Element Definition Data Handling System	2000	DHS	1	false
Element Definition On-Board Computer	2100	OBC	1	false
Element Definition On-Board Software	2200	OBSW	1	false
Element Definition Telemetry, Tracking and Telecommand System	3000	TTC	1	false
Element Definition S-Band Transponder	3100	SBT	2	false
Element Definition 3dB Combiner	3200	SBCP	1	false
Element Definition Nadir Antenna	3300	NA	1	false
Element Definition Zenith Antenna	3400	ZA	1	false
Element Definition RF Harness	3500	SBH	1	true
Element Definition Attitude and Orbit Control System	4000	AOCS	1	false
Element Definition Cold Gas Propulsion System	4100	CGPS	2	false
Element Definition Tank	4110	TANK	1	false
Element Definition Attitude Control Thruster	4120	ACT	8	false
Element Definition Orbit Control Thruster	4130	OCT	4	false
Element Definition High Pressure Latch Valve	4140	HPLV	1	false
Element Definition High Pressure Transducer	4150	HPT	1	false
Element Definition Low Pressure Transducer	4160	LPT	1	false
Element Definition Pipework	4170		1	true
Element Definition Coarse Earth Sun Sensor	4200	CESS	6	false
Element Definition Flux Gate Magnetometer	4300	FGM	3	false
Element Definition Magnetorquer	4400	MTQ	3	false

Figure 9.4: MagSat Product Tree

Element Definition Telemetry, Tracking and Telecommand System	3000	TTC	1	false
Element Definition S-Band Transponder	3100	SBT	2	false
Element Definition 3dB Combiner	3200	SBCP	1	false
Element Definition Nadir Antenna	3300	ANT	1	false
Element Definition Zenith Antenna	3400	ANT	1	false

Figure 9.5: MagSat Product Tree abbreviation consistency checking

In addition, a *RingConstraint* is applied to the *subElements* reference that defines the hierarchy of *ElementDefinitions*, excluding constellations such as cycles. Consequently, inconsistent hierarchies are automatically identified. This is the case in Figure 9.6

where the MagSat's *EPS* contains a *Solar Array* that contains a *Solar Array Panel*, that again contains the *EPS*. This is also marked as inconsistent.

	CI Number	Abbr.	Mult.	isSet
Element Definition MagSat	0000		1	false
Element Definition Electrical Power System	1000	EPS	1	false
Element Definition Power Control and Distribution Unit	1100	PCDU	1	false
Element Definition Battery	1200	BAT	1	false
Element Definition Solar Array +Y	1310	SAPY	1	false
Element Definition Solar Array +Y Aft Panel	1311	SAPYA	1	false
Element Definition Electrical Power System	1000	EPS	1	false
Element Definition Solar Array +Y Bow Panel	1312	SAPYB	1	false
Element Definition Solar Array -Y	1320	SAMY	1	false
Element Definition Solar Array -Y Aft Panel	1321	SAMYA	1	false
Element Definition Solar Array -Y Bow Panel	1322	SAMYB	1	false

Figure 9.6: MagSat Product Tree sub-element consistency checking

Some attributes, such as an *ElementDefinition's CI Number* or *Multiplicity*, are specified as mandatory attributes in the CDM. Consequently, due to the support for closed-world semantics of the MagSat SM, missing values for these attributes become flagged, as they are in fact required data.

9.3.2.2 Automated MagSat Configuration Tree Data Management

Through the introduction of functional rules to both SCDML and the MBSE CDM, the behavior resulting from dependencies between elements of the *ProductTree*, and elements of the *ConfigurationTree* is specified. For starters, the hierarchy of *ElementDefinitions*, their *name*, and their *multiplicity* strictly define the structure and content of all *ElementConfigurations*. The *ElementConfigurations* have to mirror the structure of their defining *ElementDefinitions*. Also, for each *ElementDefinition*, several *ElementConfigurations* may exist, based on the integer value of the multiplicity field. This is specified by a functional rule in the MBSE CDM and applied in the MagSat SM. This results in the model shown in Figure 9.7.

Element Configuration MagSat	[Element Definition MagSat]
Element Configuration Electrical Power System	[Element Definition Electrical Power System]
Element Configuration Power Control and Distribution Unit	[Element Definition Power Control and Distribution Unit]
Element Configuration Battery	[Element Definition Battery]
Element Configuration Solar Array +Y	[Element Definition Solar Array +Y]
Element Configuration Solar Array +Y Aft Panel	[Element Definition Solar Array +Y Aft Panel]
Element Configuration Solar Array +Y Bow Panel	[Element Definition Solar Array +Y Bow Panel]
Element Configuration Solar Array -Y	[Element Definition Solar Array -Y]
Element Configuration Solar Array -Y Aft Panel	[Element Definition Solar Array -Y Aft Panel]
Element Configuration Solar Array -Y Bow Panel	[Element Definition Solar Array -Y Bow Panel]
Element Configuration Data Handling System	[Element Definition Data Handling System]
Element Configuration On-Board Computer	[Element Definition On-Board Computer]
Element Configuration On-Board Software	[Element Definition On-Board Software]
Element Configuration Telemetry, Tracking and Telecommand System	[Element Definition Telemetry, Tracking and Telecommand System]
Element Configuration S-Band Transponder 1	[Element Definition S-Band Transponder]
Element Configuration S-Band Transponder 2	[Element Definition S-Band Transponder]
Element Configuration 3dB Combiner	[Element Definition 3dB Combiner]
Element Configuration Nadir Antenna	[Element Definition Nadir Antenna]
Element Configuration Zenith Antenna	[Element Definition Zenith Antenna]
Element Configuration RF Harness	[Element Definition RF Harness]
Element Configuration Attitude and Orbit Control System	[Element Definition Attitude and Orbit Control System]
Element Configuration Cold Gas Propulsion System 1	[Element Definition Cold Gas Propulsion System]
Element Configuration Tank	[Element Definition Tank]
Element Configuration Attitude Control Thruster 1	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 2	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 3	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 4	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 5	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 6	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 7	[Element Definition Attitude Control Thruster]
Element Configuration Attitude Control Thruster 8	[Element Definition Attitude Control Thruster]
Element Configuration Orbit Control Thruster 1	[Element Definition Orbit Control Thruster]
Element Configuration Orbit Control Thruster 2	[Element Definition Orbit Control Thruster]
Element Configuration Orbit Control Thruster 3	[Element Definition Orbit Control Thruster]
Element Configuration Orbit Control Thruster 4	[Element Definition Orbit Control Thruster]
Element Configuration High Pressure Latch Valve	[Element Definition High Pressure Latch Valve]
Element Configuration High Pressure Transducer	[Element Definition High Pressure Transducer]
Element Configuration Low Pressure Transducer	[Element Definition Low Pressure Transducer]
Element Configuration Pipework	[Element Definition Pipework]
Element Configuration Cold Gas Propulsion System 2	[Element Definition Cold Gas Propulsion System]

Figure 9.7: MagSat Configuration Tree

Secondly, other functional rules are defined in the MBSE CDM that refine the detailed characteristics of *ElementConfigurations* and *ElementOccurrences*. For instance, all aspects that are defined for one *ElementDefinition* have to be inherited by all *ElementConfigurations* of its type, which is shown in Figure 9.8. All aspects, such as operational aspects with a *DiscreteModel*, *FunctionalElectricalAspects* such as *FunctionalPorts*, and *ElectricalAspects* such as *Connectors* are mirrored accordingly at the *ElementConfiguration*, also existing as separate instances, with their properties inherited from the *typing ElementDefinition*.

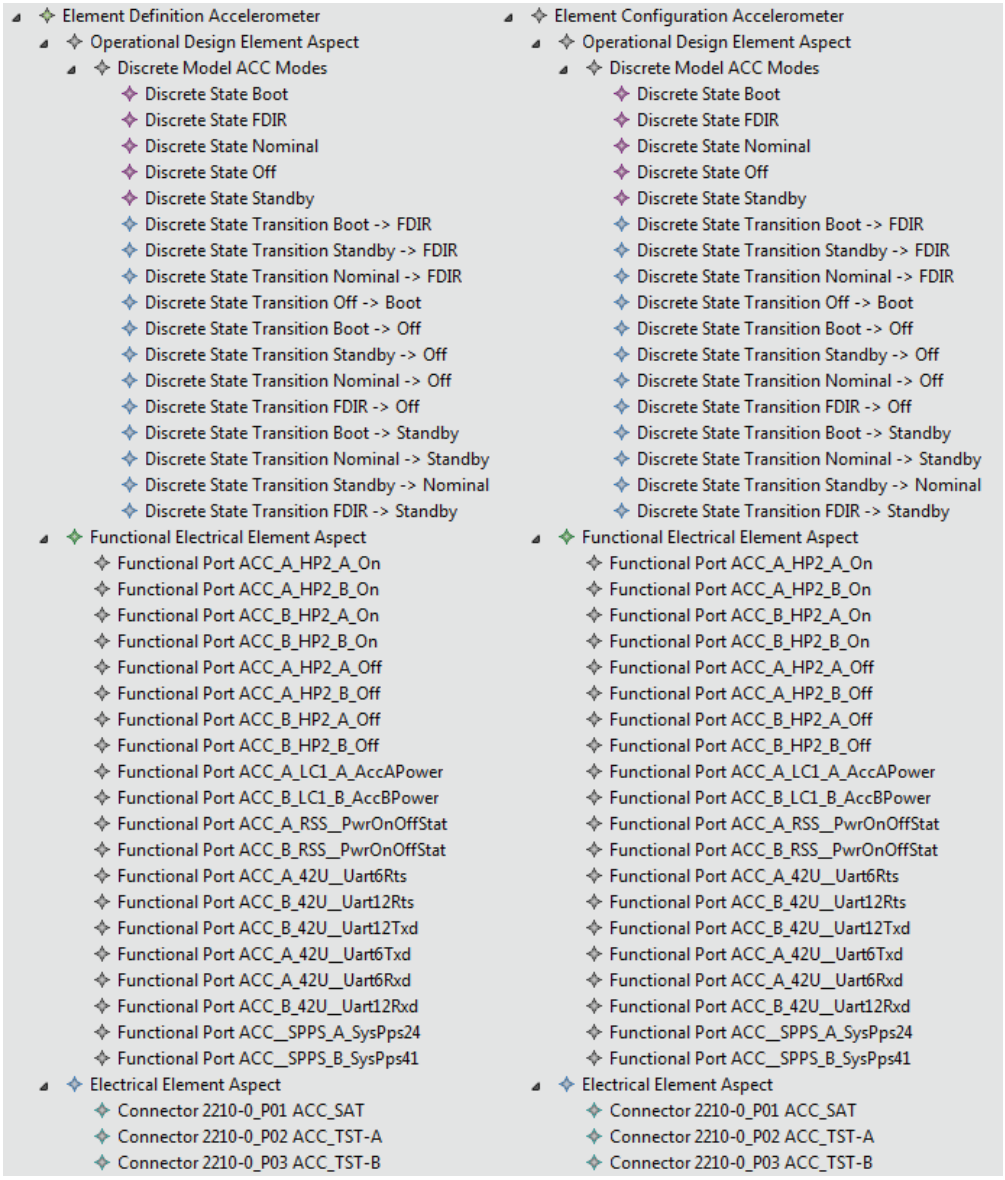


Figure 9.8: Identical System Element Aspects for Definition and Configuration

9.3.2.3 MagSat System Element Classification

In the MagSat Ontology, information for refining the exact nature of *SystemElements* is provided. This involves information regarding which kind of component the elements represent, and an allocation to a system level. This results in, for example, the following assertions:

```
Individual: magsat:MagSat
    Types:
        mbse:ElementDefinition,
        mbse:System

Individual: magsat:AOCS
    Types:
        mbse:AOCS,
        mbse:ElementDefinition,
        mbse:Subsystem

Individual: magsat:ACC
    Types:
        mbse:Accelerometer,
        mbse:Component,
        mbse:ElementDefinition,
        mbse:Equipment

Individual: magsat:GPSRA
    Types:
        mbse:Component,
        mbse:ElementDefinition,
        mbse:GPSReceiverAntenna
```

While the MagSat is mainly characterized as having the types *ElementDefinition* and *System*, the *GPSRA* (GPS Receiver Antenna) is defined as being an *ElementDefinition*, being a *Component*, and being of the *type GPS ReceiverAntenna*. Other elements, as is the case for the *ACC (Accelerometer)*, are characterized as being both a *Component*, as they cannot be decomposed further without impacting their abilities, and being an *Equipment*, as they form one closed functional entity.

9.3.2.4 MagSat Physical Property Assertion and Consistency

The same mechanism is used for asserting *Categories* and consequently *physical properties* to *SystemElements*:

```
Individual: magsat:STRE
    Types:
        mbse:Component,
        mbse:ElementDefinition,
        mbse:OperationalTemperatureRangeElement,
        mbse:StarTrackerElectronics
    Facts:
     mbse:hasMaxNonOperatingTemperature  magsat:VP_STRE_nop_max,
     mbse:hasMaxOperatingTemperature  magsat:VP_STRE_op_max,
     mbse:hasMinNonOperatingTemperature  magsat:VP_STRE_nop_min,
     mbse:hasMinOperatingTemperature  magsat:VP_STRE_op_min

Individual: magsat:VP_STRE_nop_max
    Types:
        mbse:TemperatureValueProperty
    Facts:
     qudv:hasUnit  qudv:degreeCelsius,
     mbse:hasValue  50

Individual: magsat:VP_STRE_nop_min
    Types:
        mbse:TemperatureValueProperty
    Facts:
     qudv:hasUnit  qudv:degreeCelsius,
     mbse:hasValue  -30

Individual: magsat:VP_STRE_op_max
    Types:
        mbse:TemperatureValueProperty
    Facts:
     qudv:hasUnit  qudv:degreeCelsius,
     mbse:hasValue  50

Individual: magsat:VP_STRE_op_min
    Types:
        mbse:TemperatureValueProperty
    Facts:
     qudv:hasUnit  qudv:degreeCelsius,
     mbse:hasValue  -10
```

For the *TemperatureValueProperties*, the information that they are based on the *temperature* quantity kind is given in the MBSE Ontology and inferred by the reasoner. Asserting properties such as temperature ranges to a *SystemElement* is being done on MagSat's ontology side, with use of the MBSE Ontology, that enables a dynamic change of the properties pertaining to the *OperationalTemperatureRangeElement* class. If, for instance, a project would require additional properties, such as *hasMaxStandbyTemperature* and *hasMinStandbyTemperature*, this is possible without requiring a change on the object-oriented CDM, and consequently without requiring a redeployment of the system modeling application.

As this kind of typing of any *SystemElement* offers significant flexibility, the possibility arises to produce inconsistent type constellations in respect to the domain. In order to

avoid these inconsistent assertions, the disjoint relations set between various property classes are evaluated. For instance, the following assertion forces the reasoner to return an inconsistency:

```
Individual: magsat:GPSRE
    Types:
        mbse:BootLoader,
        mbse:Component,
        mbse:ElementDefinition,
        mbse:GPSReceiverElectronics
```

The *GPSReceiverAntenna* is defined as, ultimately, being a *HardwareComponent*. A *BootLoader* is a type of software, with software-related properties, and as such incompatible for instantiation together with a *HardwareComponent*.

Another inconsistency is identified by the reasoner in the following case:

```
Individual: magsat:SBT
    Types:
        mbse:ElementDefinition,
        mbse:Equipment,
        mbse:SBandTransponder,
        mbse:SubsystemMassElement
```

The mass properties used for a Subsystem are not directly applicable to defining the mass properties of an *Equipment*, but structured slightly differently. Consequently, these classes are disjoint, forcing the reasoner to return the inconsistency.

9.3.3 Benefits Resulting from SCDMF Application

9.3.3.1 Improved proximity of system modeling application to actual engineering process

By applying the SCDMP, the CDM is directly derived from existing system-level data of the actual engineering process that is to be supported. This moves the CDM considerably closer to actual engineering data, as the nomenclature of data is that of the underlying engineering process and the data is similarly structured overall. This leads to improved acceptance and utility of the engineering application containing the SM.

The ability of OWL 2 ontologies to have multiple *types* for an *Individual* that may be changed during runtime is a novel concept in this respect. Also, providing this functionality on the side of the object-oriented MagSat model offers a similar functionality in both SMs. Both instantiation principles enable dynamic multi-instantiation in the

MBSE context, improving the information management process, moving the process closer to actual engineering needs, and to the approach originally intended by I0-23.

9.3.3.2 Improved SM utility

By providing a wider variety of constraints on CDM level, and by systematically working towards identifying required constraints using the SCDMP, the CDM becomes richer in terms of constraints. Also, the disjoint concepts offered by OWL 2 enable an additional dimension for ensuring the logical consistency in a multi-classification environment, as is given by the space system design process. These constraints are also available on SM level, where they are used to identify inconsistent model populations. This leads to an improved quality of the SM, and consequently to improved SM utility. In some cases, this can also lead to an improved quality of the system design itself, as design errors are identified that could not be identified before.

The interweaving of both object-oriented and ontological semantics on language level offers the possibility to leverage both OWA and CWA-based semantics. While the closed world is used to check if data that is required to be present is actually present, the open world semantics help in enabling numerous inference activities.

9.3.3.3 Decreased implementation effort of the SM application

The functional dependencies that previously were implicitly defined in the implementation are now available conceptually in the CDM. This makes the functional dependencies, e.g. what information is transferred from one concept to another concept, or what data is created according to which preconditions, conceptually visible on CDM level, and improving on required system modeling application implementation effort.

9.4 System Engineering Demonstration Case

This demonstration case deals with numerous activities that are performed during the main design phases of the MagSat system. In this context, activities such as the identification of single points of failure, the identification of system components with a kind of criticality involved in their design, or the identification of interacting physical effects is taking place. These activities are performed repeatedly, as they have to be re-evaluated once the underlying system design is changed or getting more refined.

9.4.1 Existing Challenges in Established Process

As mentioned in 2.4.1l, many engineering activities in the space system design context are currently not supported in a model-based manner, but form entirely manual activities performed by experienced engineers. This also applies to several of the activities performed during MagSat's design, and consequently to the activities considered in this demonstration case. The following two shortcomings identified earlier are addressed in this section:

Required knowledge not adequately formalized

The knowledge about how to perform a selected engineering activity, and the input information required for this activity, is currently not captured in a model-based manner. Sometimes, this knowledge also cannot be found in a series of documents, but is present implicitly in the experience of involved engineers. In the traditional object-oriented architecture behind system engineering applications, the used technologies are not able to support the formalization of such operational knowledge (see 5.2.1.3).

Engineering activities require manual execution

Due to the lack of capability to adequately capture operational knowledge, there is also no mechanism to automatically perform these activities. Instead, a series of manual steps have to be taken (see 5.2.1.3).

9.4.2 SCDMF Application

Applying the SCDMF in this context enables the automated execution of numerous engineering activities.

9.4.2.1 Automated Identification of Critical Elements

For example, the engineering activity for identifying *CriticalElements* in the system can be automated. For this purpose, numerous criticality categories have been formalized in the MBSE Ontology (see 8.5.1.2). Applying these concepts to the MagSat Ontology with help of a reasoner leads to, for example, the criticality assertions given in Figure 9.9.

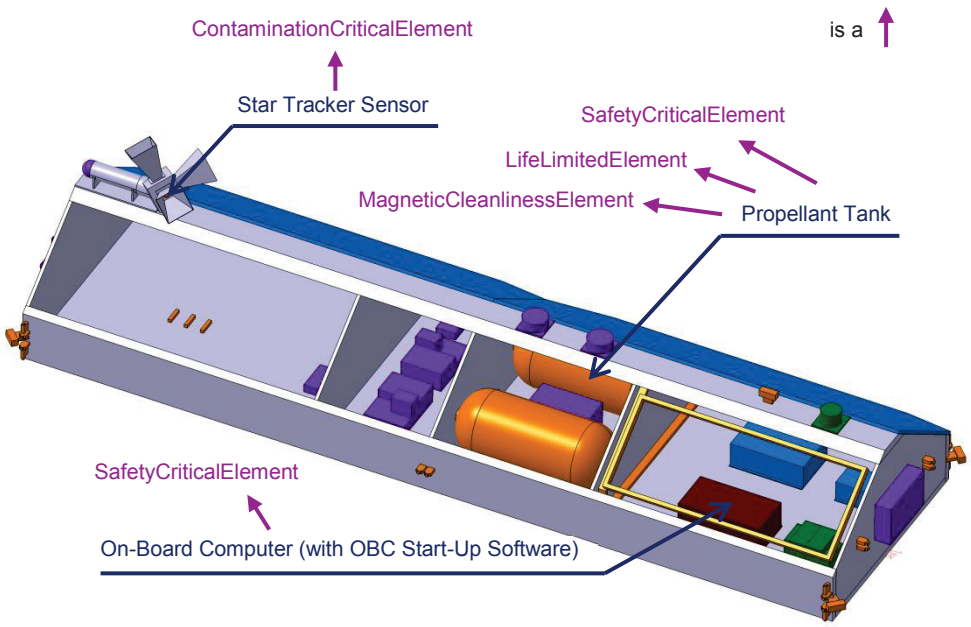


Figure 9.9: MagSat example Critical Element assertions

MagSat's *PropellantTank* is classified as a *LifeLimitedElement*, as is the case for every propellant tank by definition. As pressurization and depressurization can lead to material wear, design margins are prescribed and a total number of permissible cycles is defined that may not be exceeded. Also, the *Propellant Tank* is inferred to being a *MagneticCleanlinessElement*, as it resides aboard a *Spacecraft* that also has *MagneticInstruments* on board that may be *degraded in performance*. As a consequence, a *selection of a non-magnetic material* is required to compensate. Furthermore, the *Propellant Tank* is classified as a *SafetyCriticalElement*, as a failure may lead to *loss of spacecraft* during mission, or to *injury of personnel* during test. As precaution, numerous measures such as the *incorporation of design margins*, a *leak before burst design*, a *burst test, proof pressure test*, and *ultrasonic flaw detection* are prescribed.

```
Individual: magsat:TANK
    Inferred Types:
        eng:LifeLimitedElement,
        eng:MagneticCleanlinessElement,
        eng:SafetyCriticalElement
    Inferred Facts:
        eng:hasFailureEffect  eng:FillVentCyclesLeadToMaterialWear,
        eng:hasFailureEffect  eng:InjuryToPersonnel,
        eng:hasFailureEffect  eng:LossOfSpacecraft,
        eng:hasFailureEffect
          eng:OwnMagneticFieldMayCausePerfDegradOfMagnInstruments,
        eng:hasRiskReductionMeasure  eng:BurstTest,
        eng:hasRiskReductionMeasure  eng:DyePenetrantFlawDetection,
        eng:hasRiskReductionMeasure  eng:IncludeDesignMargins,
        eng:hasRiskReductionMeasure  eng:LeakBeforeBurstDesign,
        eng:hasRiskReductionMeasure  eng:LimitNumberOfCycles,
        eng:hasRiskReductionMeasure  eng:ProofPressureTest,
        eng:hasRiskReductionMeasure  eng:SelectionOfNonMagneticMaterial,
        eng:hasRiskReductionMeasure  eng:UltrasonicFlawDetection
```

MagSat's *OBCStartupSoftware* is also identified as being a *CriticalElement*. Due to the fact that it is of type *BootLoader* and resides in the *OBCStartupMemory*, which is of type *PROM*, the *OBCStartupSoftware* gets classified as an *UnpatchableStartupSoftware*, which forms one of the subclasses of *SafetyCriticalElement*. In the case of an error in this software, occurring under specific circumstances, it is possible that the *OBC* fails to boot. As the boot loader software resides in a *PROM* that can only be written once, it cannot be patched during the mission. As risk reduction measure, elaborate *code inspection procedures* that minimize potential errors are prescribed.

```
Individual: magsat:OBCStartupSoftware
    Inferred Types:
        eng:SafetyCriticalElement
    Inferred Facts:
        eng:hasFailureEffect
          eng:SoftwareBugInPROMLeadsToUnrecoverableStartupFailure
        eng:hasRiskReductionMeasure  eng:PerformISVVCodeInspection
```

MagSat's *STRS* is classified as being a *ContaminationCriticalElement*, as it contains optical parts such as lenses that require specific care procedures. To mitigate the risk involved, *keeping protection covers installed* and the execution of a *final visual inspection* before launch are inferred as measures to be performed.

```
Individual: magsat:STRS
    Inferred Types:
        eng:ContaminationCriticalElement
    Inferred Facts:
        eng:hasFailureEffect  eng:ContaminationOfOpticalSurface
        eng:hasRiskReductionMeasure  eng:KeepProtectionCoversInstalled
        eng:hasRiskReductionMeasure  eng:PerformFinalVisualInspection
```

As this model-supported engineering activity is a re-hosting of an existing non-model based engineering activity, qualitative validation can be performed by comparing identified *CriticalElements* by the reasoner against those of the manually performed process. The result of this comparison is illustrated in Table 9.2.

Table 9.2: Comparison of Critical Elements by reasoner vs. manual process

	Assertions in Document	Inferred Critical Element Assertions	Total Inferred Effects	Total Inferred Risk Reduction Measures
ContaminationCriticalElements	3	4	4	4
LifeLimitedElements	12	6	7	13
MagneticCleanlinessElements	10	7	11	10
SafetyCriticalElements	36	27	45	64
Total	**61**	**44**	**67**	**91**

In the group of *ContaminationCriticalElements*, the reasoner made one assertion more. This can be explained by the treatment of both *Nadir Antenna* and *Zenith Antenna* as a single category of antenna equipment in the manual process, while the reasoner considers both as distinct entities in the MagSat Ontology.

For *LifeLimitedElements*, *MagneticCleanlinessElements*, and *SafetyCritical Elements*, the reasoner made fewer assertions than were made in the manual process. This is due to the fact that the *EEA*, *ZEM*, and *HPM* instruments involve a number of characteristics and mechanisms specific to the MagSat mission that were neither modeled in the MagSat Ontology, nor abstracted with custom concepts in the MBSE Ontology due to their highly specific nature. As such, this forms a tradeoff between modeling effort and overall benefit of the CDM to other projects.

9.4.2.2 Automated Identification of Single Points of Failure

Identifying single-points of failure is also a manual activity, closely related to identifying critical elements. In this case, identifying single points of failure is realized through elaborate modeling of functions, their internal redundancy, and the mapping of functions to *SystemElements* (see 8.5.1.3).

Applying these principles to the MagSat Ontology leads to the single points of failure assertions given in the right column of Table 9.3.

Table 9.3: Comparison of single points of by reasoner vs. manual process

Identified single points of failure in manual activity	**Inferred single points of failure in MBSE Ontology**
ACC	ACC
COMB	COMB
FeedModule	FeedModule
FVV	FVV
HPF	HPF
HPLV	HPLV_1
	HPLV_2
HPMS	HPMS
HPT_1	HPT_1
HPT_2	HPT_2
PCDU	PCDU
SBH	SBH
SBNA	SBNA
TANK	TANK_1
	TANK_2
(ZEM-specific mechanism)	-
14	**15**

For example, the *ACC* forms a single point of failure as it neither exhibits some kind of internal redundancy regarding its functions, nor is it present twice aboard the spacecraft. As the *ACC* is an experimental equipment that forms a bonus objective of the MagSat mission, its failure will not impact the primary mission goals and has thus been deemed acceptable as being a single point of failure. The *HPLV* is regarded as a single component in the original analysis, but the reasoner flagged both *ElementConfigurations* of the *HPLV*, due to the difference in consideration and modeling. The same applies to the *TANK*. An element that was not identified by the reasoner is a *ZEM*-specific mechanism, that has been abbreviated in modeling of the MBSE Ontology due to its highly specific nature.

9.4.2.3 Automated Identification of Interacting Physical Effects

The MBSE Ontology can also be used to identify (undesired) interactions of components aboard the MagSat spacecraft based on their emission of, or susceptibility to, physical effects. For example, the assertions in Figure 9.10 are made.

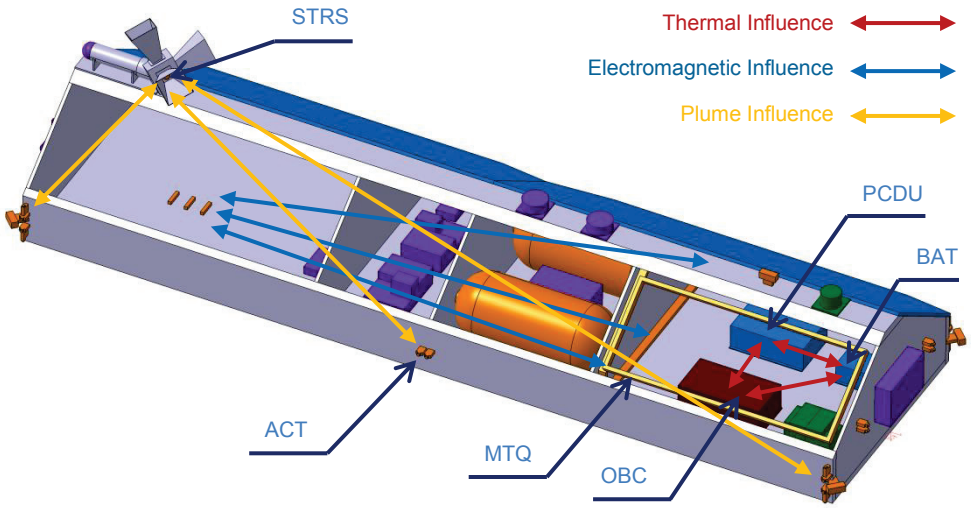


Figure 9.10: Selected interactions of physical effects for MagSat

The *STRS* is an optical instrument that uses a CCD camera with a lens system to acquire an image of the current field of view for determining MagSat's attitude. As such, it is susceptible to effects that obstruct the detriment to optical performance of the instrument, such as the gas plumes produced by firing the *ACT* thrusters for performing attitude control.

Other components, for instance the *Flux Gate Magnetometers* (FGMs) that are used for measuring the Earth's magnetic field for attitude control, are influenced by other on-board components that produce an electromagnetic field themselves. This is the case for the *Magnetorquers* (MTQs), which consist of a series of windings through which electric current is flowing, producing a force while the spacecraft is moving inside the Earth's magnetic field. This effect is used for attitude control of the MagSat.

Components such as the *OBC* (On-Board Computer), the *PCDU* (Power Control and Distribution Unit), and the *BAT* (Battery) are dissipating thermal power during their operation. This heat is influencing other components in the vicinity through conduction and thermal radiation. The assertions made across the whole MagSat model are summarized in Table 9.4.

Table 9.4: MagSat system element physical effect influences

Physical Influenced Element Kind	# elements
Magnetic Influenced Elements	27
Thermal Influenced Elements	30
Vibration Influenced Elements	4
Plume Influenced Elements	7
Outgassing Influenced Elements	6

9.4.2.4 Highlighting of Required Actions through Heuristics

The MBSE Ontology also contains a number of defined heuristics that, based on engineering rules of thumb, highlight specific aspects of *SystemElements*.

For example, any *ElementDefinition* with at least three *ProblemReports* attached to it, is required to be of increased attention, and is thus classified as a *TenuousElement* or rather *MultiplePRElement*. This is the case for the *STRApplicationSoftware*, as it has three PRs assigned.

```
Individual: magsat:STRApplicationSoftware
    Types:
        mbse:ApplicationSoftware,
        mbse:ElementDefinition
    Facts:
     mbse:hasEngineeringAnnotation  magsat:PR_SYS_01,
     mbse:hasEngineeringAnnotation  magsat:PR_SYS_07,
     mbse:hasEngineeringAnnotation  magsat:PR_SYS_11
    Inferred Types:
        mbse:MultiplePRElement
```

ElementDefinitions that were subject of numerous *Review Item Discrepancies* (RIDs) also require increased attention. In this case, an element with more than six *RIDs* is tenuous, however if it is also a *CriticalElement*, it is tenuous if it has at least three *RIDs* associated with it. For example, MagSat's *AOCS* is classified as a *MultipleRIDElement*, as it has seven *RIDs* associated with it:

```
Individual: magsat:AOCS
    Types:
        mbse:AOCS,
        mbse:ElementDefinition,
        mbse:Subsystem
    Facts:
     mbse:directlyContainsElement  magsat:CESS,
     mbse:directlyContainsElement  magsat:CGPS,
     mbse:directlyContainsElement  magsat:FGM,
     mbse:directlyContainsElement  magsat:MTQ,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_01,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_02,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_03,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_04,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_05,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_06,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_07
    Inferred Types:
        mbse:MultipleRIDElement
```

The *BAT* is also classified as a *MultipleRIDElement*, although it has only three RIDs. However, the *BAT* being a *CriticalElement* lowers the threshold.

```
Individual: magsat:BAT
    Types:
        mbse:Battery,
        mbse:Component,
        mbse:ElementDefinition,
    Facts:
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_08,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_09,
     mbse:hasEngineeringAnnotation  magsat:RID_ENG_10
    Inferred Types:
        mbse:MultipleRIDElement
```

Furthermore, *ElementDefinitions* that have assumed parameters that do not yet have their value validated by some kind of analysis are marked. While this is not problematic in the beginning of the system's design, it is required to have any previously assumed parameter confirmed once the design reaches maturity.

```
Individual: magsat:ACT
    Types:
        mbse:AttitudeControlThruster,
        mbse:Component,
        mbse:ElementDefinition,
        mbse:ThrusterPropertyElement
    Facts:
     mbse:hasSpecificImpulse  magsat:VP_ACT_specificImpulse,
     mbse:hasVacuumThrust  magsat:VP_ACT_thrust
    Inferred Types:
     eng:AssumedParameterElement

Individual: magsat:VP_ACT_specificImpulse
    Types:
        mbse:SpecificImpulse
    Facts:
     mbse:hasMaturityStatus  mbse:TeamAssumption,
     qudv:hasUnit  qudv:second,
     mbse:hasValue  50

Individual: magsat:VP_ACT_thrust
    Types:
        mbse:Thrust
    Facts:
     mbse:hasMaturityStatus  mbse:TeamConfirmed,
     mbse:hasUnit  qudv:milliNewton,
     mbse:hasValue  35
```

9.4.3 Benefits Resulting from SCDMF Application

Using an approach as demonstrated provides the following benefits:

9.4.3.1 Improved Formalization of Operational Knowledge

Knowledge about how to execute a given engineering activity, and information required as input is now formalized. While previously, this knowledge was not present in a model-based format, sometimes not even made explicit at all, the knowledge is now present in a semantic model.

On the one hand, this enables improved specification of and communication about relevant engineering knowledge. Furthermore, this knowledge will remain as experts of the underlying engineering activities retire, leave the organization, or move on to other duties.

Also, the base containing this knowledge may grow from project to project, continuously integrating lessons learned. This is illustrated in Figure 9.11. There, a knowledge base already containing knowledge from previous projects is shown, which can be applied to running projects using a reasoner. These projects may produce new

knowledge in terms of lessons learned, which may be considered valuable for future projects. Consequently, these bits of knowledge are formalized and fed back to the knowledge base, continuously extending it from project to project.

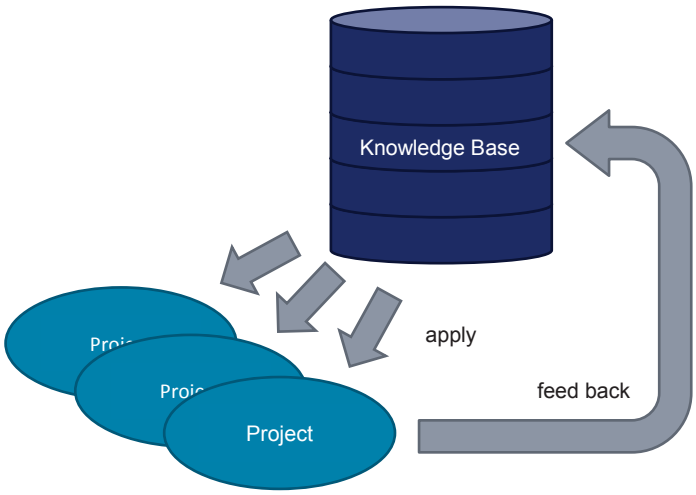


Figure 9.11: Knowledge Base and knowledge application to projects

During this feedback, an important activity is to differentiate between knowledge only relevant for a specific project, and knowledge that will also apply to other projects. For the first case, the knowledge is best stored in a project-specific knowledge base, while in the latter case it should be integrated into the generic knowledge base. In order to facilitate this, the organization has to formalize the process of extracting lessons learned from running and completed projects, and formalizing it in the knowledge base, ensuring that no project remains untreated by the feedback loop.

9.4.3.2 Automatic Application of Operational Knowledge

The knowledge specified in the Engineering Knowledge Base not only serves as a specification, but can be applied to system design data with a reasoner. This is enabled by the semantic substructure underlying OWL 2 based on DL. Furthermore, this improves the overall efficiency of the underlying engineering process, as the activity takes less time, as it merely has to be supervised by an activity expert, but does not bind the expert for a significant amount of time for activity execution.

9.4.3.3 Improved Scaling of Engineering Activities

By automatically applying modeled operational knowledge about a specific engineering activity on a model of a system, the engineering activity is essentially executed in an automated manner. This enables the engineering activity to be executed on significantly larger and more complex systems, as it requires an expert to supervise the activity, but not to directly perform its execution.

9.4.3.4 Improved System Design Quality

Having an engineering activity performed automatically based on a given set of defined knowledge ensures consistent execution of the activity on any given dataset. Furthermore, no element of the system is forgotten to be considered in the activity, as the executing algorithm works consistently across all datasets. This ensures that all elements of the system are examined in the way the engineering activity prescribes, without a system element being forgotten or skipped without notice.

9.4.3.5 Improved Process Efficiency

Through applying heuristics that highlight points of increased attention in the system's design, these critical points in the system do not have to be manually managed throughout the design cycle.

9.4.3.6 Traceability of Activity Execution

The information the reasoner has added to the model during inference can be traced across the whole logical chain that led to the ultimate inference. This enables full traceability of all involved reasoning steps of a modeled engineering activity, helping for design and analysis justification, and for more elaborate system design understanding.

9.5 System Verification Demonstration Case

This demonstration case is situated near the end of the system's design cycle, as MagSat is being assembled, integrated, and tested. After this activity, the satellite should be in a utilizable state and ready to be launched for subsequent operation. In this phase, significant testing is being performed, ranging from very early integration stages of the satellite with only few components, up to the fully integrated spacecraft.

The motivation behind these tests is to validate the correct assembly of the system, ensure the correctness of the overall design, and to formally verify applicable requirements.

9.5.1 Existing Challenges in Established Process

The current system verification process relies significantly on manual execution of activities (see 5.2.1.3). More specifically, the following concrete challenges arise within the current system verification process:

Involvement of continuous manual tracking activities

As outlined in 2.4.11, a high number of manual activities are performed. In the context of system verification for example, a lot of effort is put into manually tracking meta-information of a performed activity, and deriving the meaning of this information for the current testing activity. This includes, for instance, manually collecting information about how often a certain component has been switched on, how often a tank was pressurized and depressurized, or how often a component has been re-integrated. As was identified in 5.2.1.3, no support for this manual activity is currently given by the current modeling approaches.

Manual evaluation of large amounts of system execution data

In 2.4.10, the high relevance of system execution data was outlined. However, as demonstrated in 5.2.1.3, no dedicated support for considering system execution data in the scope of system modeling is currently given by the existing approaches. Furthermore, evaluation of this data is currently a completely manual approach (also see 5.2.1.3)

Non-semantic specification of knowledge

A challenge that was already mentioned in the previous demonstration case in 9.4.1 is also relevant in this context. The production phase of a spacecraft also relies significantly on collecting and applying operational knowledge amassed across past projects and the running project in the course of the integration and testing campaigns. This knowledge is required and used to correctly operate and debug the spacecraft. In this context, this knowledge is also not formalized, and applied in a manual process (also see 5.2.1.3).

9.5.2 SCDMF Application

The SCDMF can be used to make a number of improvements on selected engineering activities in the context of system verification.

9.5.2.1 Automated Identification of Required Tests

The part of the system's design dealing with verification also has its own lifecycle. The activities to be performed during system verification are first specified, then implemented, and subsequently executed. The phase of specification can be supported by applying operational knowledge about what kinds of tests are required for which elements in the system.

What kinds of tests are required to be executed on a given element depends on characteristics, or combinations of characteristics, that the element exhibits. For example, each element that is defined as representing an *Equipment*, i.e. each element encapsulating a specific function, has to undergo an *Integrated System Test* (IST). On the other hand, each component having a *Connector* has to undergo an *Electrical Integration Test* (ELI). While in some cases, it may occur that an element requires an *IST* and an *ELI* at the same time, other cases where the system is differently structured may arise where an *ELI* is required, but no *IST*, as the *Equipment* is not allocated to a specific component, but to a combination of components.

For example, the following assertions can be made:

```
Individual: magsat:ACC
    Inferred Facts:
     ver:requiresTest  ver:IST,
     ver:requiresTest  ver:ELI
```

For the *ACC*, an *IST* (ACC IST) is required, as it forms an *Equipment* that encapsulates a specific function. As the *ACC* also has a number of electrical *Connectors*, it also requires an *ELI* (ACC ELI). However, there are also other constellations:

```
Individual: magsat:STR
    Inferred Facts:
     ver:requiresTest  ver:IST

Individual: magsat:STRE
    Inferred Facts:
     ver:requiresTest  ver:ELI

Individual: magsat:STRS
    Inferred Facts:
     ver:requiresTest  ver:ELI
```

In this case, the *STR* is defined as an *Equipment*, but it is not one single component, but a combination of electronic control units and sensors that together provide the specified function. Consequently, the *STR* as equipment has an *IST*, but its components each require an *ELI*.

In other cases, tests are performed because they are prescribed either by standards applicable to the given context, by internal regulations, or by specific requirements. For example, The *AOCS*, due to its characteristic control loop nature, requires *a Closed Loop Test* (CLT) that ensures that all actors, sensors and the control algorithm work correctly in conjunction. In other cases, the assertions are fairly simple, as each *On-Board Control Procedure* (OBCP) by definition requires an *OBCP IST*.

```
Individual: magsat:AOCS
    Inferred Facts:
     ver:requiresTest  ver:CLT
```

For the whole spacecraft, a variety of tests are required, such as:

```
Individual: magsat:MagSat
    Inferred Facts:
     ver:requiresTest  ver:AFT,
     ver:requiresTest  ver:EMCFT,
     ver:requiresTest  ver:MFT,
     ver:requiresTest  ver:RFCFT,
     ver:requiresTest  ver:SFT,
     ver:requiresTest  ver:TVFT
```

For validation, identified tests to be performed are contrasted with the tests identified for the original MagSat design in the selected groups (Table 9.5).

Table 9.5: Comparison of manual and automated test identification

Test Type	Original MagSat	MagSat Ontology	Comment
IST	12	11	OBC IST split into OBC with MMU and without MMU
ELI	21	20	Specific mechanism left out in modeling
AFT	1	1	
SFT	1	1	
CLT	1	1	
EMCFT	1	1	
RFCFT	1	1	
TVFT	1	1	
OBCP IST	47	45	Power-Up OBCP split into three separate tests
Total	**86**	**82**	

The difference in *ISTs* occurs due to the *OBC IST* being split between an *OBC IST* without *Mass Memory Unit (MMU)*, and a dedicated *MMU IST* without the rest of the *OBC*. While the MagSat Ontology considers the *MMU* as a sub functional unit of the *OBC*, the modeling of required tests is too generic for this case. The missing *ELI* is due to the fact that a specific mechanism present in the original MagSat design was left out in modeling due to its highly specific nature. The difference in *OBCP ISTs* occurs because the procedure for *OBC power-up and start-up* is tested in three different configurations, where in the first test, an interrupt of the procedure is provoked, in the second test the procedure is performed on the *nominal side* of the *OBC*, and in the third test on the *redundant side*. This differentiation is not considered by the MBSE Ontology.

9.5.2.2 Automated Identification of Possible Tests to Execute

During the integration and testing campaign, the configuration of the integrated satellite changes frequently. Due to the prototypical nature of the system and its components, a significant amount of ad-hoc debugging and problem solving is required. Planning ahead on which units will be ready for integration and test at a given time is challenging, as significant uncertainties have to be taken into account. This is further complicated in cases where a constellation of multiple satellites is integrated in parallel, and only a limited number of testing equipment is available. Consequently, a great deal of flexibility is required for the test campaign, and a considerable amount of uncertainty has to be dealt with.

For example, the *STR IST* can only be performed with at least the following (project-specific) hardware configuration:

```
Class: magsat:STRISTCapableSystem
    EquivalentTo:
        mbse:ElementOccurrence
         and mbse:System
         and (mbse:containsElement some
            (mbse:CoreEGSE
             and (mbse:integrates some mbse:CoreEGSE)))
         and (mbse:containsElement some
            (mbse:HighPrecisionMagnetometer
             and (mbse:integrates some
               mbse:HighPrecisionMagnetometer)))
         and (mbse:containsElement some
            (mbse:LaunchPowerSupply
             and (mbse:integrates some mbse:LaunchPowerSupply)))
         and (mbse:containsElement some
            (mbse:OnBoardComputer
             and (mbse:integrates some mbse:OnBoardComputer)))
         and (mbse:containsElement some
            (mbse:PCDU
             and (mbse:integrates some mbse:PCDU)))
```

```
        and (mbse:containsElement some
            (mbse:STRLensCovers
             and (mbse:integrates some mbse:STRLensCovers)))
        and (mbse:containsElement some
            (mbse:STROGSE
             and (mbse:integrates some mbse:STROGSE)))
        and (mbse:containsElement some
            (mbse:STRProtectiveCovers
             and (mbse:integrates some mbse:STRProtectiveCovers)))
        and (mbse:containsElement some
            (mbse:STRUnitTester
             and (mbse:integrates some mbse:STRUnitTester)))
        and (mbse:containsElement some
            (mbse:StarTrackerElectronics
             and (mbse:integrates some mbse:StarTrackerElectronics)))
        and (mbse:containsElement some
            (mbse:StarTrackerSensor
             and (mbse:integrates some mbse:StarTrackerSensor)))
        and (mbse:containsElement some
            (mbse:TMTCFrontEnd
             and (mbse:integrates some mbse:TMTCFrontEnd)))

    SubClassOf:
        ver:TestCapableSystem
```

Furthermore, an integrated MagSat is supplied in the MagSat Ontology. In this MagSat representation, many elements are modeled as being integrated, as is the case for all elements required for an *STR IST,* such as a *HighPrecisionMagnetomer,* the *PCDU,* the *OBC, StarTrackerElectronics, StarTrackerSensors,* etc. Consequently, the reasoner concludes:

```
Class: magsat:MagSat
    Inferred Types: magsat:STRISTCapableElement
```

To generalize, such definitions of required components can be used to automatically determine which kinds of tests can be performed using the current integration state of the satellite, avoiding complex manual evaluation and cross-checking activities.

As this analysis activity is not an activity that is explicitly documented, no direct comparison with actual validation data can be given. Validation of the demonstrated principles is instead performed by providing a MagSat configuration that is applicable for an *STR IST* and a *GPSR IST*, but not to other tests, such as an *AFT* or *ACC IST*.

9.5.2.3 Automated Evaluation of Performed Tests

Once a test is performed, its success or failure has to be determined by evaluating the data generated during the test session. For improving the effectivity and efficiency of

this activity, the automated process outlined in Figure 9.12 is provided, utilizing the example of the *MagSat AFT*:

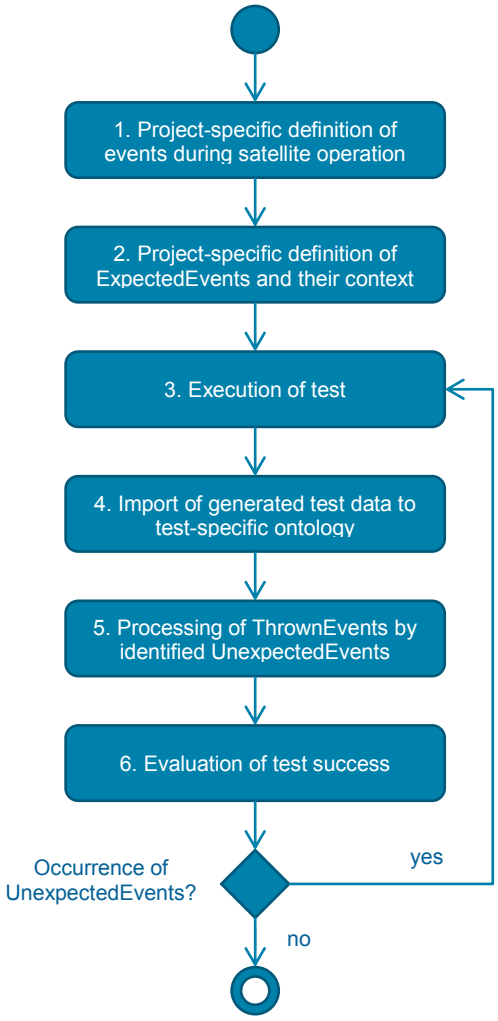


Figure 9.12: MagSat automated test evaluation process

As the initial step, a project-specific definition of the events generated during MagSat's operation, and consequently also its test, is required. These events are defined by the system's *OBSW* (On-Board Software) and structured according to the system's design and are thus system-specific. For instance, the following events are defined:

```
Class: fvtc:ACC_InvalidPpsDetectedErrorRep
    EquivalentTo:
        fvts:EventReport
         and (fvts:eventId value 528)
    SubClassOf:
        fvtc:AnomalyEvent

Class: fvtc:GPSDataLost
    EquivalentTo:
        fvts:EventReport
         and (fvts:eventId value 512)
    SubClassOf:
        fvtc:CriticalEvent

Class: fvtc:UARTProtErr
    EquivalentTo:
        fvts:EventReport
         and (fvts:eventId value 882)
    SubClassOf:
        fvtc:WarningEvent
```

For example, each *Event*, more specifically each *AnomalyEvent* with an *eventId* of *528*, is classified as an ACC_*InvalidPpsDetectedErrorRep*. This event is generated by the satellite in cases where the *ACC equipment* does not receive a correct pulse-per-second (PPS) signal for timing purposes and can thus not perform its operation correctly. The *GPSDataLost* event with *eventId 512* is thrown if no valid GPS signal is received by the *OBC*. The *UartProtErr* event with *eventId 882* is generated when faulty transmissions across MagSat's *UART* interfaces are detected.

While a *GPSDataLost* is always a *CriticalEvent*, the occurrence of this event might not impact test success in the end. For example, this event is generated after *OBC cold boot* is completed once the *OBC* is operating and detecting that no GPS signal is available. In order to enable the GPS signal, the *GPSR equipment* has to be put into operation. The *GPSDataLost* event is also generated after the *GPSR equipment* is taken out of its operational mode, as this also causes a loss of navigation signal. In both cases, the *GPSDataLost* is an *ExpectedEvent* in the context of this test.

In order to evaluate this, all *ExpectedEvents* are modeled in the MagSat test conclusion ontology as SWLR rules. For example, the following rule is used to detect expected *GPSDataLost* events after *OBC boot*:

```
GPSDataLost(?e) ^ fvts:dateTimeLocalMilliseconds(?e, ?timeEvent) ^
OBCStart(?rp) ^ fvts:dateTimeLocalMilliseconds(?rp, ?timeReadPacket) ^
swrlb:subtract(?diff, ?timeEvent, ?timeReadPacket) ^ swrlb:lessThan(?diff,
180000) ^ swrlb:greaterThan(?diff, 0) -> ExpectedEvent(?e)
```

The rule states that a *GPSDataLost* event that occurs up to 180 seconds after the OBC start is confirmed, is classified as an *ExpectedEvent*.

For concluding on *ACC_InvalidPpsDetectedErrorRep*, the following rule is defined, stating that the event is expected, if it occurs within *60* seconds of sending the command to boot the *ACC instrument*:

```
ACC_InvalidPpsDetectedErrorRep(?e) ^ fvts:dateTimeLocalMilliseconds(?e,
?timeEvent) ^ TTC00049(?rp) ^ fvts:dateTimeLocalMilliseconds(?rp,
?timeReadPacket) ^ swrlb:subtract(?diff, ?timeEvent, ?timeReadPacket) ^
swrlb:lessThan(?diff, 60000) ^ swrlb:greaterThan(?diff, 0) ->
ExpectedEvent(?e)
```

An expected *UARTProtErr* is identified with the following rule, stating that the event is not problematic if it occurs within *20* seconds after the instrument in question is turned on by enabling its power interface:

```
UARTProtErr(?e) ^ fvts:dateTimeLocalMilliseconds(?e, ?timeEvent) ^
PHC20021(?rp) ^ fvts:dateTimeLocalMilliseconds(?rp, ?timeReadPacket) ^
swrlb:subtract(?diff, ?timeEvent, ?timeReadPacket) ^ swrlb:lessThan(?diff,
20000) ^ swrlb:greaterThan(?diff, 0) -> ExpectedEvent(?e)
```

In the third step of this process, the data generated by the test session is imported into the ontology. More specifically, the data shown is from the *MagSat AFT* that was performed before the *Thermal Balance/Thermal Vacuum* (TB/TV) Test. This ontology contains the actual *EventReports* and other data generated during test, such as the commands sent and telemetry received.

```
Individual: aft:ReadPacket_5097
    Types:
        fvts:ReadPacket
    Facts:
     fvts:actualCount  1,
     fvts:actualDuration  "20000"^^xsd:long,
     fvts:conclusion  "OK"^^xsd:string,
     fvts:dateTimeLocal  "2011-06-22T04:26:22"^^xsd:dateTime,
     fvts:dateTimeLocalMilliseconds  "1308709582710"^^xsd:long,
     fvts:dateTimeMission  "2011-06-22T04:20:51"^^xsd:dateTime,
     fvts:dateTimeMissionMilliseconds  "1308709251517"^^xsd:long,
     fvts:expectedCount  1,
     fvts:expectedDuration  "180000"^^xsd:long,
     fvts:logSequenceCount  5097,
     fvts:packetDescription  " [STB 2.8.9] ColdStart"^^xsd:string,
     fvts:packetName  " OS_EVT20086"^^xsd:string,
     fvts:print  "MUTE"^^xsd:string
```

```
Individual: aft:EventReport_17
    Types:
        fvts:EventReport
    Facts:
     fvts:apid  55,
     fvts:dateTimeLocal  "2011-06-22T04:26:44"^^xsd:dateTime,
     fvts:dateTimeLocalMilliseconds  "1308709604160"^^xsd:long,
     fvts:dateTimeMission  "1999-01-01T12:00:22"^^xsd:dateTime,
     fvts:dateTimeMissionMilliseconds  "946681222160"^^xsd:long,
     fvts:dateTimePacket  "1999-01-01T12:00:22"^^xsd:dateTime,
     fvts:dateTimePacketMilliseconds  "946681222000"^^xsd:long,
     fvts:eventId  512,
     fvts:logSequenceCount  17,
     fvts:packetContent  "Event Id : GPSDataLost"^^xsd:string,
     fvts:pusSubtype  4,
     fvts:pusType  5,
     fvts:sourceSequenceCount  0

Individual: aft:SatCmd_49101
    Types:
        fvts:SatCmd
    Facts:
     fvts:commandTarget  "TTC00049()"^^xsd:string,
     fvts:conclusion  "OK"^^xsd:string,
     fvts:dateTimeLocal  "2011-06-22T06:03:32"^^xsd:dateTime,
     fvts:dateTimeLocalMilliseconds  "1308715412365"^^xsd:long,
     fvts:dateTimeMission  "2011-06-22T06:03:30"^^xsd:dateTime,
     fvts:dateTimeMissionMilliseconds  "1308715410365"^^xsd:long,
     fvts:description  " [STB 2.8.9] OBSW_UartAccAEna"^^xsd:string,
     fvts:echo  " 184C C1F4 000F 1908 8000 0800 0002 2000 0002 6000
       EE0F"^^xsd:string,
     fvts:logSequenceCount  49101,
     fvts:report  " no errors occurred"^^xsd:string,
     fvts:serviceOne  " 0841 C3E8 0015 1001 0151 105E 014C D0A2 02BB
       184C C1F4 0000 0000 4CEA"^^xsd:string,
     fvts:serviceTwo  " 0841 C3E9 0015 1001 0751 105E 014C D0A2 03D6
       184C C1F4 0000 0000 E056"^^xsd:string,
     fvts:sourceSequenceCount  500,
     fvts:tcType  "STANDARD"^^xsd:string,
     fvts:verificationTimeout  "5000"^^xsd:long,
     fvts:verificationType  "AUTO"^^xsd:string
```

Using this data, the test session can be evaluated with help of the SQWRLTab OWL-DL reasoner. The reasoner infers, for instance, the following statement:

```
Individual: aft:EventReport_17
    Types:
        fvts:EventReport
    Inferred Types:
        fvts:CriticalEvent
        fvts:ExpectedEvent
```

EventReport_17 represents a *CriticalEvent* that is thrown after the *OBC* becomes operational and detects that currently no GPS signal is received. However, as the event was received that *OBC cold boot* was completed, and as this message was received moments ago, *EventReport_17* is classified as an *ExpectedEvent*.

In order to validate all findings, the ontology-based test conclusion is contrasted with the conclusion of the actual performed *AFT* before the *TB/TV test*. Both analyses contain 255 *NormalEvents* that were all classified as *ExpectedEvents*. Both approaches also detected identical amounts of *WarningEvents*, where five were expected and four were unexpected. Regarding *AnomalyEvents*, a discrepancy occurs where manual evaluation of the actual test yielded 19 *ExpectedEvents* with no *UnexpectedEvents*, but the ontology-driven evaluation yielded 18 *ExpectedEvents* and one *UnexpectedEvent*. For *CriticalEvents*, a total of 22 *ExpectedEvents* and no *UnexpectedEvents* were recorded (Table 9.6).

Table 9.6: Comparison of manual and reasoner-based AFT evaluation

Event Type	Manual Evaluation		Reasoner-based Evaluation	
	# expected	# unexpected	# expected	# unexpected
Normal Events	255	0	255	0
Warning Events	5	4	5	4
Anomaly Events	19	0	18	1
Critical Events	22	0	22	0
Total	**301**	**4**	**300**	**5**

The single discrepancy between both approaches is explained by a faulty import of test result data into the ontological format. The system used for recording execution data from the test session writes this data into a table-based log concurrently with other applications. This can make data interpretation difficult, as import interpretation depends on numerous rows occurring together, which can get interrupted by another application. This led to the fact that a telecommand used for concluding on an expected *ZEM_Delayed TimeTC* was not correctly recognized by the importer and thus was not properly transformed it into the MagSat AFT ontology. For validation, it is concluded that this does not impact the validity of the demonstrated approach, as no false positives can occur.

In terms of overall test evaluation, the used run of the *AFT* before the TB/TV test failed, as four and five *UnexpectedEvents* occurred, respectively. The procedure for

evaluating correct function of the GPSR failed and generated four unexpected *GPSR_EvtLowFirstNavigationFixTimeOut* events that ultimately led to test failure. This conclusion is shared between both the original and the ontological approach.

9.5.2.4 Managing Implications of Collected Test Meta-Data

During the conduction of tests, a lot of meta-data is collected. This data includes, for example, information about how often a specific component was integrated and taken out of the satellite, or how often this component was switched on. This also falls into the scope of the previously introduced Engineering Heuristics.

To leverage this information, rules are defined in the MBSE Ontology (see 8.5.2) that can be applied to the MagSat, highlighting, for example the following aspects:

```
Individual: magsat:ER_GPSRE_SN02
    Types:
        mbse:ElementRealization,
        mbse:GPSReceiverElectronics
    Facts:
     ver:maxNoOfTimesSwitchedOn  "25.0"^^xsd:double,
     ver:noOfTimesSwitchedOn  "22.0"^^xsd:double
    Inferred Types:
        mbse:TestingStressToBeMinimzedElement
```

In the example above, the *GPSRE* with *serial number SN02* has a maximum of 25 cycles specified, for which it is safe to be switched on and off. However, the component was already switched on 22 times, violating the threshold of 70% of reached on-switches, and consequently gets classified as a *TestingStressToBeMinimzedElement*, indicating that tests on this component should be minimized from now on.

9.5.3 Enabled Benefits

The activities detailed in this section enable a variety of benefits on the overall system engineering process:

9.5.3.1 Improved Scaling of Engineering Activities

By being able to perform an automated execution of a given engineering activity, the execution of more complex engineering activities becomes possible. As the engineers require less effort to perform a given activity, capacities are freed up that can be used to manage increased system complexity. For example, the effort required to manually evaluate what tests need to performed on the system can be spent on other points after the process has been automated. Although the automated process still has to be

supervised, the effort spent on supervision is noticeably less than the effort required for execution, enabling the engineering activity to get considerably more complex, while remaining manageable.

9.5.3.2 Improved Process Efficiency

In addition, the efficiency of the overall engineering process is improved by automating selected engineering activities. For example, the time required to evaluate data from a given test session is significantly shortened by the proposed solution, shortening required time and effort for the overall spacecraft test campaign.

9.5.3.3 Improved System Quality

By having an automated execution of selected engineering activities according to predefined rules, it can be ensured that the activity is executed as specified and that no aspects that match the prescribed process are overlooked. This applies to, for example, not overlooking a required test for the satellite, and not overlooking any unexpected events in the conclusion of a test session.

9.5.3.4 Improved Information Gathering and Consolidation Process

The ability to make classifications based on present information improves the overall data consolidation process. For instance, this enables a quick yet exact statement about the overall testing effort required for a system at a very early design stage while only a rough architecture is known. Additionally, the meaning of gathered test metadata is automatically determined, avoiding the extra process of manually evaluating this data and manually drawing conclusions.

9.6 System Engineering Coordination Demonstration Case

This demonstration case considers the relation of engineering activities to their surrounding context. This involves the relation of data generated by these activities to the overarching system design process, the lifecycle consideration of this data, and how system design data relates to involved engineering disciplines.

9.6.1 Existing Challenges in Established Process

Currently, a number of challenges exist in the context of relating engineering data to the context into which it is embedded:

Implicit relation between design data and process artefacts

As explained previously in 2.4.4, the relation of engineering data to the artefacts of the embedding overall design process is merely given implicitly. This implicit relation leads to the necessity of manually collecting all data that is required for the next release of a specific artefact. Consequently, the need for explicit mappings, to enable automated tracking, was formulated (see 5.2.2.1).

Manual management of system lifecycle aspects

The engineering data produced in space system design is influenced significantly by the current position in the development cycle of the system (see 2.4.9). However, this lifecycle dimension to engineering data is currently not reflected by its specification, leading to the fact that the consistency of the overall SM is specified for its final state, i.e. when the design is finished. This leads to a manual management of these time-dependent aspects, where the evolution of the system's design is incrementally checked manually, until the design is complete (see 5.2.2.1).

Implicit management of discipline involvement

Each engineering discipline is involved in numerous aspects of the system to be designed. This involvement can be allocated to *System Elements*, based on specific characteristics. However, currently, this involvement is managed implicitly, not generating any overview of when what discipline has a stake in which system component (see 5.1.1).

9.6.2 SCDMF Application

9.6.2.1 Relating Engineering Data to Process Artefacts

For making a connection between actual engineering data of the system, and the considerably more abstracted process artefacts, a connection of these artefacts and CDM concepts is defined (Table 9.7). For this purpose, the artefacts defined in ECSS-E-ST-10 (ESA, 2009a), describing the general space system engineering process, are modeled and related to concepts of the MBSE CDM.

Table 9.7: Mapping of ECSS-E-ST-10 artefacts to CDM concepts

ECSS-E-ST-10 Artefact	MBSE CDM Concept(s)
Specification Tree	SClass RequirementRepository
Preliminary Technical Requirements Specification	SClass RequirementRepository SClass Requirement
Technical Requirements Specification	SClass RequirementRepository SClass Requirement
Product Tree	SClass ProductTree
Interface Control Document	SClass ElementDefinition SClass FunctionalPort SClass Connector
Test Specification	SClass TestTask SClass TestEnvironment SClass TestSpecification
Test Procedure	SClass TestProcedure SClass TestProcedureStep
Test Report	SClass TestSession SClass TestEvaluation

Using this relation, information of which data is required as input for which process artefact can be made. Consequently, the data can be specifically extracted for review input. For instance, in order to evaluate what data is required for the *Specification Tree*, the according model elements can be queried, returning the following data as core input for the Specification Tree document (Table 9.8):

Table 9.8: MagSat Specification Tree

Requirement Repository Basic Definitions and Assumptions	
	Requirement Repository Units, Models and Constants
	Requirement Repository Error Computation
	Requirement Repository Reference Frames
Requirement Repository Mission Requirements	
	Requirement Repository Constellation
	Requirement Repository System Performance

	Requirement Repository Orbit Requirements
	Requirement Repository Launch Requirements
Requirement Repository General Satellite Design and Interface Requirements	
	Requirement Repository Lifetime, Reliability, Availability and Product Assurance
	Requirement Repository Design and engineering requirements
Requirement Repository Payload Requirements	
	Requirement Repository General Payload Requirements
	Requirement Repository ZEM Interface Requirements
	Requirement Repository HPM Requirements
	Requirement Repository EEA Interface Requirements
	Requirement Repository STR Assembly Requirements
	Requirement Repository GPSR Requirements
	Requirement Repository ACC Requirements
	Requirement Repository LRR Requirements
Requirement Repository Platform Requirements	
	Requirement Repository General Platform Requirements
	Requirement Repository AOCS Requirements
	Requirement Repository Structure Requirements
	Requirement Repository CGPS Requirements
	Requirement Repository TCS Requirements
	Requirement Repository EPS Requirements
	Requirement Repository DHS Requirements
	Requirement Repository TTC Requirements
Requirement Repository Operational Requirements	
	Requirement Repository Mission Phases and System Operational Modes Requirements
	Requirement Repository Operability Requirements
	Requirement Repository Operational Interface Requirements

Requirement Repository Software Design Requirements	
	Requirement Repository General OBSW Requirements
	Requirement Repository On-Board Software Design Requirements
	Requirement Repository On-Board Software Maintainability Requirements
	Requirement Repository On-Board Software Margin
	Requirement Repository On-Board Software Images
Requirement Repository Design and Interface Requirements	
	Requirement Repository General Design and Safety Requirements
	Requirement Repository Mechanical Design and Interface Requirements
	Requirement Repository Thermal Design and Interface Requirements
	Requirement Repository Electrical Design and Interface Requirements
	Requirement Repository Magnetic Design Requirements
	Requirement Repository Charging Design Requirements
Requirement Repository Ground Support Equipment Requirements	
	Requirement Repository General GSE Requirements
	Requirement Repository MGSE and FGSE Requirements
	Requirement Repository EGSE and MDVE Requirements
Requirement Repository AIV Requirements	
	Requirement Repository General AIV Requirements
	Requirement Repository Test Requirements
Requirement Repository On-Ground Data Processing Requirements	
	Requirement Repository Level 1b Processor
	Requirement Repository End-to-End System Simulator

For subsequent specification-related artefacts the returned data gets considerably more extensive, as these also contain the requirements themselves, not only their hierarchical organization.

9.6.2.2 Managing Lifecycle Aspects of the System

For managing lifecycle aspects of system data, the concept of *Temporal Criteria* was introduced on language and CDM levels. For instance, these principles can be applied to the system's *Product Structure* for stating what data is required at what point in the lifecycle, and what data is specifically excluded.

Table 9.9: Lifecycle of System Trees in MBSE CDM

MBSE CDM Concept	MDR	PRR	SRR	PDR	CDR	QR	AR
Product Tree		✓	✓	✓	✓	✓	✓
Configuration Tree	×			✓	✓	✓	✓
Assembly Tree	×	×	×		✓	✓	✓
Shelf	×	×	×	×		✓	✓

As specified in ECSS-E-ST-10, the system's *Product Tree* is initially required by the system's *Preliminary Requirements Review* (PRR). For this purpose, it is marked as required (✓) by the MBSE CDM. However, it is also allowed to specify a *Product Tree* in the very beginning of a project, indicated by a blank field. The *Configuration Tree* has been explicitly excluded for the mission definition phase (×), may optionally be present for the *Preliminary Requirements Review* (PRR) and the *System Requirements Review* (SRR), and is finally required for engineering in the phase towards *Preliminary Design Review* (PDR). The *Assembly Tree* is required by the time of *Critical Design Review* (CDR). The *Shelf*, where elements as built are taken from and integrated into the *Assembly Tree*, may exist for the CDR, and is required at *Qualification Review* (QR), together with the actual information of which elements are integrated into which slot on the actual spacecraft (Table 9.9).

SM consistency checks executed yield different results for each defined project milestone. For instance, a consistency check executed for the PDR will mark a missing *ProductTree* and *ConfigurationTree*, will be indifferent about the *AssemblyTree*, and will mark an already present Shelf as well as *ElementRealizations* in the *Shelf*.

The same principle can be applied to the system's *Electrical Architecture*. For this purpose, the following data validity is defined (Table 9.10):

Table 9.10: Lifecycle of electrical concepts in MBSE CDM

MBSE CDM Concept	MDR	PRR	SRR	PDR	CDR	QR	AR
FunctionalPort	×	×		✓	✓	✓	✓
Connector	×	×	×		✓	✓	✓
Contact	×	×	×	×	✓	✓	✓

For *MDR* and *PRR*, no consideration of electrical aspects is given for any system component, explicitly excluding information from existing in the model. At *PDR*, *FunctionalPorts* are required. *Connectors* may exist at *PDR*, but are positively required as data input for the system's *CDR*. The *Contacts* of *Connectors* are excluded up to *PDR*, but also required for *CDR*.

The principle of *Temporal Criteria* can not only be applied globally, but also to the lifecycle of discipline-specific engineering activities, as is shown with the example of the *Functional Verification* discipline. This discipline has its own sub-milestones during the period leading up to the system's *QR* during which most of the testing takes place (Table 9.11):

Table 9.11: Lifecycle of functional verification concepts in MBSE CDM

MBSE CDM Concept	MDR	PRR	SRR	PDR	CDR	QR			AR
						TRR	PTR	TRB	
Requirements			✓	✓	✓	✓	✓	✓	✓
VerificationTask	×	×	×		✓	✓	✓	✓	✓
TestSpecification	×	×	×		✓	✓	✓	✓	✓
TestProcedure	×	×	×	×		✓	✓	✓	✓
TestImplementation	×	×	×	×		✓	✓	✓	✓
TestSession	×	×	×	×	×		✓	✓	✓
TestEvaluation	×	×	×	×	×	×	✓	✓	✓
TestReport	×	×	×	×	×	×		✓	✓
VerificationCloseout	×	×	×	×	×	×	×		✓

As verification rather gains importance towards the end of a system's design cycle, the initially required verification-owned artefact is the *VerificationTask*, relating requirements to verification activities, to be present for the CDR. The same applies to the *TestSpecification*, giving a first definition of a test. The test is then detailed with the *TestProcedure*, which is then implemented towards the *Test Readiness Review* (TRR). During the *TestSession*, the test is executed and evaluated in the *Post Test Review* (PTR). The formal report is required for the *Test Review Board* (TRB). These three milestones map to the overall QR milestone on system level. The *VerificationCloseout* is required for the *Acceptance Review* (AR).

9.6.2.3 Managing Discipline Involvement

For managing concrete involvement of a discipline in a specific element of the system's design, the MBSE Ontology can be queried. This query allocates disciplines to given elements, based on modeled characteristics of the elements (see 8.5.1.1 for the CDM-part of this activity).

Applying the knowledge about discipline involvement to the MagSat SM with a reasoner yields the results as given in Table 9.12.

Table 9.12: Discipline involvement in selected MagSat System Elements

	MagSat	**STRE**	**OBSW**	**OCT**	**BAT**
Requirements Engineering	•	•	•	•	•
Operational Engineering	•	•	•		
Mass Budget	•	•		•	•
Mechanical Engineering	•	•		•	•
Electrical Engineering	•	•		•	•
Thermal Engineering	•	•			•
Instruments Engineering	•				
Control Engineering	•	•		•	
Software Engineering	•	•	•		
Verification Engineering	•	•	•	•	•

The MagSat itself is relevant for each discipline as it contains elements of relevance to every selected discipline. The *STRE* are scoped by *Requirements Engineering* as the

equipment has defined requirements. It is scoped by *Operational Engineering* because it has behavior associated with it through a *Discrete Model*. It is scoped by the *Mass Budget*, as it has mass values associated with it and is scoped by *Mechanical Engineering* because it is a physical component that has to be accommodated on the spacecraft somewhere. Also, it is of relevance to *Thermal Engineering* as it has *maximum* and *minimum temperatures* that define its boundary conditions. Being a classical integral part of the *AOCS*, the element is scoped by *Control Engineering*. As the *STRE* contains *software*, it is of relevance to *Software Engineering*, and as it has *requirements*, it is also scoped by *Verification Engineering*.

The *OBSW* is scoped by *Requirements Engineering* for obvious reasons, and by *Operational Engineering* as it also comes with *modes*. By definition, the *OBSW* being a kind of software, it is scoped by *Software Engineering*.

The *Orbit Control Thruster* is not scoped by *Operational Engineering* and *Thermal Engineering*, and also not *Software Engineering*, as it contains no software directly. However, being part of MagSat's *AOCS*, it is again scoped by *Control Engineering*.

By using this approach, discipline involvement of a given element can automatically be allocated. Vice versa, for a given discipline, a list of elements of interest to this specific discipline can be provided.

9.6.3 Benefits Resulting from SCDMF Application

Applying these principles leads to a number of benefits:

9.6.3.1 Improved Control of Engineering Process

On the one hand, a better overview of the overall space system engineering process is provided. By supplying a direct mapping of engineering data to its overarching process artefact, the data required for producing the artefact directly becomes evident. By providing a time-dimension to engineering data through the defining CDM, its consistency can not only be checked globally, but per defined project or discipline milestone, allowing a more finely granular consideration of the SM. By inferring discipline involvement for configuration items of the engineering process, an improved overview of what disciplines are stakeholders in what system elements is provided.

9.6.3.2 Improved Scaling of Engineering Activities

By enabling this improved overview and control of the overall engineering process, the process itself increases in scalability. This means that, for instance, it can be

executed requiring fewer resources, or that it can be executed with similar resources but manage a larger and more complex system.

9.7 Analysis of the Evaluation

For concluding on the application of the SCDMF, several points are discussed. First, based on the previous demonstration, the use cases are abstracted and allocated to the technological domain in which they are performed. Second, the requirements defined in 5.1 will be closed out by mapping them to elements of the SCDMF, and by referencing where in the last chapters the requirement is considered. Subsequently, the benefits outlined throughout this chapter will be mapped to the overall business benefits of improved time, cost, and quality.

9.7.1 Engineering Activities vs. Modeling Technologies

The concrete use cases demonstrated for the MagSat spacecraft throughout this chapter can be abstracted to more generic use case types. Based on the performed demonstration, a first idea about suitability can be made, allocating a given use case type to one of the two considered modeling technologies. This allocation states that this type of use case is best suited to be performed in an environment based on the given modeling technology. The results of this allocation are presented in Table 9.13.

Table 9.13: Allocation of engineering activities to modeling domain

Object-Oriented Modeling	Ontological Modeling
System modeling	Project data tailoring
Data consistency checking	System design quality checking
Data lifecycle management	Execution data evaluation
Data process artefact relation	Engineering knowledge formalization
	System engineering activity support
	Engineering heuristics support
	Discipline involvement management

Classical system modeling activities are recommended to be performed on object-oriented modeling technologies. The same is true for checking the general consistency of the data itself, and for data lifecycle management, as both of these activities rely on checking the presence of data, a task that is more of a challenge with ontological models due to their OWA. The mapping of data to process artefacts is also recommended to be performed there.

On the other hand, the activity of project data tailoring is more easily performed in the ontological domain, as the data specification can flexibly be adapted during runtime without requiring additional migration or redeployment steps. Checking the quality of the system design itself is also recommended to be performed on the ontological side, as only there the necessary semantic connections for determining if it is a coherent system design can be made. The same applies to engineering activities that involve the evaluation of the meaning present in a large amount of execution data. Processes that involve formalizing operational knowledge and applying it for supporting a given engineering activity or for applying heuristics to the system's design are also recommended to be performed in the ontological domain for the same reasons. The same is valid for managing discipline involvement throughout the whole system decomposition.

9.7.2 Closeout of Requirements

For formally evaluating whether all requirements are considered, Table 9.14 is provided. It traces all requirements to the SCDMF element through which they are approached and realized, and to the section of Chapter 8 or 9 which demonstrates the application of an SCDMF concept in this context.

Table 9.14: Closeout of requirements on system modeling

REQ	Requirement	Realized through	Demonstrated in
1-1	Availability of explicit mappings between discipline data and process artefacts	SCDML	9.6.2.1
1-2	Availability of required constraints in a conceptual manner	SCDML	8.3.1, 8.3.3
1-3	Ability to specify closed world facts	SCDML	9.3.2.1
1-4	Capability to specify functional dependencies between model concepts	SCDML	9.3.2.2
1-5	Support for multiple explicit element characterization mechanisms	SCDML, OWL 2	8.5.2.2, 9.3.2.3
1-6	Support to define lifecycle aspects on data	SCDML	9.6.2.2
2-1	Availability of an overall process for CDM design	SCDMM	8.3.1
2-2	Availability of a procedure to derive the CDM from engineering data	SCDMM	8.3.1
2-3	Availability of a procedure to ensure exhaustiveness of modeled concepts	SCDMM	8.3.1
2-4	Availability of CDM validation procedures	SCDMM	8.3.1
2-5	Capability for providing project-specific customizations	OWL 2	8.3.2
2-6	Semantic accuracy of implemented CDM identical to specified CDM	SCDML, OWL 2	9.3.2
3-1	Support for product structure definition	MBSE CDM	8.3.1, 9.3.2.1
3-2	Support for requirements definition	MBSE CDM	8.4.1
3-3	Support for operational design definition	MBSE CDM	8.4.2
3-4	Support for system architecture definition	MBSE CDM	8.3.3, 8.3.4
3-5	Support for system verification definition	MBSE CDM	8.3.5
3-6	Support for system property definition	MBSE CDM	8.3.2
3-7	Usage of execution data for system validation	OWL 2	9.5.2
3-8	Existence of a mechanism for capturing and applying operational knowledge	OWL 2	9.4.2, 9.5.2
4-1	Compatibility to MDA and EMF	SCDML	8.5.2

9.7.3 Mapping to Business Benefits

Additionally to the requirements closeout, the benefits resulting from applying concepts from the SCDMF mentioned throughout this chapter are related to generic business benefits. These benefits include the typical benefits of the high-level business view, involving reduced development cost, faster time to market, and improved system quality. Reduced development cost is largely influenced by improving the efficiency and effectivity of the engineering processes contributing to the product. Faster time to market is somewhat similar, foremost being influenced by functions such as automatization that shorten the time needed to generate specific process results. Improved system quality is heavily driven by functions that provide a better overview on the product, and by functions that automatically identify inconsistencies or problems. While all considered improvements somehow relate to each of the three benefits, the most direct influences are marked in Table 9.15.

Table 9.15: Mapping of improvements to business benefits

Benefit	Reduced Development Cost	Faster Time to Market	Improved System Quality
Improved SM application implementation effort	•	•	
Improved utility of the SM application			•
Improved SM quality	•		•
Automatically identified system errors	•	•	•
Better proximity to actual engineering process	•	•	•
Improved formalization of operational knowledge	•		•
Automatic application of operational knowledge	•	•	•
Improved scaling of engineering activities	•	•	•
Improved engineering process efficiency	•	•	
Traceability of engineering activity execution			•
Improved information gathering and consolidation process	•	•	
Improved control of engineering process	•	•	

Some of the mentioned benefits resulting from application of one or several of SCDMF's elements are more focused on improving the overall system engineering process or sub-processes thereof by providing better efficiency or effectivity. Other improvements are more focused on improving the overall quality of the process' end-product, i.e. the system itself.

9.7.4 Conclusion of the Demonstration

The demonstration used a representative example in form of the MagSat to demonstrate the applicability and utility of the proposed approach consisting of SCDML as language, SCDMP as procedure, and MBSE CDM as conceptual model. This demonstration involved engineering activities from all system design phases, starting at basic design considerations and reaching up to system verification. Besides demonstrating the benefit of the approach in terms of reduced development cost, faster time to market, and improved system quality, a closeout of the requirements defined in Chapter 5 was performed.

10 Conclusions

This section reflects on the main research goal of improving the system design process in the space domain. For this purpose, the results will be briefly summarized, followed by a discussion of their impact, a reflection on their representativeness, and an outline of points for future research.

10.1 Result Conclusion

This research made evident that the classical approach of modeling space system engineering data using an implementation driven, object-oriented approach reached its limits. While the approach enables all of the classical functions such as data versioning, data exchange, and data consistency checking, a real exploitation of system design data to evaluate if the model represents a properly designed system is currently not possible.

Introducing knowledge-oriented processes and technologies to engineering a space system provides significant benefits. These come to fruition for both the engineering process used for producing the system, and the end-product itself. Current design approaches in space engineering put significant emphasis on a digital representation of the system, forming the main exchange hub managing the data that is used in discipline- and system-focused engineering activities. The contributions made in this work ensure, among other points, that the System Model is closely aligned to the underlying engineering processes, that it can be directly utilized for the automated identification of design issues, and that test data can automatically be evaluated, providing information regarding the system's quality. Enabling this functionality on the System Model provides a more effective and efficient system design process on the one hand, and on the other hand helps ensure overall system quality.

However, this research also made evident that knowledge-oriented modeling technologies, such as ontologies, cannot stand on their own in this context, but have to be embedded into currently employed system design approaches. This combination of existing technologies with those focused on managing knowledge has the potential to

shift the practice of merely modeling a space system for supporting data exchange, towards facilitating a genuine digital spacecraft design process that relies significantly on an automated, model-centric execution of engineering activities.

10.2 Significance of Results

In order to understand the significance of the contributions made by this work, two aspects need to be considered.

First, by introducing functionality focused on handling and applying knowledge to the domain of space engineering, a number of benefits come to play. Providing a more expressive representation of a system's design opens up more exploitation capabilities of the system design data. For example, it can now be determined from the system's model whether it does or does not represent a properly designed system, and if the design comes with a significant amount of potential problems. Existing engineering activities that have previously involved performing a great number of manual steps can now be automated to a significant degree, evolving from completely manual execution to automated execution with manual result inspection. These freed up resources form a key point for being able to design more complex systems, which require more effort for managing the increased complexity. Having the processes and technologies for formalizing and storing the operational knowledge generated by designing a space system reduces the loss of expertise when personnel move on to other responsibilities inside or outside the engineering organization. All of these aspects contribute to gaining a competitive advantage, by either resulting in reduced cost to produce a system, less time to market, or improved system quality.

Second, introducing the new functionality does not negatively impact currently established model semantics, or modeling technologies. Instead, integration is achieved by augmenting the existing object-oriented approach with the knowledge-oriented approach, retaining both semantics. As a result, existing system representations can be augmented with the proposed approach, and the existing way of producing engineering applications is not affected.

The speed of improvement that is present today in many technological domains, such as automotive engineering, aerospace engineering, or consumer electronics design, is significantly faster than the speed of innovation one or two decades ago. As this speed will likely not decrease, but increase further, a significant amount of pressure is put on engineering organizations to quickly adapt to changes of the market environment, adapt to new technologies, and react to the increased pressure from competitors to drive innovations. In this respect, the benefits enabled by this work, have to be seen not as a benefit that can be exploited, but as a necessary step that has to be taken for

freeing up the resources required for keeping up with the current speed and complexity of technological advancement.

10.3 Representativeness of Results

The MagSat project used as example throughout this work is derived from a typical space system design project. This makes the scenario representative in terms of size, overall complexity, and model complexity for the demonstrated cases. As many of these demonstration cases are based on existing engineering processes, with a re-execution being performed using the newly defined approach, this makes the outlined knowledge-management principles and technologies applicable to what is currently being done in the domain of space system design.

In numerous demonstration cases, engineering activities that are manually performed in the established approach have been made automatically executable using ontological means of modeling. The results of both the manual and the automated activities were compared in order to evaluate the correctness of the newly proposed approach. Where possible, this comparison yielded highly similar results, retaining the outcome of the manual engineering processes. This makes the new approach a viable option to be employed in the engineering of space systems in terms of correctness.

Introducing an additional perspective to a system's model also introduces additional complexity, especially if this perspective relies on a modeling technology and process different from those established. In order to ensure that a real benefit is brought overall to the system's design process, it is important that the work required to manage the additional complexity is less than the resources freed up by exploiting the new functionality offered by the improved model. In order to ensure this, the implementation integrating both the object-oriented and knowledge-oriented aspects of a system has to be realized in a way where it is treated as one unified model, avoiding any manual model management activities.

An important building block in the approach proposed in this work is the collaboration between both domain expert, and modeling expert. While the domain expert has the knowledge to engineer the product, the modeling expert is responsible for formalizing this knowledge, being proficient in designing models. Only by combining expertise from both the engineering and the modeling domain can the proposed approach be fully utilized.

10.4 Points for Further Research

The architecture defined in this work was demonstrated using a pragmatic approach, with a loose coupling of the object-oriented and the knowledge-oriented models, which currently has to be managed manually. In order to industrialize this approach, readying it for use in a productive environment, the activities required for managing the dependencies between both models have to be automated. This work has already been picked up for further pursuit by Hoppe, et al. (2017).

This work provides the basis for shifting currently manually performed engineering activities to a model that can be used for their automated execution. For this purpose, a pre-defined set of engineering activities were focused on, but a large number of engineering activities currently established in space system design remain that were not considered in this work. These still unconsidered engineering activities can also be realized ontologically with significant benefit. This includes, for example, deriving system design maturity, correctness, and completeness by evaluating key system parameters, ensuring resilient exchange of data between engineering tools, and providing functionality such as system design configurators.

Furthermore, the language, procedure and model detailed in this work can be examined regarding their suitability with other engineering domains that are faced with similar challenges. Automotive engineering for example also has a strong interdisciplinary characteristic, with numerous disciplines from different companies producing components for a given car model, making the aspects in this work focusing on interdisciplinary coordination applicable. Domains such as infrastructure engineering could benefit from the parts of this work focused on collecting and applying operational knowledge, enabling support for avoiding mistakes made in the past in future projects.

For the more distant future, an integration of selected aspects from both the ontological and object-oriented domain is conceivable. For example, several concepts considered in SCDML's design and other object-oriented modeling languages could be introduced to OWL 2's successor, providing a range of benefits. This includes support for part-of/containment/aggregation relationships, better support for reasoning with numeric values, or the possibility to explicitly decide between open world or closed world reasoning.

Bibliography

Abele, L., Legat, C., Grimm, S. & Müller, A. W., 2013. Ontology-based Validation of Plant Models. *11th IEEE International Conference on Industrial Informatics (INDIN)*, 29-31 July, pp. 236-241.

Allemang, D. & Hendler, J., 2011. *Semantic Web for the Working Ontologist.* 2nd ed. Waltham: Morgan Kaufmann.

ANSI/X3/SPARC Study Group on Data Base Management Systems, 1975. *Interim Report. FDT, ACM SIGMOD bulletin. Volume 7, No. 2,* New York, NY, USA: ACM.

Aßmann, U., Ebert, J., Walter, T. & Wende, C., 2013. Ontology and Bridging Technologies. In: J. Z. Pan, et al. eds. *Ontology-Driven Software Development.* Berlin: Springer, pp. 179-192.

AUTOSAR, 2016. *AUTOSAR Release 4.3.* [Online]
Available at: http://www.autosar.org/standards/classic-platform/release-43/
[Accessed 13 February 2017].

Baader, F. et al., 2007. *The Description Logic Handbook.* 2nd ed.
Cambridge: Cambridge University Press.

Bakema, G. & Zwart, J. P., 2008. The FCO-IM Analysis Process of Fact Stating Sentences. *Second Libyan International Symposium on Information Systems Modeling and Development*, 19 January.

Boehm, B., 1986. A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes,* 11(4), pp. 14-24.

Bollen, P., 2008. SBVR: A Fact-Oriented OMG Standard. In: Z. Tari, ed. *On the Move to Meaningful Internet Systems: OTM 2008 Workshops.* Berlin: Springer, p. 718–727.

Bone, M. & Cloutier, R., 2010. The Current State of Model Based Systems Engineering: Results from the OMG™ SysML Request for Information 2009. *8th Conference on Systems Engineering Research (CSER 2010)*, 17-19 March, pp. 225-232.

Borst, W. N., 1997. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse,* Twente: University of Twente.

Buede, D. M., 2009. *The Engineering Design of Systems.* 2nd ed.
New Jersey, NJ, USA: John Wiley & Sons.

Chourabi, O., Pollet, Y. & Ahmed, M. B., 2008. *Ontology based knowledge modeling for System Engineering projects.* Marrakech, Morocco, IEEE.

CogNIAM.eu, 2015. *CogNIAM.eu.* [Online]
Available at: http://www.cogniam.eu/

Dassault Systèmes, 2017. *CATIA V5 Portfolio - Dassault Systèmes 3D Software.* [Online]
Available at: https://www.3ds.com/products-services/catia/products/v5/portfolio/
[Accessed 6 May 2017].

de Koning, H. P. et al., 2014. Open Concurrent Design Tool ESA Community Open Source Ready to Go!. *6th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, 8-10 October.

Delicado, B., 2016. The Surprising Path to Greater Creativity: Using systems models in Engineering to improve understanding and problem solving across domains and boundaries in complex projects and modern organisations. *7th International Conference on Systems & Concurrent Engineering for Space Applications (SECESA 2016)*, 5-7 October.

Eisenmann, H. & Cazenave, C., 2014. Evolving a Classic SRDB into an Engineering Database. *6th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, 8-10 October.

Ernadote, D., 2015. An Ontology Mindset for System Engineering. *2015 IEEE International Symposium on Systems Engineering (ISSE)*, 28-30 September.

ESA, 2003. *ECSS-E-70-41A: Space engineering - Ground systems and operations - Telemetry and telecommand packet utilization.* Noordwijk, The Netherlands: ESA.

ESA, 2008a. *ECSS-E-ST-31C: Space engineering - Thermal control general requirements.* Noordwijk, The Netherlands: ESA.

ESA, 2008b. *ECSS-E-ST-32C Rev. 1 - Space engineering - Structural general requirements.* Noordwijk, The Netherlands: ESA.

ESA, 2008c. *ECSS-E-ST-35-01C: Space engineering - Liquid and electric propulsion for spacecraft.* Noordwijk, The Netherlands: ESA.

ESA, 2008d. *ECSS-E-ST-70-31C: Space engineering - Ground systems and operations - Monitoring and control data definition.* Noordwijk, The Netherlands: ESA.

ESA, 2008e. *ECSS-Q-ST-10-04C: Space product assurance - Critical-item control.* Noordwijk, The Netherlands: ESA.

ESA, 2009a. *ECSS-E-ST-10C: Space engineering – System engineering general requirements..* Noordwijk, The Netherlands: ESA.

ESA, 2009b. *ECSS-E-ST-10-06C: Space engineering - Technical requirements specification.* Noordwijk, The Netherlands: ESA.

ESA, 2010. *ECSS-E-TM-10-25: Engineering design model data exchange (CDF).* Noordwijk, The Netherlands: ESA.

ESA, 2011a. *ECSS-E-TM-10-23A: Space engineering - Space system data repository.* Noordiwjk, The Netherlands: ESA.

ESA, 2011b. *ECSS-Q-TM-70-52A: Space product assurance - Kinetic outgassing of materials for space.* Noordwijk, The Netherlands: ESA.

ESA, 2012a. *The Virtual Spacecraft Design Project.* [Online]
Available at: http://vsd.esa.int/
[Accessed 13 February 2017].

ESA, 2012b. *ECSS-S-ST-00-01C: Glossary of terms.* Noordiwjk, The Netherlands: ESA.

ESA, 2012c. *ECSS-E-ST-20-07C Rev. 1: Space engineering - Electromagnetic compatibility.* Noordwijk, The Netherlands: ESA.

ESA, 2012d. *ECSS-U-AS-10C: Space sustainability - Space debris.* Noordwijk, The Netherlands: ESA.

ESA, 2013a. *EGS-CC - European Ground Systems - Common Core.* [Online]
Available at: http://www.egscc.esa.int/

ESA, 2013b. *Collaboration website of the European Cooperation for Space Standardization.* [Online]
Available at: http://ecss.nl/
[Accessed 1 June 2016].

ESA, 2014. *Open Concurrent Design Tool (OCDT) Community Portal.* [Online]
Available at: https://ocdt.esa.int/
[Accessed 13 February 2017].

Evans, A. & Kent, S., 2003. Core Meta-Modelling Semantics of UML: The pUML Approach. In: R. France & B. Rumpe, eds. *UML'99 — The Unified Modeling Language, LNCS Volume 1723.* Berlin: Springer, pp. 140-155.

FBM WG, 2014. *Fact-Based Modelling Metamodel (version WD08).* [Online]
Available at: http://www.factbasedmodeling.org/Data/Sites/1/media/FBM1002WD08.pdf
[Accessed 13 February 2017].

Feldmann, S. et al., 2015. Towards Effective Management of Inconsistencies in Model-Based Engineering of Automated Production Systems. *2015 IFAC Symposium on Information Control in Manufacturing (INCOM 2015)*, 11-13 May, pp. 916-923.

Fernández, M., Gómez-Pérez, A. & Juristo, N., 1997. METHONTOLOGY: From Ontological Art Towards Ontological Engineering, AAAI Technical Report SS-97-06. *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*, 24-26 March, pp. 33-40.

Fischer, P. M., Eisenmann, H. & Fuchs, J., 2014. Functional Verification by Simulation based on Preliminary System Design Data. *6th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, 8-10 October.

Forsberg, K. & Mooz, H., 1991. The Relationship of System Engineering to the Project Cycle. *Proceedings of the First Annual Symposium of National Council on System Engineering*, October, pp. 57-65.

Frenzel, C., 2010. *Mooop – A Generic Integration of Object-Oriented and Ontological Models (TR 2010-14),* Augsburg: Universität Augsburg, Institut für Informatik.

Friedenthal, S., Moore, A. & Steiner, R., 2008. *A Practical Guide to SysML.* Burlington, United States: Morgan Kaufmann.

Friedenthal, S. & Sampson, M., 2014. *INCOSE International Workshop 2014, MBSE Workshop Introduction.* [Online]
Available at: http://www.omgwiki.org/MBSE/doku.php?id=mbse:incose_mbse_iw_2014
[Accessed 29 November 2016].

Gašević, D., Djurić, D. & Devedžić, V., 2009. *Model Driven Engineering and Ontology Development.* 2nd ed. Berlin: Springer.

Gómez-Pérez, A., Fernández-Lopez, M. & Corcho, O., 2004. *Ontological Engineering.* London: Springer.

Graves, H., 2007. Ontology Engineering for Product Development. *OWL: Experiences and Directions, 3rd International Workshop (OWLED 2007)*, 6-7 June.

Graves, H., 2009. Integrating SysML and OWL. *Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009)*.

Graves, H., 2012. Integrating Reasoning with SysML. *INCOSE International Symposium,* 22(1), p. 2228–2242.

Graves, H. & Horrocks, I., 2008. Application of OWL 1.1 to Systems Engineering. *OWL: Experiences and Directions, 5th International Workshop (OWLED 2008)*, 26-27 October.

Gruber, T. R., 1993. *Toward Principles for the Design of Ontologies, Technical Report KSL-93-04,* Stanford, CA, USA: Stanford Knowledge Systems Laboratory.

Halpin, T., 2009. Object-Role Modeling. In: L. Liu & M. T. Özsu, eds. *Encyclopedia of Database Systems.* New York: Springer, pp. 1941-1946.

Halpin, T. & Morgan, T., 2008. *Information Modeling and Relational Databases.* 2nd ed. Burlington: Morgan Kaufmann.

Halpin, T. & Wijbenga, J. P., 2010. FORML 2. In: I. Bider, et al. eds. *Enterprise, Business-Process and Information Systems Modeling.* Berlin: Springer, pp. 247-260.

Hendler, J., 2001. Agents and the Semantic Web. *IEEE Intelligent Systems,* 16(2), pp. 30-36.

Hendler, J., Berners-Lee, T. & Miller, E., 2002. Integrating Applications on the Semantic Web. *Journal of the Institute of Electrical Engineers of Japan,* 122(10), pp. 676-680.

Hennig, C., Baldesarra, M. & Eisenmann, H., 2016. An engineering data management infrastructure covering mission analysis up to system qualification. *7th International Conference on Systems & Concurrent Engineering for Space Applications (SECESA 2016)*, 5-7 October.

Hennig, C. & Eisenmann, H., 2014. Applying Selected Knowledge Management Technologies and Principles for Enabling Model-based Management of Engineering Data in MBSE. *6th International Workshop on Systems and Concurrent Engineering for Space Applications (SECESA)*, 8-10 October.

Hennig, C. & Eisenmann, H., 2016. Concluding on the Data Modelling Technologies Required for Realizing the Vision of ECSS-E-TM-10-23. *7th International Conference on Systems & Concurrent Engineering for Space Applications (SECESA 2016)*, 5-7 October.

Hennig, C., Eisenmann, H., Viehl, A. & Bringmann, O., 2015. On Languages for Conceptual Data Modeling in Multi-Disciplinary Space Systems Engineering. *3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015)*, 9-11 February.

Hennig, C., Eisenmann, H., Viehl, A. & Bringmann, O., 2016b. A Methodology for Deriving Conceptual Data Models from Systems Engineering Artefacts. *4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, 19-21 February.

Hennig, C. et al., 2016a. SCDML: A Language for Conceptual Data Modeling in Model-Based Systems Engineering. *4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, 19-21 February.

Hennig, C., Viehl, A., Kämpgen, B. & Eisenmann, H., 2016c. Ontology-Based Design of Space Systems. *15th International Semantic Web Conference (ISWC 2016)*, 17-21 October.

Hoppe, T., Eisenmann, H., Viehl, A. & Bringmann, O., 2017. SEMF – The Semantic Engineering Modeling Framework. *5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, 19-21 February.

Hughes, T. P., 1998. *Rescuing Prometheus.* New York City, NY, USA: Vintage Books.

IBM, 2017. *IBM - Rational DOORS.* [Online]
Available at: http://www-03.ibm.com/software/products/de/ratidoor
[Accessed 6 May 2017].

IEEE, 2009. *IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008.* [Online]
Available at: http://ieeexplore.ieee.org/servlet/opac?punumber=4772738
[Accessed 14 February 2017].

INCOSE, 2014. *Systems Engineering Vision 2025.* [Online]
Available at: http://www.incose.org/docs/default-source/aboutse/se-vision-2025.pdf
[Accessed 4 December 2016].

INCOSE, 2015. *INCOSE Systems Engineering Handbook.* 4th ed. New Jersey, NJ, USA: John Wiley & Sons.

INCOSE, 2016. *What is Systems Engineering.* [Online]
Available at: http://www.incose.org/AboutSE/WhatIsSE
[Accessed 27 April 2016].

ISO, 1987. *Technical Report 9007: Information processing systems - Concepts and terminology for the conceptual schema and the information base,* Geneva, Switzerland: ISO.

ISO, 2015a. *ISO/TS 15926-11: Industrial automation systems and integration - Integration of life-cycle data for process plants including oil and gas production facilities - Part 11: Methodology for simplified industrial usage of reference data.* ISO: Geneva, Switzerland.

ISO, 2015b. *ISO/IEC/IEEE 15288: Systems and software engineering - System life cycle processes.* ISO: Geneva, Switzerland.

Jenkins, S. & Rouquette, N., 2012. Semantically-Rigorous Systems Engineering Using SysML and OWL. *5th International Workshop on Systems & Concurrent Engineering for Space Applications (SECESA 2012)*, 17-19 October.

Jet Propulsion Laboratory, 2016. *BinTray - JPL-IMCE / gov.nasa.jpl.imce.ontologies.public / 1.0.1.* [Online]
Available at: http://bintray.com/jpl-imce/gov.nasa.jpl.imce/gov.nasa.jpl.imce.ontologies.public/1.0.1
[Accessed 24 February 2017].

Kalfoglou, Y., 2000. Exploring Ontologies. In: S. K. Chang, ed. *Handbook of Software Engineering & Knowledge Engineering, Volume 1: Fundamentals.* New Jersey: World Scientific Publishing, pp. 863-887.

Kiko, K. & Atkinson, C., 2008. *A Detailed Comparison of UML and OWL (TR-2008-004),* Mannheim: University of Mannheim, Fakultät für Mathematik und Informatik, Lehrstuhl für Softwaretechnik.

Klüwer, J. W., Skjæveland, M. G. & Valen-Sendstad, M., 2008. *ISO 15926 templates and the Semantic Web.* Houston, TX, USA, W3C.

Kogalovsky, M. R. & Kalinichenko, L. A., 2009. Conceptual and Ontological Modeling in Information Systems. *Programming and Computer Software,* 35(5), pp. 241-256.

Lemmens, I., Nijssen, M. & Nijssen, S., 2007. A NIAM2007 Conceptual Analysis of the ISO and OMG MOF Four Layer Metadata Architectures. In: R. Meersman, Z. Tari & P. Herrero, eds. *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops.* Berlin: Springer, pp. 613-623.

Leung, C. M. R. & Nijssen, G. M., 1998. Relational database design using the NIAM Conceptual Schema. *Information Systems,* 13(2), pp. 219-227.

MathWorks, 2017. *MATLAB - MathWorks.* [Online]
Available at: https://www.mathworks.com/products/matlab.html
[Accessed 6 May 2017].

Mehdi, A. & Wissmann, J., 2013. EQuIKa System: Supporting OWL Applications with Local Closed World Assumption. *GI-Jahrestagung*, Band 220 of LNI, pp. 1943-1948.

Microsoft, 2017. *FORMULA - Modeling Foundations.* [Online]
Available at: http://research.microsoft.com/formula
[Accessed 26 February 2017].

Modeliosoft, 2017. *Modelio Open Source - UML and BPMN free modeling tool.* [Online]
Available at: https://www.modelio.org/
[Accessed 6 February 2017].

Musen, M. A., 2015. The Protégé Project: A Look Back and a Look Forward.
AI Matters, 1(4), pp. 4-12.

NASA, 2007. *NASA Systems Engineering Handbook (NASA-SP-2007-6105) RevI.* [Online]
Available at: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080008301.pdf
[Accessed 14 February 2017].

Neema, S., Scott, J. & Bapty, T., 2015. *CyPhyML Language in the META Toolchain, TR ISIS-15-104,* Nashville, TN, USA: ISIS, Vanderbilt University.

Nijssen, G. M., 1978. *A Framework for Discussion in ISO/TC97/SC5/WG3,*
Geneva, Switzerland: ISO.

No Magic, 2016. *MagicDraw.* [Online]
Available at: http://www.nomagic.com/products/magicdraw.html
[Accessed 31 May 2016].

Obeo, 2016. *Obeo Designer: Domain Specific Modeling for Software Architects.* [Online]
Available at: http://www.obeodesigner.com/
[Accessed 31 May 2016].

Oberle, D., 2014. How Ontologies Benefit Enterprise Applications. *Semantic Web,* 5(6),
pp. 473-491.

OMG, 2013. *MOF Support for Semantic Structures (SMOF), Version 1.0.* [Online]
Available at: http://www.omg.org/spec/SMOF/1.0/
[Accessed 14 February 2017].

OMG, 2014a. *MDA Guide revision 2.0.* [Online]
Available at: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01
[Accessed 2 February 2017].

OMG, 2014b. *Object Constraint Language, Version 2.4.* [Online]
Available at: http://www.omg.org/spec/OCL/2.4/
[Accessed 14 February 2017].

OMG, 2014c. *Ontology Definition Metamodel (ODM), Version 1.1.* [Online]
Available at: http://www.omg.org/spec/ODM/1.1/
[Accessed 14 February 2017].

OMG, 2015a. *Meta Object Facility (MOF) Version 2.5.* [Online]
Available at: http://www.omg.org/spec/MOF/2.5/
[Accessed 18 December 2016].

OMG, 2015b. *OMG Unified Modeling Language (OMG UML) Version 2.5.* [Online]
Available at: http://www.omg.org/spec/UML/2.5/
[Accessed 14 February 2017].

OMG, 2015c. *OMG Systems Modeling Language (OMG SysML) Version 1.4.* [Online]
Available at: http://www.omg.org/spec/SysML/1.4/
[Accessed 14 February 2017].

OMG, 2015d. *Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.3.* [Online]
Available at: http://www.omg.org/spec/SBVR/1.3/
[Accessed 14 February 2017].

OMG, 2016. *ReqIF 1.2.* [Online]
Available at: http://www.omg.org/spec/ReqIF/1.2/
[Accessed 18 June 2017].

PolarSys, 2016. *Topcased | PolarSys.* [Online]
Available at: https://www.polarsys.org/topcased
[Accessed 1 December 2016].

Puleston, C., Parsia, B., Cunningham, J. & Rector, A., 2008. Integrating Object-Oriented and Ontological Representations: A Case Study in Java and OWL. In: A. Sheth, et al. eds. *The Semantic Web - ISWC 2008, LNCS Volume 5318.* Berlin: Springer, pp. 130-145.

Rahmani, T., Oberle, D. & Dahms, M., 2010. An Adjustable Transformation from OWL to Ecore. *13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010)*, 3-8 October, pp. 243-257.

RHEA System, 2015. *CDP - Concurrent Design Platform.* [Online]
Available at: http://www.rheagroup.com/products/cdp/
[Accessed 4 December 2015].

Rouquette, N., 2010. OWL Ontologies and SysML Profiles: Knowledge Representation and Modeling. *2nd Biannual Symposium on Eclipse Open Source Software & OMG Open Specifications*, 22 June.

Royce, W., 1970. Managing the Development of Large Software Systems.
Proceedings of IEEE WESCON, 26 August, pp. 1-9.

Sarder, B. & Ferreira, S., 2007. Developing Systems Engineering Ontologies. *2007 IEEE International Conference on System of Systems Engineering (SoSE 2007)*, 16-18 April .

Silva Parreiras, F., 2011. *Marrying Model-Driven Engineering and Ontology Technologies: The TwoUse Approach.* Koblenz: Universität Koblenz-Landau.

Stachowiak, H., 1973. *Allgemeine Modelltheorie.* Wien: Springer.

Studer, R., Benjamins, V. R. & Fensel, D., 1998. Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering,* Volume 25, pp. 161-197.

Suárez-Figueroa, M. C., 2010. *NeOn Methodology for Building Ontology Networks,* Madrid: Universidad Politécnica de Madrid.

Sure, Y., Staab, S. & Studer, R., 2004. On-To-Knowledge Methodology (OTKM).
In: S. Staab & R. Studer, eds. *Handbook on Ontologies.* Berlin: Springer, pp. 117-132.

Sztipanovits, J. et al., 2014. OpenMETA: A Model- and Component-Based Design Tool Chain for Cyber-Physical Systems. In: S. Bensalem, Y. Lakhneck & A. Legay, eds. *From Programs to Systems. The Systems perspective in Computing, LNCS No. 8415.* Berlin: Springer, pp. 235-248.

Thakker, D., Dimitrova, V., Cohn, A. G. & Valdes, J., 2015. PADTUN - Using Semantic Technologies in Tunnel Diagnosis and Maintenance. In: F. Gandon, et al. eds. *The Semantic Web. Latest Advances and New Domains, LNCS Volume 9088.* Berlin: Springer, pp. 683-698.

Thales, 2015. *An Introduction to Arcadia.* [Online]
Available at:
http://download.polarsys.org/capella/publis/An_Introduction_to_Arcadia_20150115.pdf
[Accessed 1 June 2016].

The Eclipse Foundation, 2014. *Capella.* [Online]
Available at: https://www.polarsys.org/capella/
[Accessed 4 December 2015].

The Eclipse Foundation, 2015. *Papyrus.* [Online]
Available at: https://eclipse.org/papyrus/
[Accessed 31 May 2016].

The Eclipse Foundation, 2016a. *Eclipse desktop & web IDEs.* [Online]
Available at: https://eclipse.org/ide/
[Accessed 15 February 2017].

The Eclipse Foundation, 2016b. *Eclipse Modeling Project.* [Online]
Available at: http://www.eclipse.org/modeling/emf/
[Accessed 15 February 2017].

The Eclipse Foundation, 2016c. *org.eclipse.emf.ecore (EMF Documentation).* [Online]
Available at:
http://download.eclipse.org/modeling/emf/emf/javadoc/2.11/org/eclipse/emf/ecore/package-summary.html
[Accessed 15 February 2017].

The Eclipse Foundation, 2017a. *Eclipse Modeling Tools | Downloads | Mars Service Release 2.* [Online]
Available at: http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/mars2
[Accessed 25 March 2017].

The Eclipse Foundation, 2017b. *Sirius - The easiest way to get your own Modeling Tool.* [Online]
Available at: http://www.eclipse.org/sirius/
[Accessed 9 January 2017].

Valera, S., 2014. The Fact Based Modelling Unifying System FAMOUS. *International Workshop on Fact-Oriented Modeling (ORM 2014)*, 29-31 October .

Van Renssen, A. S. H. P., 2005. *Gellish - A Generic Extensible Ontological Language,* Delft: Technische Universiteit Delft.

Van Ruijven, L. C., 2012. Ontology for Systems Engineering. *6th UKSim/AMSS European Symposium on Computer Modeling and Simulation (EMS)*, 14-16 November, pp. 371-376.

Van Ruijven, L. C., 2015. Ontology for Systems Engineering as a base for MBSE. *25th Annual INCOSE International Symposium (IS2015)*, 13-16 July.

W3C, 2004. *SWRL: A Semantic Web Rule Language.* [Online]
Available at: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/
[Accessed 31 May 2016].

W3C, 2006. *A Semantic Web Primer for Object-Oriented Software Developers.* [Online]
Available at: http://www.w3.org/2001/sw/BestPractices/SE/ODSD/
[Accessed 7 September 2016].

W3C, 2008. *SPARQL Query Language for RDF.* [Online]
Available at: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
[Accessed 31 May 2016].

W3C, 2012a. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition).* [Online]
Available at: http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/
[Accessed 31 May 2016].

W3C, 2012b. *OWL 2 Web Ontology Language Primer (Second Edition).* [Online]
Available at: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/
[Accessed 31 May 2016].

W3C, 2012c. *OWL 2 Web Ontology Language Manchester Syntax (Second Edition).* [Online]
Available at: http://www.w3.org/TR/owl2-manchester-syntax/
[Accessed 15 February 2017].

W3C, 2014a. *RDF 1.1 Concepts and Abstract Syntax.* [Online]
Available at: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
[Accessed 31 May 2016].

W3C, 2014b. *RDF Schema 1.1.* [Online]
Available at: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/
[Accessed 31 May 2016].

W3C, 2017. *Time Ontology in OWL (W3C Working Draft 02 February 2017).* [Online]
Available at: http://www.w3.org/TR/2017/WD-owl-time-20170202/
[Accessed 3 March 2017].

Wagner, C., 2014. *Model-Driven Software Migration: A Methodology.* Wiesbaden: Springer.

Wende, C. et al., 2013. Ontology Reasoning for Consistency-Preserving Structural Modeling. In: J. Z. Pan, et al. eds. *Ontology-Driven Software Development.* Berlin: Springer, pp. 193-218.

Yamaura, S., Seiko, S., Hirako, K. & Hiramatsu, T., 2016. A model-based framework to ensure traceability from operational requirements to system design for an on-demand small SAR satellite system. *7th International Conference on Systems & Concurrent Engineering for Space Applications (SECESA 2016)*, 5-7 October.

Index

D

E

F

N

O

P

Q

R

S

T

U

V

W

X

Digital models of complex systems, such as aircraft, spacecraft, or infrastructure systems, are becoming increasingly important. While currently employed technologies allow modeling these systems and managing the data produced during their design, these technologies do not allow deriving knowledge about the modeled systems, including whether they actually represent correct systems in their context.

This work approaches this issue by providing a language, a methodology, and a conceptual data model to represent space systems, and to examine the domain semantics of the modeled engineering data. This enables activities such as the automated identification of critical parts of the system's design, inferring knowledge about the system's design from collected test data, and the identification of single points of failure.

ISBN 978-3-7315-0720-8

9 783731 507208 >

Gedruckt auf FSC-zertifiziertem Papier