ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Διδακτορική Διατριβή

Μοντελο-κεντρική Σχεδίαση Συστημάτων με επίγνωση της Παρεχόμενης Ποιότητας Υπηρεσίας βασισμένη στη Γλώσσα Μοντελοποίησης Συστημάτων

Χρήστος ΚΟΤΡΩΝΗΣ

Αθήνα, 2021

HAROKOPIO UNIVERSITY OF ATHENS, GREECE
SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS

DOCTORAL DISSERTATION

QoS-Aware Model-Based Systems Design Using Systems Modeling Language

---

Christos KOTRONIS

Athens, 2021

HAROKOPIO UNIVERSITY OF ATHENS, GREECE
SCHOOL OF DIGITAL TECHNOLOGY
DEPARTMENT OF INFORMATICS AND TELEMATICS

DOCTORAL DISSERTATION

QoS-Aware Model-Based Systems Design Using Systems Modeling Language

Author:
Christos KOTRONIS

THIS DOCTORAL DISSERTATION WAS EXAMINED BY THE FOLLOWING COMMITTEE:

Mara Nikolaidou
Supervisor
Professor
Informatics & Telematics
Harokopio University of Athens

Aphrodite Tsalgatidou
Supervising Committee
Associate Professor
Informatics & Telecommunications
National & Kapodistrian University of Athens

George Dimitrakopoulos
Associate Professor
Informatics & Telematics
Harokopio University of Athens

Constantine Vasilakis
Professor
Informatics & Telecommunications
University of Peloponnese

Dimosthenis Anagnostopoulos
Supervising Committee
Professor
Informatics & Telematics
Harokopio University of Athens

Christos Michalakelis
Associate Professor
Informatics & Telematics
Harokopio University of Athens

Cleopatra Bardaki
Assistant Professor
Informatics & Telematics
Harokopio University of Athens

Examination date: 20 December 2021

H.F.R.I.
Hellenic Foundation for
Research & Innovation

GENERAL SECRETARIAT FOR
RESEARCH AND TECHNOLOGY

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. no. 1526).

The acceptance of the Doctoral Dissertation from the Department of Informatics and Telematics of Harokopio University of Athens does not imply the acceptance of the author's point of view.

Declaration of Authorship

I, Christos KOTRONIS, solemnly declare that this dissertation titled,
"QoS-Aware Model-Based Systems Design Using Systems Modeling Language", and the work presented in it are my own. I confirm that:

- I am the owner of the copyrights of this original work and this work does not defame any person or offend the copyrights of others.
- I accept that the Library of Harokopio University of Athens can change the contents of this work, deliver this work in electronic format through the Institutional Repository, copy this work using any format or media, and keep more than one copies for maintainability or security reasons.
- This work was done wholly or mainly while in candidature for a research degree at Harokopio University of Athens.
- Where any part of this dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

In memory of my father, Panagiotis.

Acknowledgements

First and foremost I am extremely grateful to my supervisor Prof. Maria Nikolaidou for her invaluable advice, continuous support, patience, and motivation during my PhD study. Her guidance, immense knowledge, and plentiful experience have encouraged me and helped me during the time of my academic research, under-graduate and post-graduate studies, and in my daily life.

I would also like to thank Prof. Dimosthenis Anagnostopoulos, member of my supervising committee, for his insightful comments and encouragement. Moreover, I would like to thank Assoc. Prof. Aphrodite Tsalgatidou, member of my supervising committee, for her helpful comments and suggestions. It is their kind help and support that helped me during the stages of my research and the completion of this dissertation.

The cooperation with Dr. Anargyros Tsadimas, Dr. George-Dimitrios Kapos and Dr. Vasileios Dalakas was excellent. I would like to thank them for their mentorship and valuable contribution to our common research effort. I am grateful for all the technical discussions and support, our efficient and fruitful cooperation, the exchanges of ideas, and research endeavors during all those years. I would also like to thank Loretta Mitsi, Fotini Daneli and Eleni Kalampaliki for the great administrative support they provided, and their help and support throughout my academic life.

I express my sincere gratitude to the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Innovation (GSRI) for supporting my research, under the three-year HFRI PhD Fellowship grant (GA. no. 1526).

Many thanks go to my friend and colleague Ioannis Routis, for his support, the many interesting discussions and exchanges of ideas, and the beautiful working environment and time spent together when we shared a common office. I am grateful for the efficient cooperation we had in our research endeavors. Also, I would like to thank Assoc. Prof. George Dimitrakopoulos, Prof. Abbes Amira, Dr. Faycal Bensaali, Elena Politis, and Hamza Djelouat for the excellent cooperation in our Qatar National Research Fund EMBIoT project. They were really helpful and influential in shaping this dissertation.

My gratitude extends to my friends Vasileios Korlos and Dimitrios Karagkounis who have always been a major source of support, and provided happy distractions to rest my mind outside of my research. My appreciation goes to Anastasia-Dimitra-Danai Lipitakis for constantly listening to me rant and talk things out, for cracking jokes when things became too serious and difficult for me, and for providing stimulating discussions about science, research, and life. Their support and encouragement, especially during the last years of this effort, helped me prioritize this work in order for it to be successfully completed.

Last but not least, I would like to send my warmest thanks to my family, namely my father Panagiotis Kotronis, my mother Eirini Krokida, and my brother Dr. Vasileios Kotronis, for their unconditional love, help, and support all these years, and for giving me the extra strength and motivation to pursue my goals to the very end. Without them believing in me, I would not have made it to the end. This dissertation is dedicated to them.

Contents

List of Figures

List of Tables

List of Acronyms

AADL	Architecture Analysis and Design Language
ABC	Activity-Based Costing
ACIMS	Arizona Center for Integrative Modeling and Simulation
ACT	Activity Diagram
AF	Architecture Frameworks
AI	Artificial Intelligence
Ar	Actual Resources
AUTOSAR	Automotive Open System Architecture
BDD	Block Definition Diagram
BEC	Building Energy Conservation
BPMN	Business Process Model & Notation
CapEx	Capital Expenditures
CD	Class Diagram
CEN	European Committee for Standardization
CHS	Cyber-Human Systems
Cn	Connectivity
COCOMO	Constructive Cost Model
COSYSMO	Constructive Systems Engineering Cost Model
CPHS	Cyber-Physical Human Systems
CPS	Cyber-Physical Systems
CSD	Composite Structure Diagram
CSM	Cameo Systems Modeler
CST	Cameo Simulation Toolkit
CSV	Comma-Separated Values
Ct	Constraints
CTL	Computational Tree Logic
Dc	Dictionary
DES	Discrete-Event Simulation
DEVS	Discrete Event System Specification
DMM	Domain Meta-Model
DoDAF	Department of Defense Architecture Framework
DSM	Decision Support Model
DSR	Design Science Research
DSRM	Design Science Research Methodology
DSS	Decision Support System
ECG	Electrocardiogram
ECMN	Equipment Coupling Modeling Notation

EMOF	Essential Meta-Object Facility
FSM	Finite State Machines
FURPS	Functionality, Usability, Reliability, Performance, and Supportability
HiLCPS	Human-in-the-Loop Cyber-Physical Systems
HSCD	Human Service Capability Description
HSR	High Speed Rail
IBD	Internal Block Diagram
IBM	International Business Machines
ICS	Integrated Control Systems
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
If	Information
INCOSE	International Council on Systems Engineering
IoMT	Internet of Medical Things
IoT	Internet of Things
Is	Interaction Scenarios
ITU	International Telecommunication Union
LoS	Level of Service
LTL	Linear Temporal Logic
MARTE	Modeling and Analysis of Real-Time and Embedded systems
MATSim	Multi-Agent Transport Simulation Toolkit
MBSA	Model Based Safety Analysis
MBSD	Model-Based Systems Design
MBSE	Model-Based Systems Engineering
MD	MagicDraw
Md	Metadata
MDA	Model-Driven Architecture
MoE	Measures of Effectiveness
MOF	Meta-Object Facility
NFR	Non-Functional Requirements
OCL	Object Constraint Language
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OOSEM	Object-Oriented System Engineering Method
Op	Operational
OpEx	Operating Expenses
PD	Parametric Diagram
PDA	Push-Down Automata
Pj	Projects
PKG	Package Diagram
Pm	Parameters
Pr	Personnel
Pr	Processes
Prolog	Programming in Logic

PTC	Parametric Technology Corporation
QoS	Quality of Service
QVT	Query/View/Transformation
RD	Requirement Diagram
REMS	Remote Elderly Monitoring System
Rm	Roadmap
Rq	Requirement
Rs	Resources
RSS	Railway Simulation System
RTS	Railway Transportation Systems
RTS UAV ISR	Remote Targeting System for Unmanned Aerial Vehicle Intelligence, Surveillance and Reconnaissance
SAE	Society of Automotive Engineers
Sc	Security
SD	Sequence Diagram
SD	Systems Design
Sd	Standards
SE	Systems Engineering
SICS	Swedish Institute of Computer Science
SIMAN	Simulation Management
Sm-Ov	Summary & Overview
SoS	Systems of Systems
Sr	Structure
St	States
St	Strategic
STM	State Machine Diagram
Sv	Services
SysML	Systems Modeling Language
SYSMOD	Systems Modeling Toolbox
TCA	Traditional Cost Accounting
TCO	Total Cost of Ownership
TCQSM	Transit Capacity and Quality of Service Manual
TOGAF	The Open Group Architecture Framework
Tr	Traceability
TRANSIMS	TRansportation ANalysis and SIMulation System
Tx	Taxonomy
UAF	Unified Architecture Framework
UAFP	UAF Profile
UC	Use Case Diagram
UML	Unified Modeling Language
XML	eXtensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Abstract

Systems Engineering (SE) is a methodological approach for the design, implementation, technical management, operation, and potential improvement of systems. Systems Design (SD) is one of the core activities of this process. Model-Based Systems Design (MBSD) is a prominent approach for the efficient design of systems, and in particular complex Systems of Systems (SoS). MBSD advocates the creation of conceptual system models that provide unified, visual representations of a system. All stakeholders should be able to understand these models at different levels of complexity (e.g., ranging from a single unit to an entire SoS). MBSD can be implemented using the Systems Modeling Language (SysML), an Object Management Group (OMG) standard. SysML enables the description of a wide range of SoS and can be applied in different design activities and system domains, being accepted by both academia and industry.

Among several parameters involved in SD, Quality of Service (QoS) is a critical factor that constrains design decisions. QoS reflects the operating levels of a system under certain conditions that affect the quality of the provided functionality. Despite it being a clear indicator of the quality of services that the end-users of a system are expected to enjoy, there is no complete framework that integrates QoS into design efficiently using MBSD and SysML. Moreover, the current version of SysML does not directly address QoS concepts. However, the primitive notion of SysML requirements can be used and extended to describe and manage QoS. The focus of extended QoS requirements is to capture what the system provides to its users, service-wise, rather than to encapsulate system behavior. To evaluate such requirements one has to assess system outcomes, via system analysis, to verify whether the latter satisfy the users (or not). Satisfying certain QoS requirements requires a designer to decide upon different key aspects of the system, such as its configuration. Serving QoS is further tied to the costs needed to acquire/operate the system.

The aim of this work is to explore QoS using MBSD and SysML. We build upon the frameworks of Friedenthal, Unified Architecture Framework (UAF), and Systems Modeling Toolbox (SYSMOD), and we extend them to manage QoS. In particular, QoS requirements are specified and integrated into SysML models. Analysis models (based on simulations or mathematical solvers) are employed to evaluate these requirements. Moreover, the system model encapsulates a decision-support model, for decision-making, as well as a cost model, for assessing costs.

We apply our QoS-aware SysML-based MBSD framework to two domains, i.e. Railway Transportation Systems (RTS) and Cyber-Physical Human Systems (CPHS). In RTS, we focus on the operation of such systems based on the specification and evaluation of the quality of the comfort of commuters while they wait at stations or trains. This quality is quantified using Level of Service (LoS), a quantitative index for the overall performance of a service from the viewpoint of the service provider. In parallel, the operating costs, depending on the achieved LoS, are analyzed. In CPHS, we focus on engaging human participants in design, considering their concerns and quality requirements. The participants dive into system details via specific views (e.g., human and/or system view) and modeling constructs, depicting key performance indicators. Their concerns are satisfied with an appropriate system design through a process that continuously develops and maintains the bonds between the human participant and the system. Therefore, humans are expected to establish deep awareness of the CPHS and increased willingness to participate when it is rolled out. These case studies demonstrate the generality and applicability of our framework.

SUBJECT AREA: Systems Engineering
KEYWORDS: Model-Based Systems Design, Systems Modeling Language, Quality of Service, Requirements Verification

Περίληψη

Η Μηχανική Συστημάτων (Systems Engineering (SE)) αποτελεί μία μεθοδολογική προσέγγιση για τη σχεδίαση, την υλοποίηση, την τεχνική διαχείριση, τη λειτουργία, και την πιθανή βελτίωση συστημάτων. Συγκεκριμένα, η Σχεδίαση Συστημάτων (Systems Design (SD)) είναι μία από τις βασικές δραστηριότητες αυτής της διαδικασίας. Η Μοντελο-κεντρική Σχεδίαση Συστημάτων (Model-Based Systems Design (MBSD)) είναι η προτιμητέα προσέγγιση για την αποτελεσματική σχεδίαση συστημάτων και ιδίως σύνθετων συστημάτων αποτελούμενων από υποσυστήματα (Systems of Systems (SoS)). Το MBSD υποστηρίζει τη δημιουργία εννοιολογικών μοντέλων συστημάτων τα οποία παρέχουν ενοποιημένες, οπτικές αναπαραστάσεις ενός συστήματος. Όλοι οι ενδιαφερόμενοι (stakeholders) πρέπει να είναι σε θέση να κατανοήσουν αυτά τα μοντέλα σε διαφορετικά επίπεδα πολυπλοκότητας (παραδείγματος χάριν, επίπεδα που κυμαίνονται από μια διακριτή μονάδα του συστήματος μέχρι και ένα ολόκληρο σύστημα). Επίσης, το MBSD μπορεί να υλοποιηθεί με τη χρήση της Γλώσσας Μοντελοποίησης Συστημάτων (Systems Modeling Language (SysML)), ένα πρότυπο της Ομάδας Διαχείρισης Αντικειμένων (Object Management Group (OMG)). Η SysML επιτρέπει την περιγραφή ενός ευρέος φάσματος σύνθετων συστημάτων και μπορεί να εφαρμοστεί σε διαφορετικές σχεδιαστικές δραστηριότητες και διαφορετικά πεδία συστημάτων, ενώ είναι αποδεκτή τόσο από την ακαδημαϊκή κοινότητα όσο και από τη βιομηχανία.

Μεταξύ αρκετών παραμέτρων που εμπλέκονται στη σχεδίαση συστημάτων, η Παρεχόμενη Ποιότητα Υπηρεσίας (Quality of Service (QoS)) αποτελεί κρίσιμο παράγοντα που περιορίζει τις αποφάσεις σχεδίασης. Το QoS αντικατοπτρίζει τα επίπεδα λειτουργίας ενός συστήματος κάτω από συγκεκριμένες συνθήκες οι οποίες επηρεάζουν την ποιότητα της παρεχόμενης λειτουργικότητας. Παρά το γεγονός ότι το QoS είναι η εύληπτη ένδειξη της ποιότητας των υπηρεσιών που αναμένουν οι τελικοί χρήστες από το σύστημα που χρησιμοποιούν, δεν υφίσταται ολοκληρωμένο πλαίσιο που ενσωματώνει αποτελεσματικά την έννοια της παρεχόμενης ποιότητας υπηρεσίας κατά τη σχεδίαση, χρησιμοποιώντας αποδοτικά MBSD και SysML. Επιπλέον, η τρέχουσα έκδοση της SysML δεν υποστηρίζει άμεσα έννοιες συνυφασμένες με την ποιότητα υπηρεσίας. Ωστόσο, η έννοια της απαίτησης που παρέχει η ίδια η SysML μπορεί να χρησιμοποιηθεί και να επεκταθεί για την περιγραφή και διαχείριση QoS. Ο στόχος των εκτεταμένων απαιτήσεων QoS είναι να απεικονίσουν την ποιότητα που παρέχει το σύστημα στους χρήστες του, από άποψη υπηρεσιών, και όχι να περιγράψουν τη συμπεριφορά του συστήματος. Για την αξιολόγηση τέτοιων απαιτήσεων είναι αναγκαίο να αναλυθεί το σύστημα και να αξιολογηθούν τα αποτελέσματα που προκύπτουν από την ανάλυση, έτσι ώστε να επαληθευτεί εάν ικανοποιούνται οι χρήστες (ή όχι). Η ικανοποίηση ορισμένων απαιτήσεων QoS απαιτεί από το σχεδιαστή να αποφασίσει βασικές πτυχές του συστήματος, όπως η διάταξή του (configuration). Αποφάσεις σχετικές με το QoS κατά τη σχεδίαση επηρεάζονται και από παράγοντες κόστους, όπως είναι τα έξοδα που απαιτούνται για την απόκτηση ή/και λειτουργία του συστήματος.

Στόχος του διδακτορικού έργου είναι η διερεύνηση του QoS κατά τη σχεδίαση συστημάτων με τη χρήση μοντελο-κεντρικής σχεδίασης MBSD και της γλώσσας SysML. Το έργο βασίζεται στο πλαίσιο που προτείνεται από τον Friedenthal et al., το Ενοποιημένο Πλαίσιο Αρχιτεκτονικής (Unified Architecture Framework (UAF)) καθώς και την Εργαλειοθήκη Μοντελοποίησης Συστημάτων (Systems Modeling Toolbox (SYSMOD)). Αυτά τα επεκτείνουμε για να διαχειριστούμε QoS. Ειδικότερα, καθορίζονται απαιτήσεις QoS και ενσωματώνονται σε SysML μοντέλα. Μοντέλα ανάλυσης (analysis models), βασισμένα είτε σε προσομοιώσεις είτε σε μαθηματικούς επιλυτές, χρησιμοποιούνται για την

αξιολόγηση των απαιτήσεων αυτών. Επιπλέον, το κεντρικό SysML μοντέλο ενσωματώνει μοντέλο υποστήριξης αποφάσεων για τη λήψη αποφάσεων κατά τη σχεδίαση, καθώς και μοντέλο κόστους για την αξιολόγηση των κεφαλαιακών δαπανών (Capital Expenditures (CapEx)) ή λειτουργικών εξόδων (Operating Expenses (OpEx)) του συστήματος.

Το πλαίσιο MBSD SysML με επίγνωση της παρεχόμενης ποιότητας υπηρεσίας εφαρμόστηκε σε δύο πρακτικά πεδία, ήτοι τα συστήματα μεταφορών σταθερής τροχιάς (Railway Transportation Systems (RTS)) και τα cyber-physical ανθρωπο-κεντρικά συστήματα (Cyber-Physical Human Systems (CPHS)). Στα RTS, εστιάζουμε στη λειτουργία τους με βάση τον προσδιορισμό και την αξιολόγηση της ποιότητας της άνεσης των επιβατών ενώ περιμένουν σε σταθμούς ή κινούνται μέσα σε τρένα. Η ποιότητα άνεσης ποσοτικοποιείται με την υιοθέτηση και χρήση του όρου Level of Service (LoS), ως τον ποσοτικό δείκτη της απόδοσης μιας υπηρεσίας από την πλευρά του παρόχου υπηρεσιών. Παράλληλα, αναλύονται τα λειτουργικά έξοδα τέτοιων συστημάτων, ανάλογα με τα επιτευχθέντα επίπεδα λειτουργίας LoS. Στα CPHS, εστιάζουμε στη συμμετοχή των ανθρώπων στη σχεδίαση του συστήματος, λαμβάνοντας υπόψη τις ανησυχίες και τις απαιτήσεις ποιότητας που έχουν. Οι συμμετέχοντες εμπλέκονται στη σχεδίαση και στις λεπτομέρειες του συστήματος μέσω συγκεκριμένων οπτικών (παραδείγματος χάριν, οπτική ανθρώπου ή/και οπτική συστήματος) και στοιχείων του μοντέλου, που απεικονίζουν βασικούς δείκτες απόδοσης. Οι ανάγκες των συμμετεχόντων ικανοποιούνται με την κατάλληλη σχεδίαση και διάταξη του συστήματος μέσω μιας διαδικασίας που αναπτύσσεται συνεχώς και διατηρεί το δεσμό μεταξύ του ανθρώπινου συμμετέχοντα και του συστήματος. Ως εκ τούτου, οι συμμετέχοντες αναμένεται να κατανοήσουν το σύστημα και να είναι πρόθυμοι να συμμετάσχουν και να το χρησιμοποιήσουν. Τα πρακτικά πεδία που αναφέρθηκαν αναδεικνύουν τη γενικότητα και την εφαρμοσιμότητα του πλαισίου που προτείνεται.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Συστημάτων
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μοντελο-κεντρική Σχεδίαση Συστημάτων, Γλώσσα Μοντελοποίησης Συστημάτων, Παρεχόμενη Ποιότητα Υπηρεσίας, Επαλήθευση Απαιτήσεων.

Chapter 1

Introduction

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll, Alice in Wonderland

The systematic study, design, and performance evaluation of complex multi-level systems that operate in our physical world are required so that we can understand not only the systems themselves from multiple points of view, but also how they can operate more efficiently, providing their services to end-users. Such systems are practically hierarchies of independent and interactive subsystems, and are typically referred to as Systems of Systems (SoS) [20]. Indicative examples that have been the focus of several research efforts are transportation systems [10], information systems [193], and even healthcare systems [14] that focus on human users.

Model-Based Systems Design (MBSD) [62], a current trend which is adopted by the International Council on Systems Engineering (INCOSE), is a methodological approach for the efficient design of such SoS. Model-based design advocates the creation of conceptual system models that provide unified, visual representations of a system that are clear to all stakeholders at different levels of complexity (e.g., ranging from a single independent structural unit to an entire SoS) [26]. MBSD can be implemented using the Systems Modeling Language (SysML), an Object Management Group (OMG) standard which is endorsed by INCOSE. SysML [199] enables the description of a wide range of systems and SoS and can be applied in diverse design activities and in different system domains, being accepted by both academia and industry [72].

This chapter is organized as follows. We begin in Section 1.1 with an overview of the problem and motivation for this dissertation, and summarize our primary objectives. Then, in Section 1.2, we present the research methodology that provides the guidelines for conducting our research. And finally, in Section 1.3, we describe the structure of this dissertation.

1.1 Problem statement

SoS design is considered to be a complex process of defining and developing systems to satisfy and meet specific user requirements. It includes, among other activities, specification of system structure and requirements, analysis of behavior and performance, and exploration of the most appropriate system design solutions. According to Wymore [293], there are several prominent parameters that should be taken into account when designing a system. In addition to the provided services (system output) or the available technology, Quality of Service (QoS) is a factor that constrains design decisions. The term of QoS [114] has been established by the International Telecommunication Union (ITU) [227] in order to reflect the operating levels of a system under certain conditions that drastically affect the quality of the provided functionality [217].

Despite it being a clear indicator of the quality of services that the end-users of a system are expected to enjoy, QoS is not adequately supported in well-known MBSD frameworks and methodologies, such as Systems Modeling Toolbox (SYSMOD) [287] and Unified Architecture Framework (UAF) [203]. The majority of them focus more on the system and its directly related requirements, like performance, rather than the services provided to the user. Moreover, they typically consider the system designer as the primary stakeholder rather than the user. However, QoS reflects the user's expectations and concerns. Summarizing, currently there is no complete framework that integrates QoS into system design efficiently, using MBSD and SysML.

Moreover, the current version of the SysML specification [261] does not directly address QoS concepts, such as availability, time, comfort, etc. Quality metrics are not supported by a standard notation, i.e. specific SysML elements. Therefore, the primitive notion of SysML requirements [72] can be leveraged and extended to address QoS concepts. The need to extend SysML features to enable modeling of QoS and exploration (e.g., proper analysis of system outcomes) of achievable QoS is identified as a critical issue [83]. To this end, few relevant efforts are noted, such as [129], which attempt to define guidelines for modeling QoS parameters, and how to represent them in SysML.

Our aim is to integrate the exploration of QoS into MBSD. This involves several aspects of the design process: the definition of QoS entities using SysML primitives such as requirements, the integration of such entities into the design of the system, the analysis of the system through simulations or mathematical solvers, the evaluation of the system by verifying the requirements against analysis results, and finally the process of decision-making based on the system design solutions that satisfy the requirements.

To achieve our high-level aim, we target the following concrete objectives.

1. A way for the description and management of QoS in the form of requirements is needed.

2. Analysis models are required to facilitate the computation of QoS and check whether it can be fulfilled by the system; these models should be integrated within the core system model, helping a system designer perform QoS analysis within the same environment where corresponding requirements are defined.

3. Design decisions (i.e. proper system design configurations) that should verify QoS requirements should be proposed by the design environment, e.g., in the form of feedback.

4. Cost parameters should be leveraged while designing a system; cost can be considered a QoS parameter since QoS and cost are inter-dependent, and can be defined within the system model as requirement. In essence, a system designer should be facilitated to model, design and evaluate a system from a cost perspective, performing a cost analysis both at a high level (e.g., on the level of overall capital or operational expenses of a system) or in a fine-grained fashion (e.g., the individual cost of a single structural unit).

To manage these objectives, we extend the framework of Friedenthal et al. [72], where the system model is depicted as an integrating framework of structure, behavior and analysis models, as well as requirements. To serve QoS aspects in system design we seamlessly integrate external tools that support system analysis and decision-making. We further propose basic SysML extensions that facilitate the specialization of the SysML meta-model to represent QoS concepts. We verify the validity and generality of our MBSD framework by applying it to two different system domains with their own distinct objectives; one from the transportation domain (Railway Transportation Systems (RTS)) [148] and one from the human-centric Cyber-Physical Human Systems (CPHS) domain [145]. To facilitate domain specialization, domain-specific SysML profiles were

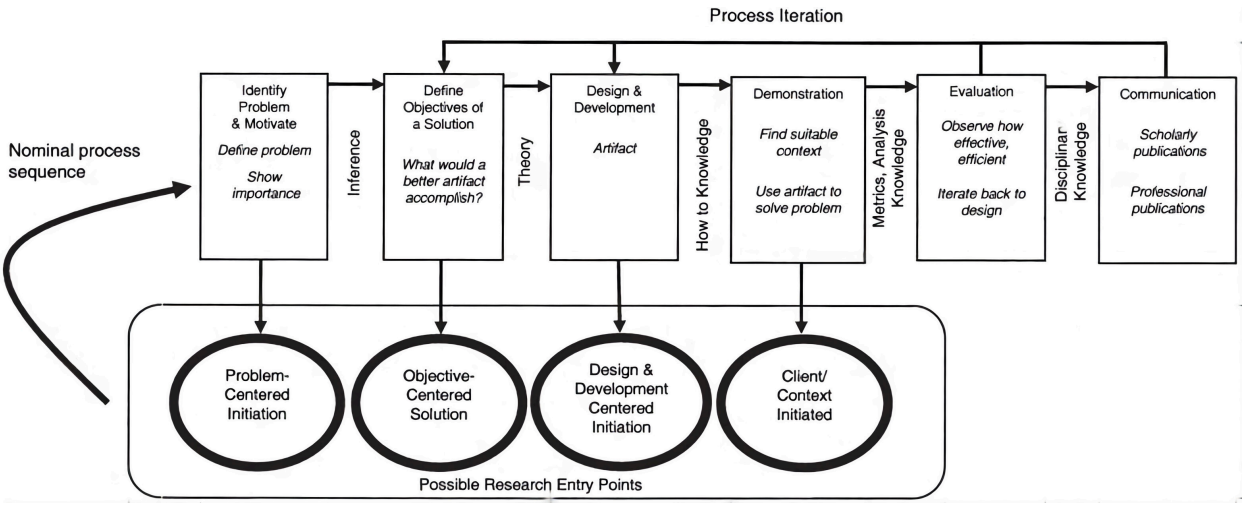


FIGURE 1.1: DSRM process model, presented in the work of Peffers et al. [215].

defined and used according to the proposed framework. The research on the application domains was supported by partnerships and projects that ran through the course of the PhD.

Having identified the problem, our aim and objectives, we conduct our entire research using the research methodology that is described in the following.

1.2 Research methodology

According to Simon [252]: "whereas natural sciences and social sciences try to understand reality, design science attempts to create things that serve human purposes". Scientific literature [3, 107, 100, 112, 84], relevant to design science, reveals the ongoing and increased interest in Design Science Research (DSR) as an approach to research and develop knowledge and artifacts (e.g., concepts, models, etc.), in order to design system solutions or improve existing ones and solve problems [18]; in this dissertation, we follow the rationale of the DSR.

To conduct this type of research, guidelines, provided by a suitable methodology, are needed [279, 280]. Thus, we adopt the Design Science Research Methodology (DSRM), a scientific approach to generate models, as presented in the work of Peffers et al. [215]. The DSRM includes six successive and iterative steps, illustrated in Figure 1.1. The research begins with the *identification of the problem*, defining the primary research problem and motivations and justifying the value/importance of a solution. The next step is the *definition of objectives of a solution*; these objectives attempt to answer the question: "what would a better artifact accomplish?". The *design and development* of artifacts like frameworks, software, models, methods, etc., comprise the third step, supporting the achievement of the defined objectives and highlighting the way to solve the problem. The *demonstration* step follows, where artifacts are selected to actually solve the problem, and are applied to the context of concrete use cases. Afterwards, during *evaluation*, the efficiency and effectiveness of the methodology is studied, comparing objectives and actual –observed– results from the use of the artifacts; in case the evaluation is inconclusive, corrective actions can be taken where required, leading to the previous steps of setting objectives (second step) or designing and developing an alternative solution (third step). The final step is *communication*, where research publicity actions can be carried out, e.g., through scholarly or professional publications in international conferences and journals.

In our work, the following activities are carried out in each step of the DSRM:

(i) **Problem identification and motivations:** In the majority of MBSD frameworks and methodologies, no emphasis in introducing QoS has been given, even though QoS captures what the system provides to its respective end-users, service-wise [217]. Our proposed QoS-aware MBSD framework focuses on this shortcoming.

(ii) **Setting objectives to solve the problem:** Our main objective is to enable system designers to integrate additional quality characteristics within system models in the form of QoS requirements, during MBSD. In addition, designers should evaluate system performance and assess analysis outcomes regarding QoS, in order to verify whether the system satisfies its end-users (or not). The objectives can be specified and adjusted depending on the development of the research and the design of the system, and can include the automated generation of recommendations/feedback for the improvement and optimization of the provided quality.

(iii) **Design and development of QoS-aware MBSD framework to achieve the objectives:** This step includes the following:

- The development of meta-models that formalize the aspects and concepts used by a modeling language, like Unified Modeling Language (UML) [259] or SysML [199], and that are applied to system domains to model them. In this work, a QoS meta-model is developed for the specification of the quality of services provided in system models, in relation to requirements. A cost meta-model is also introduced to facilitate the integration of cost-related information into the system models, providing cost-related entities that group cost properties, as well as the automated computation and evaluation of these cost properties, conforming to budgetary requirements.
- The development of SysML profiles, depending on the meta-models, as well as on the system domains, to enable the definition of structural and QoS characteristics under cost restrictions in system models. At this stage, software development (e.g., plugins) compatible with SysML modeling toolsets (such as MagicDraw (MD) [178] or Cameo Systems Modeler (CSM) [32, 37]) may be required to further facilitate model configuration.
- The selection and use of mathematical or simulation-based analysis models and frameworks, compatible with the domains, for system performance analysis.
- The integration of mathematically-computed or simulation-based analysis results back into the system models, through the aforementioned extension software (plugins), and their utilization for the evaluation of the quality of the provided services under the cost restrictions.

(iv) **Demonstration:** The proposed framework is applied to two heterogeneous, real-world system domains, i.e. a public RTS and a healthcare CPHS, in order to examine the research proposals on a practical level. Each application domain represents a system that has different scopes and objectives. In the context of the RTS [103], system operators need to study the improvement of spatial comfort of the passengers inside trains or while waiting at stations, measured based on international standards (e.g., Level of Service (LoS) [270]) against the corresponding increase of operational costs. In the context of the healthcare CPHS [255, 119], patients need the exploration of different solutions for the system that is installed within their homes to monitor their health condition, under different service and budget (capital costs) restrictions which are set by themselves, according to their needs.

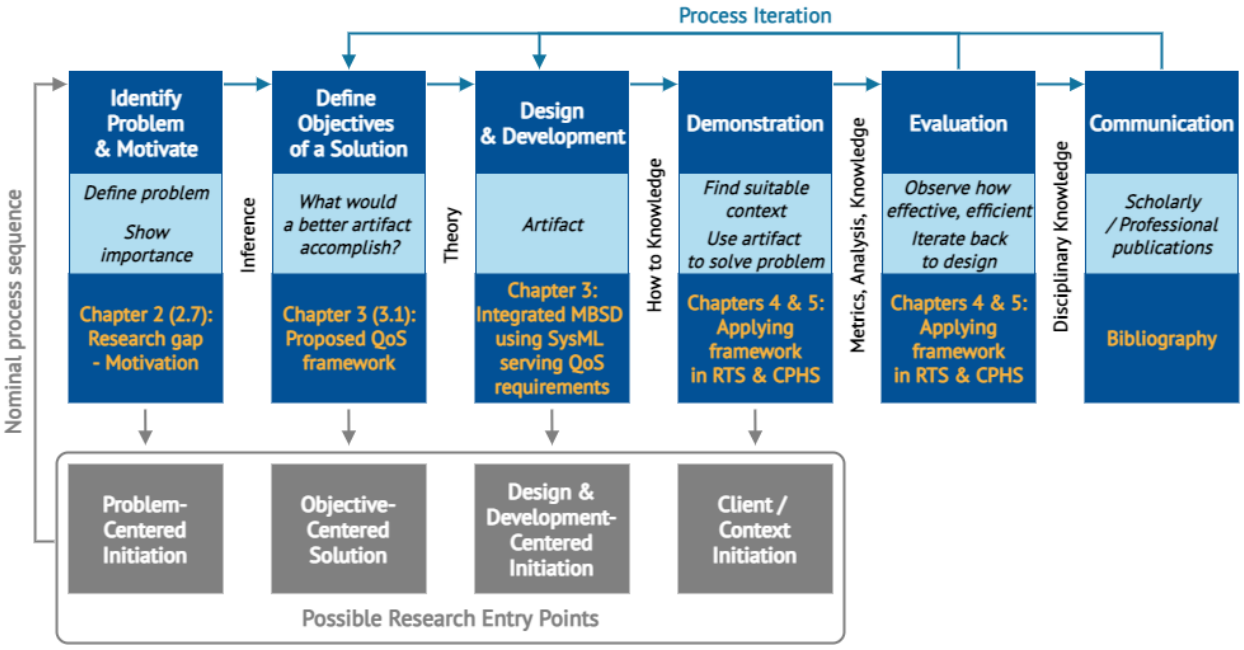


FIGURE 1.2: Dissertation outline, compliant to the steps of the DSRM process (adapted from the work of [215]).

(v) **Evaluation:** With the applied solutions from the demonstration step, tests are conducted to examine the research results in terms of various criteria (efficiency, effectiveness, usability, etc.). Based on the DSRM, if the evaluation is inconclusive or unacceptable, changes to previous steps can be made while the process can be repeated in several iterations. In essence, systems are tested against QoS requirements and cost restrictions; this enables system designers to re-do steps (if necessary) and provide effective solutions that are acceptable by the end-users of the systems.

(vi) **Publication:** The results of the research are published in international conferences and journals [141, 142, 143, 144, 145, 146, 147, 148, 149, 140, 151, 150]. The selected design research organization (i.e. problem, objectives, design/development, demonstration, evaluation) is used for the writing of the peer-reviewed publications.

1.3 Dissertation outline

Our dissertation is structured in conformance to the iterative DSRM process (see Section 1.2). Based on Figure 1.2, chapters correspond to different steps of the research methodology as described in the following.

In **Chapter 2**, we review related work and background information that is required to support our work on the proposed framework and application domains; related work is based on the large-scale literature survey that we carried out during all the years of the PhD research. In addition, we outline our motivations and scope.

In **Chapter 3**, we analyze our integrated MBSD framework, based on SysML, to serve QoS and QoS-related requirements under specific cost restrictions. Based on the work of Friedenthal et al. [72], we also depict the SysML system model as the core of our framework. We describe external models, like analysis or decision-support models, which are integrated to it. In addition, we analyze custom meta-models, like a QoS meta-model, containing QoS-related SysML extensions

to serve quality of service in SoS design or a cost meta-model, comprising cost-specific SysML extensions for the economic assessment of these SoS, which are also applied to the system model. We conclude the chapter by presenting the environment where our integrated framework is implemented.

In **Chapter 4**, we present the application of the proposed MBSD framework to the domain of RTS, in order to explore the QoS-related design of such systems under restrictions of operational costs. We focus on the spatial comfort of passengers within moving trains or while waiting at stations, as a key metric of QoS in RTS; we classify the quality of passenger comfort using the term of LoS. In addition, we discuss the need to perform a cost analysis in a railway system, balancing passenger comfort LoS and operational costs. We apply our QoS meta-model to create a RTS SysML profile for the exploration of passenger comfort LoS, as well as our cost meta-model to enable the design and evaluation of such systems from an operational cost perspective. As a case study, we present the improvement of passenger comfort LoS under operational cost constraints in the Athens Metro RTS.

In **Chapter 5**, we present the application of our framework in the domain of CPHS, in order to explore the QoS-related design of such systems under restrictions of acquisition costs. We focus on the engagement of human participants in the design of CPHS, depicting their human concerns as key performance and acceptance indicators in these systems. In addition, we discuss the need to perform a cost analysis in human-centric systems like CPHS, balancing different service and budget restrictions, set by the users themselves. We apply our QoS meta-model to create a CPHS SysML profile in order to support the human-centric perspective in CPHS design, as well as our cost meta-model to enable the design and evaluation of such systems, from an acquisition cost perspective. As a case study, we present the design of a Remote Elderly Monitoring System (REMS) under the budgetary constraints of patients, regarding the equipment installed in their homes; we also depict the decisions that a REMS designer can make w.r.t. the configuration of a REMS that covers the needs of patients.

In **Chapter 6**, we conclude this dissertation with a concrete review of the contributions of our research; first we provide contributions individually for each application domain, and then we provide a critical assessment of the work within a larger research context. Finally, we offer an outlook on future work in the associated areas of interest.

Research overview. The overview of our research, containing motivation, problems, objectives, and research methodology is summarized in Table 1.1, along with the main contributions of our work which shall be discussed in detail in Chapter 6 (Conclusions).

TABLE 1.1: Research overview.

Motivation	Problem statement	Research objectives	Research methodology	Contributions
QoS should be integrated in MBSD methodologies – directly affects users – critical services – low QoS = low acceptance – no easy computation	How to integrate QoS in all MBSD activities (not effectively supported).	Description and management of QoS.	Based on ⇒ Design Science concepts. Artifact ⇒ Extended~QoS-aware MBSD SysML framework. Evaluation ⇒ Applied framework in two practical application domains.	QoS-aware MBSD SysML framework as suggested by Friedenthal – is compatible with UAF and SYSMOD, – has basic SysML profiles to support QoS and decision-making. Ideas for SysML 2 standard. Provide solutions for managing QoS in RTS and CPHS domains (through specific SysML profiles).
		Automated decision-making to achieve QoS.		
		Apply framework in discrete SoS domains.		
	No native SysML support for QoS concepts.	Enable modeling of QoS (via requirements).		
		Explore achievable QoS (analysis of system outcomes).		

Chapter 2

Related work - Motivations

> "If I have seen further than others, it is by standing upon the shoulders of giants."
>
> Isaac Newton

In this chapter we provide the related work and background information that is required to support our work on the proposed framework (Chapter 3) and application domains (Chapters 4 and 5).

Specifically, in Section 2.1, we analyze the term of QoS and its value in the design of SoS.

In Section 2.2, we present an overview of MBSD, as the most prevalent approach for the efficient design of complex systems and SoS.

In Section 2.3, we describe SysML, a de facto standard that facilitates the adoption of MBSD; we depict SysML's primary features (see Section 2.3.1), and present a survey of QoS and cost SysML profiles (see Sections 2.3.2 and 2.3.3).

In Section 2.4, we present an overview of well-known MBSD methodologies that primarily use SysML, like Friedenthal et al. [72] that depicts the SysML system model as an integrating framework for efficient MBSD (see Section 2.4.1), Weilkiens's SYSMOD [286] that uses SysML to provide an overview of the MBSD process, focusing on system requirements (see Section 2.4.2), and the UAF [203] which acts as a standardized SysML framework that can be applied to MBSD, facilitating the development of interoperable systems with traceability to requirements and across different views (see Section 2.4.3).

We further present short surveys on works in literature for system model transformations within MBSD in Section 2.5, as well as surveys on MBSD approaches, based on SysML, to enable effective decision-making during system design in Section 2.6.

In Section 2.7, we describe our motivations and scope that are based on the QoS, MBSD, and SysML concepts, described in detail in previous sections.

Lastly, we present our application domains in Section 2.8. Specifically, we depict an overview of RTS works in Section 2.8.1, and an overview of CPHS works in Section 2.8.2; we conclude these chapters by further describing remaining issues that need to be explored in each application domain.

2.1 Systems of Systems (SoS) design and Quality of Service (QoS)

The term of QoS [114] has been established by the ITU [227] as "the collective effort of service performance, which determines the degree of satisfaction to the end-user". Additionally, QoS is defined as "a set of user–perceivable attributes of that which makes a service what it is. It is expressed in user–understandable language and manifests itself as a number of parameters, all

of which have either subjective or objective values" [23]. Thus, QoS covers both subjective and objective aspects of a system service. This means that the scope of QoS is a broader area than the scope of system performance; the latter is objective and refers to the ability of the system to provide desired functionality. There are several attempts in literature that focus on the exploration of system performance and assess the system (verify it and validate it) based on system performance requirements [273, 50]. According to Wymore [293], performance is among the several prominent parameters that are taken into account when designing a system besides provided services (system output) or available technology. It should be noted that Wymore adopts the term performance to describe all non-functional requirements the system should conform to, also including availability, security, QoS, etc.

In general, QoS can serve as a clear indicator of the quality of experience that the user of the system is expected to enjoy. Guaranteeing a minimum threshold for specific qualitative or quantitative characteristics of the system renders the provided service either acceptable or non-acceptable for the user [217].

Some classic examples, where systems have defined QoS, can be found in the telecommunications field. The work of [138] is among the few attempts to include the term of quality of service as well as, quantitative characteristics of QoS, during system designing stages. Relevant to that notion, a generic conceptual framework to describe quality of service characteristics is provided [98].

2.2 Model-Based Systems Design (MBSD)

The process of design and development of SoS (i.e. Systems Engineering (SE) [95, 92]) is a methodological approach for their design, implementation, technical management, operation, and potential improvement. In particular, Systems Design (SD) is the core activity of this process; according to Ryschkewitsch et al. [234]: "systems engineering is first and foremost about getting the right design". SD is defined by Wymore [293] as "the development of a model based on which the real physical system can be implemented, meeting all its requirements". Therefore, the term of Model-Based Systems Design (MBSD) [62, 283] is introduced. MBSD has been proposed as a horizontal approach for the seamless integration of diverse complex system development activities, minimizing errors, effort, cost, and time of development [281].

On account of the following, MBSD is the most prevalent approach for efficiently designing SoS [171]. It advocates the construction of conceptual system models that provide unified, visual representations of a system, which are clear to all involved stakeholders, at different levels of complexity (e.g., from an independent structural entity to an entire SoS) [26]. In addition, a core system model is the means for effectively transitioning from a design/development activity to another, while allowing the use of appropriate methods and tools for each activity. Specifically, it contributes to the precise definition and specification of systems, their comprehensive evaluation, their validity and correctness, as well as their efficient operation. Based on Friedenthal et al. [72], the model of a system is used as point of reference throughout the entire design of a system, i.e. from defining its architecture/structure, to collecting, verifying and managing its requirements, to assessing and evaluating its performance, up to its improvement, in case requirements are not met during the assessment stage. Such models can be verified and be re-iterated at any design stage [16]. Note that typically, system models are described by the SysML standard [261, 185], which is discussed in the following section. MBSD is also verbose regarding system modeling artifacts, and presupposes the design and development of modeling structures. Thus, MBSD approaches specialize either in terms of application domain [2, 184, 264] or in terms of system category [242, 113, 266, 4, 13, 10, 14, 193].

2.3 Systems Modeling Language (SysML)

According to Wolny et al. [292], over the last thirteen years, research interest moved towards the adoption of UML [259, 245] or SysML [199, 286] as system modeling languages for modeling and designing the structure, behavior, and requirements of systems.

In particular, SysML [286, 199] is a de facto standard, endorsed by INCOSE, that facilitates the adoption of MBSD, providing (i) semantics, i.e. rules that govern the specification and structure of system models, and (ii) notation, to graphically/textually represent the semantics. According to INCOSE [199], "SysML enables the specification, analysis, design, verification, and validation of a broad range of systems and SoS that may include hardware, software, information, processes, personnel, and facilities". SysML became an OMG standard in 2007 [199]; its current version is SysML v1.6, issued by the OMG in December 2019. It is applied in diverse design/engineering activities and system domains, while it is widely known and well-accepted by both the academia and the industry [292, 221] due to its effectiveness in representing complex systems and SoS [17, 242, 88], and supporting the verification and validation of such systems [164].

SysML is a dialect of UML 2, and is defined as an extension of the UML meta-model, i.e. a UML 2 profile [230]. In general, a modeling language contains a number of distinct language concepts, which are represented by a collection of meta-classes [259]. Meta-classes define the behavior of certain elements and instances and include a set of properties and constraints to do so. The collection is called meta-model, and the underlying meta-model for SysML is UML4SysML (i.e. UML for SysML) [96].

SysML also includes built-in mechanisms (like profiles) to customize the language. Regarding a profile, it is considered a kind of structural diagram that provides a generic extension mechanism for customizing models for particular domains and platforms. According to Alhir [5], "extension mechanisms allow refining standard semantics in strictly additive manner, preventing them from contradicting the standard semantics". In addition, profiles extend and customize the language by adding new model elements, creating new properties, and specifying new semantics, in order to make the language suitable to a particular problem domain; there are three extensibility mechanisms [105]:

- *Stereotypes* provide a way to extend the language, as well as the possibility to add certain properties to an already existing meta-class. They facilitate the definition of new model elements, that may be derived from existing ones, although having specific properties suitable for a specific domain. Typically, a profile contains a set of stereotypes representing a contiguous set of concepts for a specific modeling domain.
- *Tagged value definitions* are used to extend the properties of a stereotype during its specification, associating additional information to it.
- *Constraints* are used to extend or specify semantics/conditions of model elements, that should be fulfilled by the system.

In essence, to describe domain-specific systems, SysML profiles are used [5], while the stereotype is the basic SysML structural element to define specialized entities based on existing ones, and can be applied to UML model constructs like *classes*, *attributes* and *operations* [105].

Based on [88], the main reasons for selecting SysML for system modeling are the following: (i) it has been proven to be expressive enough to model complex systems with heterogeneous components, and (ii) the SysML profiles provide the necessary flexibility for tuning or extending native constructs; examples include, but are not limited to, generalizations of SysML Block or SysML Requirement stereotypes [72].

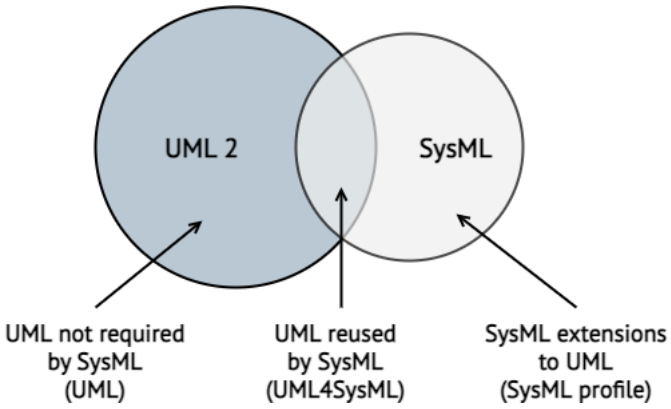


FIGURE 2.1: Relation of SysML dialect to UML 2 parent language (UML4SysML).

2.3.1 SysML features

As shown in the Venn diagram of Figure 2.1, SysML reuses a subset of UML 2 (UML4SysML) [96], and it defines its own extensions, i.e. new modeling constructs that either have no counterpart in UML or replace UML constructs. That is, it utilizes UML model elements and standard diagrams, such as activity, sequence, state diagrams [259], to model the behaviour of systems, and provides specialized, discrete diagrams for designing and modeling SoS as a hierarchy of autonomous building blocks, either composite or simple, in different levels of detail. Relationships between discrete block diagrams describing different system views are also explored in SysML. The language caters for requirement modeling as well. While the relationships provided to depict requirement interrelations proved to be semantically efficient, requirements themselves were initially described in a textual form, leading to numerous extensions to accommodate requirement description, especially quantitative ones [72].

Basic features of SysML and its improvements over UML for more effective MBSD [230, 105, 72, 286] are presented in the following.

- SysML reuses seven of the fourteen diagrams of UML 2.

  The *Activity Diagram (ACT)* represents system behavior in terms of the ordering of actions within the system, based on the availability of inputs and outputs [117].

  The *Sequence Diagram (SD)* also refers to system behavior in terms of a sequence of messages exchanged between parts of the system [33].

  The *State Machine Diagram (STM)* represents behavior of a system entity in terms of its transitions between states, triggered by events [55].

  The *Use Case Diagram (UC)* represents functionality in terms of how a system or a system entity is used by external entities (i.e., actors) to accomplish a set of goals [55].

  The *Class Diagram (CD)* describes the static structure of a system [177]. In SysML, the CD is replaced by the *Block Definition Diagram (BDD)*, that represents structural elements as building blocks, as well as their composition and classification in the form of hierarchies [72].

  The *Composite Structure Diagram (CSD)* depicts the internal structure of a system, including the interaction points of system parts to others [201]. CSD is replaced by the *Internal Block Diagram (IBD)* that represents interconnections between parts of blocks [96] in SysML.

  Finally, the *Package Diagram (PKG)* represents the organization of a system model in terms of packages that contain model entities [99]. Note that the activity and package diagrams

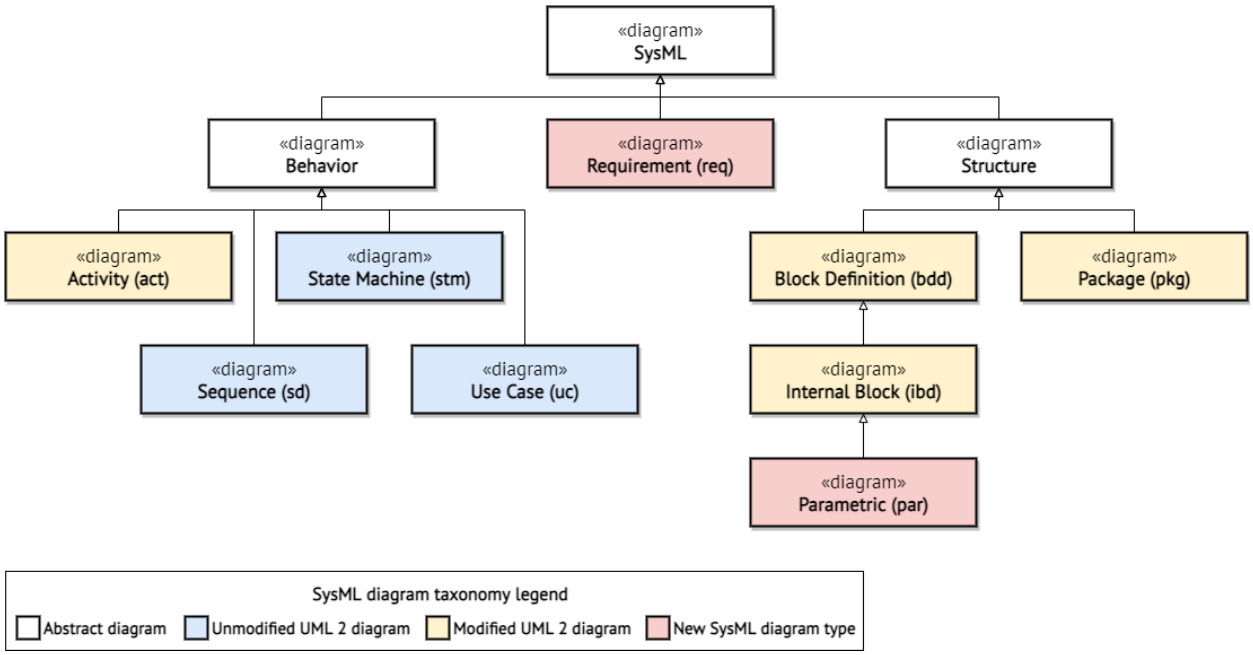


FIGURE 2.2: SysML diagram taxonomy.

are slightly modified, while the sequence, state machine, and use case diagrams remain the same as in UML 2.

SysML also adds two new diagram types: (i) the *Requirement Diagram (RD)*, that is used for the design and engineering of requirements, representing them in a text-based form and their relationships with other requirements, design model entities, and test cases [199], and (ii) the *Parametric Diagram (PD)*, that is used for the performance analysis or quantitative analysis of a system, representing constraints on system parameters [72, 211].

Figure 2.2 depicts the taxonomy of the nine SysML diagrams, each one focused on a particular aspect of a system. These diagrams reduce the software-centric restrictions of UML, being more flexible and expressive, and extend the capabilities of SysML to model and design a wide range of complex systems. In addition, based on diagram types and model constructs, SysML is a comparatively smaller language than UML, thus, it is easier to learn and apply.

SysML represents four basic areas/concepts that are expressed as "Four pillars of SysML" by Cris Korbyn [70]; these pillars include the modeling of *Structure*, *Behavior*, *Requirements*, and *Parametrics*, and refer to essential diagram types of SysML, as shown in Figure 2.3. Based on the work of Hampson [93], system structure can be divided into (i) system hierarchy, that is depicted within a BDD, and (ii) internal system structure that is depicted in an IBD. System Behavior can be described by an ACT that has modifications from its UML counterpart; note that UC, STM, or SD can also be used. Requirements can be defined, represented, and related to the system model within a RD. Finally, parametrics deal with the definition of constraints to give system parameters additional capabilities (e.g., performance analysis) in the system model; a PD supports this need in SysML. The structural BDD and IBD, the RD and the PD are described in detail in the following.

- SysML also supports *Allocation* tables, a tabular format that can be dynamically derived from SysML allocation relationships, and supports requirements, functional, and structural allocations; UML provides only limited support for tabular notations. This capability facilitates

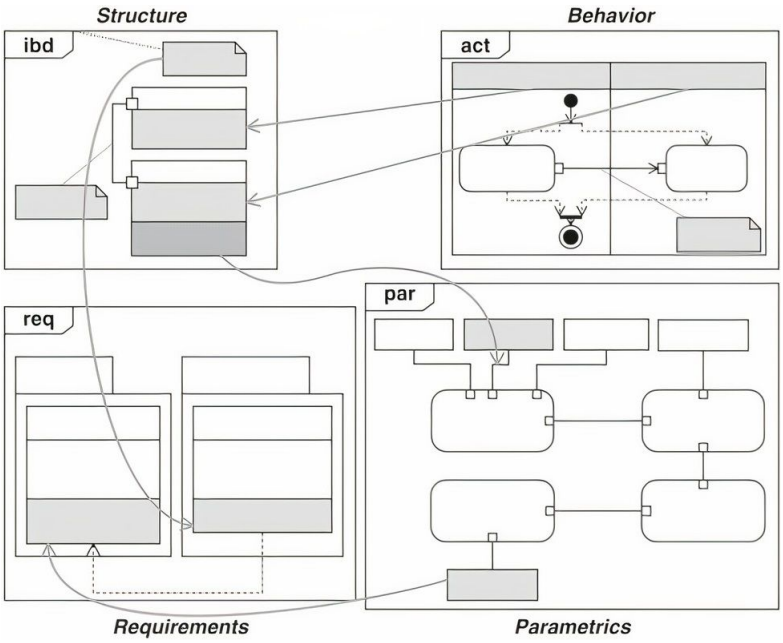


FIGURE 2.3: "Four pillars of SysML", shown in the work of [72].

the automated verification and validation of a system.

- The final improvement is that SysML's model constructs support views and viewpoints [199]. These constructs extend the capabilities of UML, and are architecturally aligned with IEEE-Std-1471-2000 standard [102].

We further describe the basic diagram types of SysML, based on the work of [105, 72, 286], in the following.

Block Definition Diagram (BDD). This diagram is used to model the static structure of a system. It is derived from the UML Class Diagram, while it redefines it replacing a class with a block, i.e. an atomic SysML unit of system structure. This diagram depicts definition of blocks, and how these blocks relate to each other to form the system.

The SysML Block is a unifying concept for describing the structure of an entity, like system, software, hardware, data, person, etc., as well as compositional, logical, and/or inheritance relationships between entities. A block may contain: (i) structural characteristics, like value properties that describe values for quantities, or ports that specify interaction points on blocks (where items can flow in or out), (ii) functional characteristics, like operations that describe the functionality of a block (what it can do), (iii) relationships, that include associations to other blocks, generalizations from other blocks, or allocations from/to other model entities, and (iv) other user-defined properties.

Internal Block Diagram (IBD). It is used to depict the internal view and usage of a system block. In comparison with UML 2, it redefines the CSD by supporting blocks and their ports. It can be instantiated from a BDD to represent the final assembly of all blocks within a main system block. Composite blocks from the BDD are instantiated on the IBD as parts. These parts are assembled through connectors, linking the parts directly or via their ports.

Requirement Diagram (RD). According to [199], "a requirement is a capability or condition that a system must ("shall") satisfy". There are functional requirements that specify a function that a system must perform, and non-functional requirements that specify quality criteria that can be used to test the effectiveness of system functions. A basic use of requirements is to formalize the needs of the system stakeholders (e.g., end-users), which will be realized as functionality and constraints, that should be satisfied by the system or its provided services. SysML provides the RD, defining a visual and graphical representation of text-based requirements, specialised associations between these requirements or with other system model elements, and how they can be managed in a structured and hierarchical environment.

The SysML Requirement includes two basic properties, i.e. an identifier (id) and a textual description. Note that additional user-defined properties such as a verification method can also be added. SysML added new kinds of relationships for requirements; these include *containment*, that is used when a requirement may comprise others, *deriveReqt*, that is used when one or several requirements are derived from another requirement, *satisfy*, that depicts one or several model elements fulfilling a requirement, *verify*, that is used when one or several model elements, e.g. a test case, verify that the system fulfils a requirement, and *refine*, when one or several model elements, e.g. a use case, further refines a requirement, providing additional characteristics to it. SysML provides a graphical depiction of these relationships. It also provides means to capture rationale and traceability for a specific requirement using *allocate*, *trace* or *copy* relationships.

Parametric Diagram (PD). This SysML diagram is intended to support the analysis of a system (performance, reliability, etc.), ensuring consistency between the system design model and multiple engineering analysis models, and facilitating the identification and management of critical technical performance measures/properties. To perform system analysis, the PD defines SysML constraint blocks. PD and constraint blocks allow the definition of constraints that represent rules that constrain the properties of a system, or that define rules that the system must conform to. Constraint blocks express these rules in the form of mathematical equations or logical expressions, and define a set of parameters that are constrained by these equations/expressions. Some of the constraint parameters may be bound to value properties of system blocks via connectors that link them. The expression language for equations or expressions can be can be formal (e.g., Modelica [74], Object Constraint Language (OCL) [229]) or informal, while the computational engine for their computation is not provided by SysML itself, rather by external, applicable analysis tools.

2.3.2 SysML QoS profiles

SysML has been widely accepted by the scientific community, due to its effectiveness in representing complex systems and SoS [17, 242]. However, despite the extensive interest in using the SysML in MBSD, a review of the relevant literature reveals that there has not been significant progress in adopting service quality in the examined complex systems.

In [129], service quality primitives are introduced to SysML's conceptual framework and an attempt to define guidelines for modeling quality of service parameters is made. In [160], the estimation of the system's quality, requires suitable evaluation characteristics for design and analysis. An efficient way to manage quality, is via model-based approaches and a system model perspective [165], that can lead to efficient system design and development. While various model-based design techniques [118] have been proposed, only few properly model and integrate critical requirements, like QoS, etc. [300] into the system. For example, the authors of [53] investigate specific QoS requirements in Cyber-Physical Systems (CPS), while depicting some state-of-art CPS

QoS models. In addition, in [79], adaptive systems are designed via quality requirements and parametric models, verified during system execution.

In order to design a system, its architecture is specified to meet specific criteria in terms of its behavior. Such architectures are typically depicted by hierarchical models of structural components (more or less complex) that interact with each other. However, the question is: *how is the correctness of the proposed architecture judged?* The conditions under which these architectures will provide the prescribed services, i.e. criteria that describe QoS, such as their effectiveness and efficiency, are important for their evaluation.

Few literature works include quantitative QoS characteristics when designing systems of various domains [138, 98, 129, 160, 300, 53, 79]. Given that SysML as a system modeling language has already been widely accepted by the scientific community due to its effectiveness in describing and visualizing complex system architectures, the following questions arise: (i) *how are parameters that describe QoS depicted in SysML models*, (ii) *how do these parameters relate to the building blocks that describe these systems*, and (iii) *how is QoS assured*? The parameters of a system model that describe QoS can be represented by the concept of the requirement. The latter is used to practically depict the specifications under which the system should operate [116].

Requirements are created during the initial phase of the requirements analysis of a complex system, and are specified and validated during all system design phases. According to [219], this makes their management an extremely complex process. Requirements are categorized into functional, i.e. how the system should operate, and non-functional, i.e. under which conditions it should operate. Satisfying Non-Functional Requirements (NFR) can be important during the systems designing process, since they determine the conditions that affect the functionality of the system. NFR may include performance (e.g., response time, throughput, etc.), reliability, availability or usability characteristics [87], while maintaining a qualitative and quantitative nature.

In order to confirm that the requirements are met, various attempts and approaches, like [194], to evaluate the performance of SysML system models using simulation, have been made. In parallel, test cases using SysML can be described, in order to assess and verify qualitative NFR. Exploiting NFR, the quality of service can be assured, depicted and integrated into a SysML system model. In addition, there are toolsets that can be used [134], [239], supporting the verification of requirements.

To sum up, integrating frameworks and methodologies [15], [251] within commercial software packages [178] can provide an appropriate design environment and support the representation and integration of QoS in SysML system models. However, research efforts and commercial solutions focusing on SysML as a driving force towards MBSD, do not provide adequate and efficient support of QoS management.

2.3.3 SysML cost profiles

According to Wymore [293], besides the desired output and available technology (e.g. services), performance and cost are considered critical design parameters that should be measured, evaluated, and preserved during system design. Most system design methodologies, model-based or not, mainly focus on the achievement of diverse performance requirements for the studied system(s) [160, 187]. However, if the system designer focuses solely on performance, ignoring the incurred costs/expenses, the resulting system could be cost-prohibitive, and thus unusable [260].

Literature is rather poor into the cost context, since cost is typically superseded by performance or other factors during system design [129]. In the cases where cost is taken into account not all its different dimensions (e.g., cost categories like operational expenses, component/equipment costs, etc.) are explored; only the design costs or the costs incurred during the system's

architectural changes are studied [47]. Common pitfalls of such approaches are the lack of generality, scalability, dynamicity and simplicity. Therefore, there is a need to cover this identified gap in order to develop and propose a framework into which the cost dimension will merit the importance it really has.

A number of methodologies are available in the pertinent literature, which perform cost analysis on individual system domains. In the public transportation domain, the authors of [82] perform a comparison of different cost assessment methodologies with their mathematical formulations. Essentially, they provide transportation planners and policy makers with a systematic process for estimating costs that are representative of the area and service, for analysis and decision-making purposes. Their approach can be used as an intermediate tool to allow planners to perform railroad cost analysis more easily, determining cost functions. In a similar context, in [271], the Activity-Based Costing (ABC) is employed as a suitable method for a business-economic transportation cost model, while traditional book-keeping approaches, like Traditional Cost Accounting (TCA) that make rigid assumptions about system resource utilization in railways, are deprecated. However, adequate model-based design methods, which are driven by representative system models that properly model and integrate critical design requirements (including costs) are still needed [300].

Moving beyond a single application domain and employing MBSD, we identify the following lines of related work. In [248], three methods of estimating system costs: analytical, analogous and parametric, are described. Focusing mostly on parametric estimates, its authors propose a commercial cost model that though has some stringent requirements, such as cost engineer training, fine-grained calibration and additional components (such as databases). Constructive Systems Engineering Cost Model (COSYSMO) [158] is a well-known costing methodology, widely adopted with model-based system design, targeting SoS. Most approaches exploit it as basis for effective cost analysis. For example, in [167], the COSYSMO Effort Equation is employed for a proposed parametric effort formula for constructive cost models. Focus is given on translations between models/tools in MBSD, specifically mapping architectural elements into behavior/performance analysis and cost model inputs. SysML, Department of Defense Architecture Framework (DoDAF) [209], COSYSMO, Constructive Cost Model (COCOMO) [22] or COCOMO II [158] parametric cost models, and other frameworks and environments are also investigated in that work [167]. As a case study, the Remote Targeting System for Unmanned Aerial Vehicle Intelligence, Surveillance and Reconnaissance (RTS UAV ISR) is explored. In addition, cost model interfaces for components of the architectures are developed in order to evaluate cost effectiveness in such environments. In [210], the aforementioned COSYSMO framework and in particular, a practical implementation of the COSYSMO cost estimating relationship, is exploited. Extending an MBSD modeling environment with SysML, the authors achieve an end-to-end estimation of systems engineering effort in designing and developing a system.

The authors of [260] focus on equipment costs. Since most studies make use of use case-specific ad-hoc models (typically, a combination of visualization and spreadsheet modeling) that are both hard-to-use and error-prone, the authors develop the Equipment Coupling Modeling Notation (ECMN), a flowchart-like notation, based on a small number of building blocks, that facilitates hierarchical modeling by means of nesting models (using sub-models).

In [129], basic indicators, like cost and performance are explored in SysML's conceptual framework and an attempt to define guidelines for modeling related parameters is made. However, that work does not provide solid, practical examples of SysML models and extensions. In addition, the authors of [7] carry out a comprehensive research on managing non-functional cost/performance requirements through MBSD. They observe that in the design process, several research efforts focus only on specific requirement types and systems. In essence, there is a lack of a comprehensive and generic framework that adequately covers a broad range of requirements and systems.

We note that there are more specialized efforts that exploit the SysML profiling mechanism to design a system, while performing a techno-economic or cost analysis to evaluate it [65, 192]. In [162], a trade-off analysis among alternatives for a system model is presented, addressing MBSD within a SysML context. This way, the authors meet design objectives, such as cost, performance or other inputs derived from the system stakeholders' needs. Since these needs are often conflicting, the trade-off analysis is expected to provide a balanced solution [72]. Among several methods that are used to specify, design and verify a system, based on cost or performance factors, the authors propose the scenario-driven Object-Oriented System Engineering Method (OOSEM) from OMG that primarily uses SysML. Even though OOSEM can be used to evaluate design objectives such as cost, it reaches a limitation when a large number of system alternatives and solutions must be evaluated.

An effort that aims for generality, is [65], where a model-driven techno-economic approach for the estimation of economic parameters of cloud service deployment is proposed. The authors employ SysML to describe cloud architectures, emphasizing cost-related properties. The Total Cost of Ownership (TCO) for cloud infrastructure and services is their use case; they incorporate TCO into SysML cloud models, while cloud providers are facilitated in computing it. An evolution of this approach, in the same domain, can be found in [192]; this inspired the use of SysML requirements that are attached (and should be satisfied by) specific cost entities. However, for both these works it is not clear whether the generality claim holds also for wildly different domain contexts, like personalized healthcare and public transportation.

2.4 MBSD frameworks using SysML

SysML is a modeling language that provides notation/syntax and semantics for the design of a system [105] (see Section 2.3). However, this language does not provide pragmatics, nor does it advise directly on the design process [286]; i.e. the language itself does not specify the design framework. MBSD frameworks that use specified language elements are necessary for the effective design of different systems [288, 284]. In particular, such frameworks can be characterized as "the collection of related processes, methods, and tools, used to support the discipline of systems engineering in a model-based/model-driven context" [62]. Popular ones are the INCOSE OOSEM [214, 166], the schema of Friedenthal et al. [72], SYSMOD [287] or even UAF [203]; since our framework is based on the three latter frameworks, we further describe them in the following sections.

2.4.1 System model as integrating framework

MBSD is challenging since it involves stakeholders (or specialists) from different domains, such as system designers, network architects, requirements engineers, simulation experts, etc. that have their own view of the system. A common canvas is needed that combines these multiple views (subsystem, mechanics, electrics/electronics, requirements, structure, etc.), enabling their communication and keeping the consistency of information. MBSD provides opportunities to capture and harmonize the information from multiple disciplines within an integrated digital model of the system [226, 73]. Specifically, a main MBSD target is the generation of such a model, used to describe complex system aspects across multiple views of disciplines and technical domains [71]. Often a system model generated with SysML is used as a central-placed model in design. "A central system model acting as single source of truth" [101] can capture system requirements and decisions that fulfill them at different levels of abstraction. In essence, the central model can be used from anyone participating to the system design process while it is consistent and able to

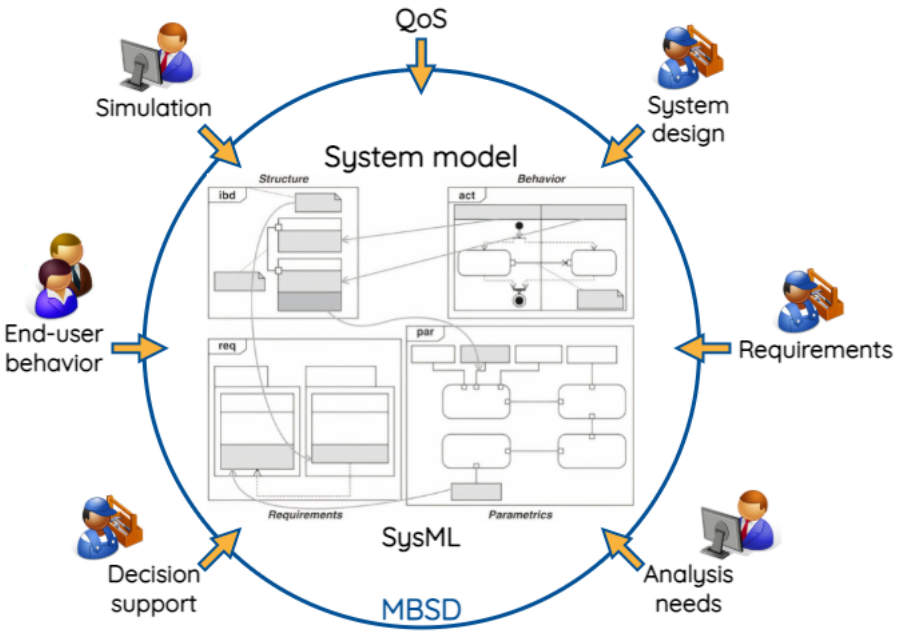


FIGURE 2.4: System model as an integrating framework.

reflect the changes from one perspective to other perspectives. Since dependencies across multiple disciplines are explicitly addressed in the model, a change made in one part of the model is automatically reflected in all the impacted parts of the model. This feature makes it possible to maintain consistent and up-to-date information in the integrated model. Figure 2.4 illustrates the "circle" of MBSD with its central system model where all different views, design activities, and aspects of the system must be combined and integrated into. We further analyze the system model as an integrating framework in Section 3.1.1, incorporating QoS into the circle of MBSD to manage it.

Since the central model serves all design activities [168], it should be technology-neutral, multi-layered, modular and composite, facilitating the integration of system sub-models; the system model has the objective to consider and analyze the system as a whole (e.g., behavior of the whole system which is the result of multiple subsystems and components acting together) while its sub-models may correspond to different perspectives, describing one certain aspect of a system in detail.

2.4.2 Systems Modeling Toolbox (SYSMOD)

According to Weilkiens [286], SYSMOD is a pragmatic, high-level, domain-agnostic, requirements-driven approach that is "used to collect and model the requirements of a system, and to derive a (system) design that meets these requirements". In essence, SYSMOD primarily uses SysML, and provides an overview of the system design process, offering a toolbox of design tasks with input and output artifacts (captured and recorded in a set of SysML diagrams), guidelines, and best practices [287]. Figure 2.5, presented in [91], depicts an overview of the SYSMOD design process that comprises five basic phases, starting from the identification of system stakeholders and the collection of requirements, and ending to the modeling/design of the structure of the system. Note that SYSMOD promotes an iterative design philosophy where activities within the process can be revisited in order to correct errors and improve the system.

The phases are the following:

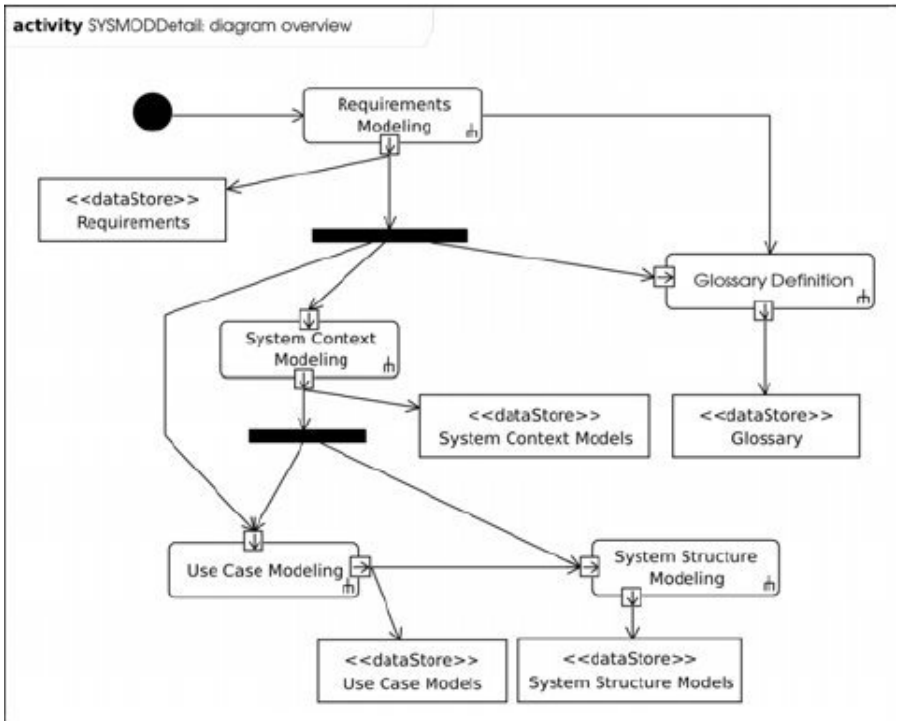


FIGURE 2.5: Overview of SYSMOD design process, as depicted in the work of [91].

(i) Description of the context of a project, where its goals are determined, the objectives of the system are analyzed, and all system stakeholders are identified.

(ii) Collection of requirements of the stakeholders via interviews, questionnaires, simple observation, or document analysis.

(iii) Collection of the domain's knowledge, i.e. the vocabulary containing the main activities and terms of the project, so that there are no misinterpretations, and anyone involved in the project is fully aware of the terminology that is used in it.

(iv) Modeling/design of the context of the system, where use cases are described, depicting functions of the system and how stakeholders interact with it.

(v) Modeling/design of the structure and different states of the system, where the system and its components are specified, and the system model is analyzed and tested.

Representing requirements in SYSMOD

SYSMOD concepts can be described with (and are related to) concepts of the SysML v1.5 [261] and onwards. Since SysML provides general elements for the design of systems, SYSMOD-specific elements are added to the SysML vocabulary via a set of extended stereotypes, grouped in the SYSMOD SysML profile [287]. In the following, we focus on the requirement-related extensions that depict how and which requirements are represented in SYSMOD approach.

According to Figure 2.6, the SYSMOD profile extends the SysML *Stakeholder* element, and the *Concerns* that a stakeholder may have; note that concerns are properties of the stakeholder element, and can be the source for specified requirements, related to system stakeholders. The

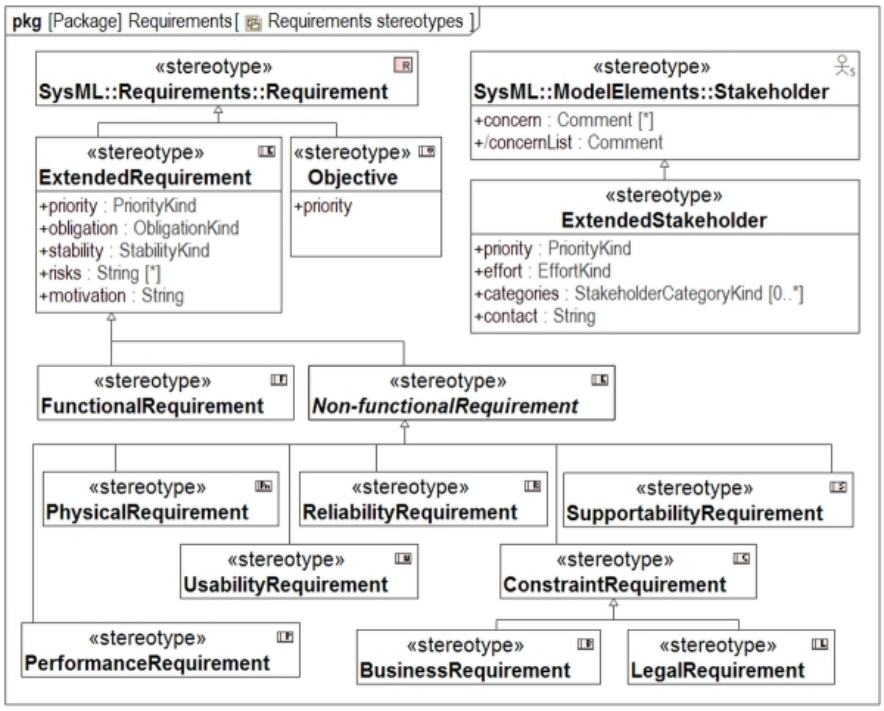


FIGURE 2.6: SYSMOD profile: Requirements taxonomy, presented in [287].

SYSMOD *ExtendedStakeholder* element prioritizes such stakeholders and adds *contact* information, and the *effort* accounted per stakeholder.

In addition, SYSMOD includes the concept of the *Objective*; by using the basic text-based SysML Requirement element, it recommends the way to describe the objective(s) of a requirement. Furthermore, SYSMOD recommends specific additional properties attached to a requirement, such as *priority* for the importance of a requirement, *obligation* to distinct optional or mandatory requirements, *stability* to depict any requirement change, *risk*, and *motivation* for the necessity of a requirement. Moreover, a categorization of different requirements in *Functional* and *Non-Functional* ones is provided in the SYSMOD profile (see Figure 2.6). Requirements can be grouped into various categories, e.g., based on the Functionality, Usability, Reliability, Performance, and Supportability (FURPS) model [89]; according to Weilkiens [286], "SysML does not dictate categories", thus, its extension mechanism can be used for the addition of system-specific requirement categories. SYSMOD divides non-functional requirements in multiple categories to cover different requirement types in finer granularity; such categories are *Physical*, *Usability*, *Reliability*, *Constraint*, *Performance*, and *Supportability*.

Challenges depicting requirements in SYSMOD

Having highlighted the representation of requirements in SYSMOD, we delve into the primary challenge that occurs while depicting them. SYSMOD is able to describe requirements on multiple levels of abstraction, providing a set of different requirement types, and leveraging the raw mechanisms and expressive capabilities of SysML. The latter are: (i) the capability of extending requirements, i.e. adding properties properties like priority, obligation, or risk, and describing them via text, and (ii) the evaluation of requirements via SysML parametric diagrams. However, in reality, there are requirements that need to be specified solely from a quantitative point of view, while being verifiable. Despite of its capabilities, SYSMOD does not address the issue of the description

of quantitative requirements, nor does it natively provide a toolset or additional capabilities for the proper description of such requirements or the categories that encapsulate them; these are outside the focus of that approach. In addition, the aforementioned mechanisms are inefficient and incapable of properly representing and evaluating quantitative requirements.

Possible extensions to support QoS

Our framework is fully compliant with SYSMOD by adopting its basic concepts. However, we provide an alternative approach to efficiently manage two of these concepts: (i) the concerns of a stakeholder, and (ii) a specific requirement type, i.e. quantitative requirements.

Regarding the first concept, in the context of our framework, concerns represent "the interests pertaining to a system's design, or operation, critical to one or more stakeholders", according to the Institute of Electrical and Electronics Engineers (IEEE) architecture group [102], and can be the source of system requirements. Thus, we needed to depict concerns in a way that is (i) useful to the system designer, (ii) clearly defined within the design model, and (iii) effectively verifiable; for this purpose, we treat them as SysML Requirements. Note that the aforementioned SYSMOD objectives concept can be defined in the form of these concerns.

Regarding the second concept, in this work, we focus more on representing quantitative requirements pertaining to QoS and its restrictions. We use SysML profiles that enable the specification and management of QoS characteristics of a system, as well as the design of the physical architecture of the system in order to verify QoS quantitative requirements. SYSMOD offers no explicit support for such requirements; they could in theory constitute a new category in a SYSMOD context, and they are compliant with the application of SYSMOD in different layers of abstraction. However, the question on whether properties, like SYSMOD's obligation, or risk, are required for the efficient description of requirements, still remains open.

Specifically, QoS requirements can be either functional or non-functional. They are typically quantitative; their focus is not so much to encapsulate the behavior of the system but rather to capture what the system provides to its users, service-wise. The latter should actually match user/stakeholder expectations and address related concerns. In order to evaluate these requirements one has to assess system outcomes in order to verify whether they satisfy the users (or not). This is why both simulations and stress-testing come into play in our framework. Such requirements need to be tested against arithmetic quantities and other system properties. However, simple adjustment of system properties is not sufficient, since there is a core need to measure outcomes and assess results related to provided system services, to verify whether they meet certain objectives. Thus, this is a different requirement category that we need to account for.

In a nutshell, we provide additional SysML mechanisms (i.e. extension mechanism) that facilitate the application of SYSMOD, bringing the design focus on QoS in general, quantitative QoS requirements in particular, and physical system architectures that verify QoS requirements.

2.4.3 Unified Architecture Framework (UAF)

Reichwein and Paredis [228] analyze the concept of Architecture Frameworks (AF), which define the structure and content (elements, data, etc.) of system architectures, and encapsulate best practices and guidelines to efficiently describe and design them. Moreover, in their work, different views of a system, depending on the activities and objectives of the system stakeholder(s) (e.g., requirements, functionality, etc.), are illustrated.

The first presented AF was the Zachman framework [297], followed by The Open Group Architecture Framework (TOGAF) [224], DoDAF [163], etc. The most recent one is the UAF [203]. It is used to create standardised and coherent complex system architectures (based on models),

independently from design methodologies [97], utilizing existing OMG standards, like UML or SysML as a modeling language. UAF is a standardized framework that can be applied to MBSD, facilitating the development of interoperable systems with traceability to requirements and across different views, using one integrated architecture model that enables impact, gap or engineering analysis, trade studies, or simulations [186]. UAF unifies the terminology across multiple frameworks, methodologies and different domains, providing common concepts and models. Note that this framework supports different tool vendors and can be implemented in most popular modeling tools, like NoMagic MagicDraw [178] and Catia CSM [32] from Dassault Systémes, International Business Machines (IBM) Rhapsody [104] or Parametric Technology Corporation (PTC) Integrity Modeler [303]. In addition, UAF is not applicable only to defense architectures, like DoDAF; it is flexible enough and its scope is generic, expanding to SoS architectures of any domain, including industry, enterprises, and Internet of Things (IoT) systems.

The specification of UAF comprises three basic components, i.e. the *UAF Grid*, the *UAF Domain Meta-Model (DMM)*, and the *UAF Profile (UAFP)*; these components are described in the following.

UAF Grid. It encodes multiple views of system stakeholders via a grid-type architecture, as depicted in Figure 2.7. The grid is composed of (i) horizontal rows that correspond to design domains (e.g., strategic, services, functional, requirements, etc.), based on different views of system stakeholders, and (ii) vertical columns that correspond to different types of models, i.e., architectural representations that describe the corresponding domains (e.g., taxonomy, constraints, etc.).

In a top-down reading of the grid (see Figure 2.7), UAF supports the following design domains (rows), based on the work of Hause et al. [97].

- *Metadata (Md)*: This domain captures and provides all meta-data and supporting information relevant to an entire system or enterprise architecture, e.g., meta-model, domain-specific profiles and extensions, views to be built, processes of the developing architecture, etc.
- *Strategic (St)*: The strategic domain refers to the process of managing capabilities (capability taxonomy, composition, dependencies, and evolution); note that a capability is a high-level specification of an enterprise's ability to execute a specified course of action [203].
- *Operational (Op)*: It illustrates the logical architecture of an enterprise, describing requirements, operational behavior, structure, and exchanges that are required to support (exhibit) capabilities.
- *Services (Sv)*: This domain refers to the description of services, service specifications, and required/provided service levels of these specifications; these are needed to support the Op domain, and exhibit a capability.
- *Personnel (Pr)*: The Pr domain enables an understanding of the human role, e.g., decisions of stakeholders, in systems/enterprise architectures. It provides a structured linkage from the system engineering community to the manpower, personnel, training, and human factors communities.
- *Resources (Rs)*: This domain captures a solution architecture that comprises resources, like software, artifacts, capability configurations, and natural resources, that implement operational requirements.
- *Security (Sc)*: It illustrates security assets, security constraints (e.g., policy, laws, and guidance), security controls, and measures, that are required to address specific security concerns.

	Taxonomy Tx	Structure Sr	Connectivity Cn	Processes Pr	States St	Interaction Scenarios Is	Information If	Parameters Pm	Constraints Ct	Roadmap Rm	Traceability Tr
Metadata Md	Metadata Taxonomy Md-Tx	Architecture Viewpoints [a] Md-Sr	Metadata Connectivity Md-Cn	Metadata Processes [a] Md-Pr	-	-	Conceptual Data Model, Logical Data Model, Physical schema, real world results	Environment Pm-En, Measurements Pm-Me	Metadata Constraints [a] Md-Ct		Metadata Traceability Md-Tr
Strategic St	Strategic Taxonomy St-Tx	Strategic Structure St-Sr	Strategic Connectivity St-Cn	-	Strategic States St-St	-			Strategic Constraints St-Ct	Strategic Deployment, St-Rm; Strategic Phasing St-Rm	Strategic Traceability St-Tr
Operational Op	Operational Taxonomy Op-Tx	Operational Structure Op-Sr	Operational Connectivity Op-Cn	Operational Processes Op-Pr	Operational States Op-St	Operational Interaction Scenarios Op-Is			Operational Constraints Op-Ct	-	-
Services Sv	Service Taxonomy Sv-Tx	Service Structure Sv-Sr	Service Connectivity Sv-Cn	Service Processes Sv-Pr	Service States Sv-St	Service Interaction Scenarios Sv-Is			Service Constraints Sv-Ct	Service Roadmap Sv-Rm	Service Traceability Sv-Tr
Personnel Pr	Personnel Taxonomy Pr-Tx	Personnel Structure Pr-Sr	Personnel Connectivity Pr-Cn	Personnel Processes Pr-Pr	Personnel States Pr-St	Personnel Interaction Scenarios Pr-Is			Competence, Drivers, Performance Pr-Ct	Personnel Availability, Personnel Evolution, Personnel Forecast Pr-Rm	Personnel Traceability Pr-Tr
Resources Rs	Resource Taxonomy Rs-Tx	Resource Structure Rs-Sr	Resource Connectivity Rs-Cn	Resource Processes Rs-Pr	Resource States Rs-St	Resource Interaction Scenarios Rs-Is			Resource Constraints Rs-Ct	Resource evolution, Resource forecast Rs-Rm	Resource Traceability Rs-Tr
Security Sc	Security Taxonomy Sc-Tx	Security Structure Sc-Sr	Security Connectivity Sc-Cn	Security Processes Sc-Pr	-	-			Security Constraints Sc-Ct	-	-
Projects Pj	Project Taxonomy Pj-Tx	Project Structure Pj-Sr	Project Connectivity Pj-Cn	Project Activity PJ-Pr	-	-			-	Project Roadmap Pj-Rm	Project Traceability Pj-Tr
Standards Sd	Standard Taxonomy Sd-Tx	Standards Structure Sd-Sr	-	-	-	-			-	Standards Roadmap Sr-Rm	Standards Traceability Sr-Tr
Actuals Resources Ar		Actual Resources Structure, Ar-Sr	Actual Resources Connectivity, Ar-Cn	Simulation [b]					Parametric Execution/ Evaluation [b]	-	-
Dictionary * Dc											
Summary & Overview SmOv											
Requirements Rq											

FIGURE 2.7: UAF view matrix (grid) [97].

- *Projects (Pj)*: This domain describes projects and project milestones, how those projects deliver capabilities, the organizations contributing to the projects, as well as the dependencies between projects.
- *Standards (Sd)*: It includes standards (technical, operational, business), applicable to an architecture, as set of rules governing the arrangement, interaction, and interdependence of solution elements and/or parts.
- *Actual Resources (Ar)*: It illustrates the expected or achieved actual resource configurations, the actual relationships between them, as well as the analysis, e.g. evaluation of different alternatives, what-if, trade-offs, validation and verification, on the actual resource configurations.
- *Dictionary (Dc)*: This is a view that aims to define all the elements that are used in an architecture, and it contains tables that show the definitions of terms and acronyms.
- *Summary & Overview (Sm-Ov)*: It provides executive-level summary information to allow quick reference and comparison among architectural descriptions.
- *Requirement (Rq)*: This view is used to represent requirements, their properties, as well as the relationships between them and/or to other UAF architectural elements (e.g., SysML satisfy, verify, refine, trace).

Traversing the UAF grid from left to right, the different view types or model kinds (UAF columns), are described in the following; note that these types can be mapped to specific SysML diagrams.

- *Taxonomy (Tx)*: This model kind presents all the elements as a specialization/generalization hierarchy, provides a text definition for each one, and references the source of the elements. The Tx can be mapped to a SysML BDD.
- *Structure (Sr)*: It describes the definitions of the dependencies, connections, and relationships (e.g., composition) between different elements. The Sr can be mapped to both SysML BDD and IBD.
- *Connectivity (Cn)*: Connectivity is mapped to a SysML IBD, describing the connections, relationships, and interactions between different elements (or parts of them).
- *Processes (Pr)*: This type captures activity-based behaviors and flows via SysML activity diagrams or Business Process Model & Notation (BPMN) [198] process diagrams; it describes activities, inputs/outputs of these activities, activity actions and flows between them.
- *States (St)*: It captures state-based behavior of an element within SysML state diagrams. It is a graphical representation of states of a structural element, and how it responds to various events and actions.
- *Interaction Scenarios (Is)*: It is mapped to sequence diagrams, expressing a time-ordered examination of the exchanges between participating elements as a result of a particular scenario.
- *Information (If)*: The If model kind addresses the information perspective on operational, service, and resource architectures. It allows analysis of an architecture's information and data definition aspect, without consideration of implementation specific issues.

- *Parameters (Pm)*: It shows measurable properties of an element in the physical world, as well as elements and relationships that are involved in defining the environments applicable to capability, operational concept or set of systems.
- *Constraints (Ct)*: This type details the measurements that set performance requirements constraining capabilities. It also defines the rules and conditions governing system behavior and structure within a SysML BDD and PD.
- *Roadmap (Rm)*: The roadmap addresses how elements in the architecture change over time (or over periods of time).
- *Traceability (Tr)*: Describes the mapping between elements in the architecture. This can be between different domains, viewpoints within domains, and structure and behaviors.

Every grid element at the intersection of a row and column, depicts a single view or viewpoint specification for describing and designing the system [57]. In essence, the UAF grid provides guidance for which SysML diagram type should be used to create a specific UAF view.

UAF Domain Meta-Model (DMM). This meta-model establishes the underlying foundational modeling/design constructs to be used in the system design; SysML is used to represent elements of the UAF meta-model graphically [269]. The UAF meta-model improves the ability to exchange architecture data between related tools which are UML-/SysML-based and tools that are based on other standards.

UAF Profile (UAFP). The UAFP enables the extraction of specified and custom models from an integrated architecture description [268]. The models describe a system from a set of stakeholders' concerns such as security or information through a set of predefined viewpoints and associated views. In essence, it is an implementation of the DMM that specifies how UAF views can be modeled using the SysML notation. Since the UAFP is an extension of SysML, it inherits the same notation (e.g., blocks, etc.). Additionally, UAF supports the concepts of SysML views and viewpoints, where a viewpoint specifies a method for building a specific view from model data dynamically.

According to [174], typically, UAF domains (see Figure 2.7) allow for a logical and systematic flow of architecting activities for an enterprise architecture; this flow begins with the Sm-Ov domain, where concerns and objectives drive a strategic plan to increase value to enterprise stakeholders. In St domain, the strategic plan deploys capabilities to address gaps and shortfalls. Capabilities are actualized by operational roles, activities, and performers in Op domain. Operational concepts are implemented through services (Sv domain), resources (Rs domain), and personnel (Pr domain). In turn, resources comply with standards (Sd domain). Risk and threats are mitigated through necessary security and protection controls via the Sc domain. Requirements, constraints, and concerns are understood and communicated to projects (Pj domain). Finally, plans deliver the resources according to project activities and milestones, while the resources are characterized and verified (in Ar domain).

There are cases where a subset of the UAF grid can be leveraged, in order to explore a specific design problem using MBSD, e.g., explore the design of a system or even a SoS under specific constraints, utilizing SysML models [268]. MBSD has systems engineers/designers and solution providers as its main targets; these targets are treated as stakeholders who have concerns for related UAF views (not all of the views) to design and model a system; note that these stakeholders

should be able to design the system using SysML, employing a single design tool to fully describe the system, rather than recurring to different external tools for representing different layers and views. MBSD of systems and SoS may naturally have implications in multiple domains, thus, an exhaustive collection of domains may need to be explored. However, the Metadata (Md), Resource (Rs) and Actual Resources (Ar) domains are the primary domains that should be explored, since system architecture design choices are depicted in them, and the aforementioned stakeholders have more interest (and concerns) in these specific domains [97].

Specifically, the Md is related to the system meta-model upon which the system architecture is based (which may result in a small tailoring of the full UAF meta-model); the Md-Taxonomy (Tx) model kind can be used, showing the definitions of all elements used within the system model, while the Md-Structure (Sr) shows the list of applicable system views.

The Rs domain represents the "solution" to the problem. Rs-Tx, Rs-Sr, and Rs-Connectivity (Cn) refer to structural views, Rs-Processes (Pr), Rs-States (St), and Rs-Interaction Scenarios (Is) represent behavioral views. The Rs-Sr domain defines the physical resources and references the resource structure, connectors and interfaces in a specific context. The Rs-Pr domain represents human and system functionality. The Rs-Constraints (Ct) domain defines limitations, constraints, and performance parameters for resources, their interactions, performed functions, and data. It also specifies traditional textual rules/non-functional requirements for the system. The addition of SysML parametrics to this view provide a computational means of defining resource constraints within a specific system context. An interesting research question is how may quality constraints be fitted in the Rs-Ct, in order to be taken into account during the construction of the solution of the problem in the Rs domain? We note that there is no adequate evidence on how this issue should be treated.

The Ar can also be used as a row of UAF for effective MBSD. The Ar-Sr domain describes the analysis, e.g. evaluation of different alternatives, trade-offs, etc., on the actual resource configurations as it provides a means to capture different solution architectures. The detailed analysis is carried out using the Rs-Ct view. Ar-Sr also shows the possible relationships between organizational resources. The key relationship is composition, i.e. one organizational resource being part of a parent organization. The Ar-Parameters (Pm) domain depicts the measurements of resources. It includes the definition of performance characteristics and measurements, as well as the identification of non-functional requirements.

Going back to quality constraints in system design, one should take into account that when utilizing UAF, the constraints view participating in a specific domain, e.g., the Rs domain, is related to neighboring cells of the Constraint (Ct) column. Thus, quality constraints in the Rs domain should be extracted from data/information that is already defined in the Ct column of the unified framework. In essence, service quality can be a constraint that restricts the system-under-study, and should be explored in all UAF domains. Similar to this, cost can also be considered a constraint under which a system operates. Thus, the exploitation of QoS and cost constraints should be treated in the Ct column and more specifically, should be transformed into Rs constraints when designing the system. Another interesting research question is how can we represent QoS and cost parameters utilizing SysML within the Ct column? This is an issue that is not explicitly, adequately or efficiently addressed by the framework itself or the UAFP/UAF DMM.

We note that a "tailored" version of the UAF grid is depicted in Figure 2.8, where the red-framed rows refer to the domains that may support MBSD, while the green-framed column(s) illustrate the view types that support quality (and cost) system constraints; all the other cells are covered since they are out of the system design scope.

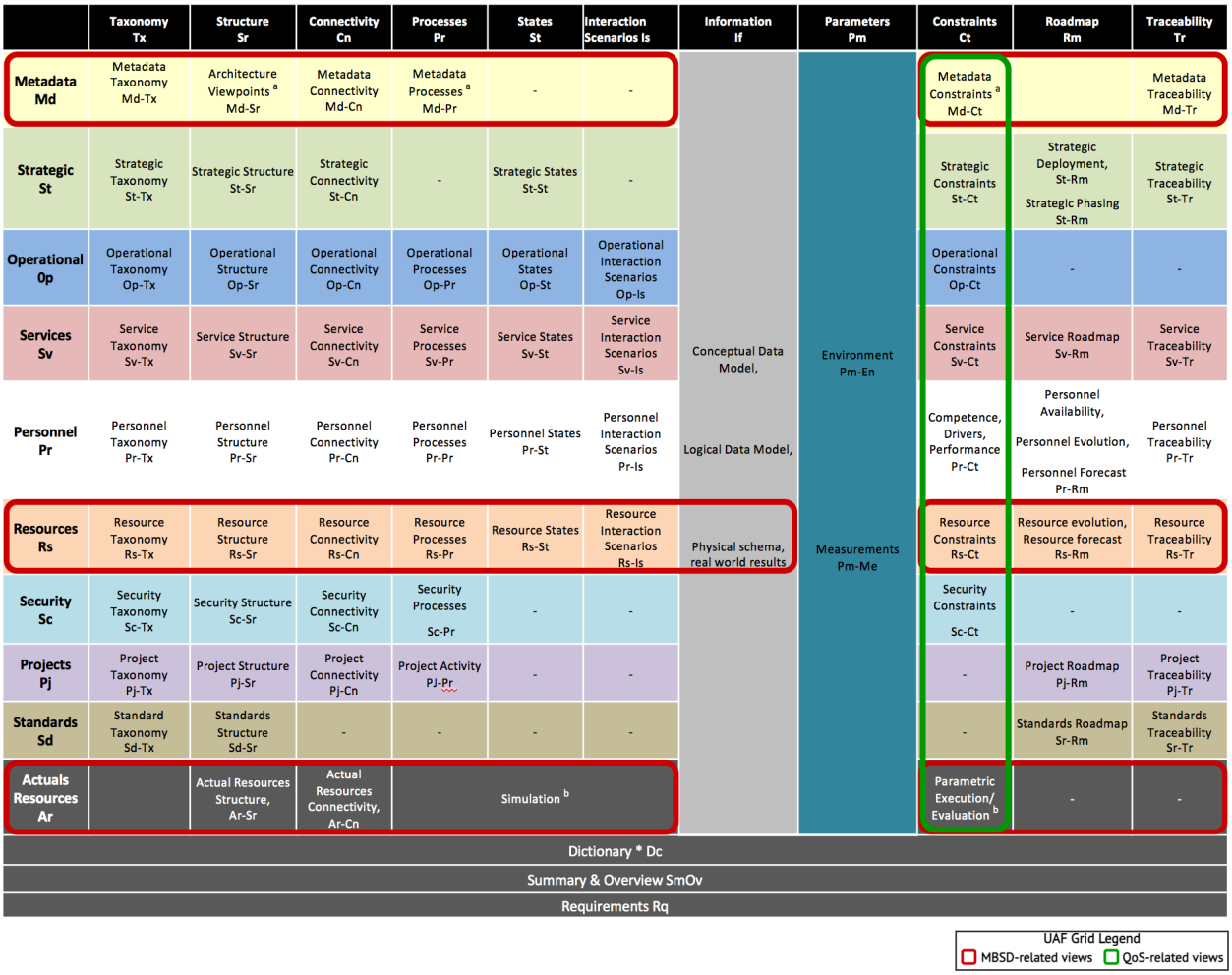


FIGURE 2.8: UAF views, supporting MBSD and quality characteristics.

2.5 Model transformations in MBSD

SysML is based on Meta-Object Facility (MOF) [218], a general modeling infrastructure that enables the definition of meta-models, as well as the conversion of SysML models to other modeling languages in a standardized manner. In order to support different MBSD activities, distinct aspects of the system model are required [72]. These aspects must be derived from the SysML system model and can be provided by transforming it into analytically executable models [128, 12, 48, 126], using standard transformation languages, like Query/View/Transformation (QVT) [155]. These models can be used to achieve the following: (i) study of dynamic system behaviors (e.g., by repetitive execution of simulations), (ii) validity of the system and verification of its proper operation (e.g., validating system requirements), (iii) improvement of system design (e.g., by adapting structural and other entities or functional system properties), and (iv) selection of the most appropriate design solution (e.g., based on the validation of requirements).

Query/View/Transformation (QVT). QVT, defined by OMG, is a standard for model transformation in the Model-Driven Architecture (MDA) [254]; in particular, it is a language for defining relations between a source and a target model, conforming to respective meta-models [202, 189]. The QVT specification has a hybrid declarative/imperative nature, with the declarative part being split into a two-level architecture: (i) a user-friendly *relations* meta-model and language which supports complex object pattern matching and object template creation, and (ii) a *core* meta-model and language that is defined using minimal extensions to Essential Meta-Object Facility (EMOF) [265] and OCL [229]. EMOF is the part of the MOF 2 specification that is employed for defining simple meta-models using simple concepts [8]. A transformation between candidate models is specified as a set of relations that must hold for the transformation to be successful. Using the relations transformation language we can transform the source model to a target model. To accomplish this, the two models should conform to MOF specification.

Discrete Event System Specification (DEVS). Discrete-Event Simulation (DES) is the process of codifying the behavior of a complex system as an ordered sequence of well-defined events [66]. In this context, an event comprises a specific change in the state of the system at a specific point in time. Discrete-event modeling is the process of depicting the behavior of a complex system as a series of well-defined and ordered events, and works well in virtually any process where there is variability, constrained or limited resources or complex system interactions.

Discrete Event System Specification (DEVS) is a modular and hierarchical formalism for modeling and analyzing general systems that can be DES [124]. The DEVS formalism describes a system as a mathematical expression using the set theory [298]. There are two kinds of models in DEVS: *atomic* and *coupled* models [133]. An atomic model defines the behavior of a system; it depicts the system as a set of input/output events, internal states, and phases along with specific behavior functions regarding the advancement of the simulation time, event consumption/production, and internal state/phase transitions. A coupled model is a composite model that consists of a set of atomic models, messages that simultaneously cross the connections/coupling paths between the atomic models, information or objects that travel along the couplings, and input/output ports [247]. In [123, 124], an integrated framework for utilizing existing SysML models and automatically producing executable discrete event simulation code is introduced. This approach utilizes MDA concepts. Although this approach is not simulation-specific, DEVS was employed, due to the similarities between SysML and DEVS, mainly in system structure description, and the mature, yet ongoing research on expressing executable DEVS models in a simulator-neutral manner [124]. DEVSys framework includes: (i) a SysML profile for DEVS, enabling integration of

simulation capabilities into SysML models, (ii) a meta-model for DEVS, allowing the utilization of MDA concepts and tools, (iii) a transformation of SysML models to DEVS models, using a standard model transformation language like QVT, (iv) the generation of DEVS executable code for a DEVS simulation environment with an eXtensible Markup Language (XML) interface.

2.6 Decision-making in MBSD

SysML has been proven to be expressive enough to model complex systems with heterogeneous components [292] and provides the necessary flexibility for tuning or extending native constructs using the profile mechanism [105]; however, it cannot support all design activities such as simulation, to explore the performance of a possible design solution. This limitation, already recognized in [72], has led to numerous approaches to integrate SysML models with external tools more appropriate for specific activities, mainly analytical models to explore performance [292, 276]. A prominent example of such efforts is the SysML2Modelica profile, converting SysML models to executable Modelica models [212].

An important design activity on par with the previous ones is decision-making. Although SysML is used to depict design decisions, decision-making capabilities should also be supported to assist the work of the designer [233]. There are works in literature that propose MBSD approaches based on SysML to enable effective decision-making during system design [132, 246]. Via such approaches, system designers are facilitated to reach alternative system solutions that result in the most preferred expected system design outcome. In [238], the authors have proposed a framework, applicable to decision-making within MBSD, where design alternatives based on system performance requirements can be evaluated. In the work of [232], a model-based framework based on (i) SysML to model system structure and requirements, and (ii) on the AltaRica Data Flow language to support model simulation, allows system designers and managers to make decisions regarding system maintenance. In [85], a method of systems modeling and partial simulation for long-term urban planning and development is proposed. This method acts as an integrated tool for efficient decision-making. Another example is the work of [27], where SysML is used for a web-based decision support tool, incorporated within the Framework for Assessing Cost and Technology (FACT). Based on such approaches, decision-making in MBSD has been explored; most works specify system requirements that are in turn transformed or used as input to external tools [80]. The decision-making problem is solved in these tools and results are produced, depicting that a designer can explore alternative design solutions and improve the system. However, these results should be incorporated back into the SysML models to facilitate requirement verification and become available to the system designer within the design process. This incorporation is either unclear or omitted entirely in these works.

Note that the Programming in Logic (Prolog) language has also been used in combination with SysML [244]. In most cases, it is used to provide rules for verifying the completeness and consistency of models or the overall system evaluation [34]. An alternative approach is presented in [19]. A non-SYSML model-based configuration approach that helps engineers create software configurations for Integrated Control Systems (ICS) via the SICStus Prolog constraint solver of the Swedish Institute of Computer Science (SICS), is proposed. A similar approach could be applied for the SysML language.

2.7 Research gap - Motivation

Adopting QoS during the study and design of a system gives an intelligible indication of the quality of the services, provided by the system, that its end-users are expected to gain when using/operating it. Therefore, focus is shifted from the characteristics and properties of the system to its adequacy to provide the expected QoS levels. QoS should be treated as a critical design requirement, enabling the system designer to natively include it in the design of a system.

In essence, the primary questions which remain unanswered in literature and we address in this work, are the following:

1. How can QoS be efficiently defined in a SysML system model? Based on Section 2.1, QoS should be matched with individual SysML requirements in components of system models. The focus of such QoS requirements is not so much to encapsulate the behavior of the system but rather to capture what the system provides to its users, service-wise. The latter should actually match user expectations and address related concerns. Note that QoS is directly tied to the acceptable cost that is required to achieve and maintain it. Thus, cost can also be considered a QoS parameter; we refer to the concept of QoS parameters, containing both QoS and cost.

2. With what framework is it possible to manage the representation, but also the verification of requirements, as a means of certifying expected QoS? QoS requirements can be either functional or non-functional and are characterized as quantitative. In order to evaluate them one has to assess system outcomes to verify whether they satisfy the system users (or not). This is why simulations, mathematical solvers or stress-testing should be integrated within the core system model via external tools. Such requirements need to be tested against arithmetic quantities and other system properties. However, simple adjustment of system properties is not sufficient, since there is a core need to measure outcomes and assess results related to provided system services, to verify whether they meet certain objectives.

The scope of this work is **QoS-aware MBSD using the SysML standard**. In particular, we focus on: (i)

- QoS awareness due to the criticality of taking QoS into account during system design (see Section 2.1),
- (ii) MBSD, since this is the most prominent approach for designing complex systems (see Section 2.2), and
- (iii) SysML as the basic modeling language/standard used in system design (see Section 2.3).

Thus, our framework is aligned with these three pillars, w.r.t. scope. The designer is enabled to operate (i.e., design a system) within an "integrated" scope where the design, QoS parameters, analysis, and even decision-making are included in a unified system model. The objective is to support effective QoS management within this scope; the designer should be enabled to explore different design solutions, make QoS-oriented design decisions, and directly verify the effects of these decisions on the system model without changing modeling environments. Note that other approaches, like UAF, follow the inverse approach where the modeling/design scope is fragmented into different viewpoints. Consequent management of design choices remains fragmented; it is very hard for a designer to have a spherical view of the system in contrast to our work.

In short, our primary objectives are the following. First, a way for the description and management of QoS in the form of requirements is needed. Second, analytical models are required to

facilitate the computation of QoS and check whether it can be fulfilled by system; these models should be integrated within the central system model, helping a system designer perform QoS analysis within the same environment where corresponding requirements are defined. Third, design decisions (i.e. proper system design configurations) that should verify QoS requirements should be proposed by the design environment, e.g., in the form of feedback. Finally, cost should be defined within the system model as system requirements, since it is considered a QoS parameter. Cost entities should also be integrated within system design to facilitate cost computation and analysis.

2.8 Application domains for QoS-aware MBSD

In this section, we present the literature relevant to the design (model-based or not) of two distinct system domains: (i) the railway (fixed-track) transportation domain that contains RTS, and (ii) the human-centric cyber-physical domain or CPHS. Each domain is characterized by its own design aspects and modeling challenges and objectives. We begin with RTS design works in Section 2.8.1 and analyze opened issues that need to be explored for this particular domain. We then describe CPHS design attempts in Section 2.8.2, while focus is shifted on a specific healthcare CPHS, i.e. the REMS (see Section 2.8.2), a system that is installed in the homes of patients to monitor their health condition. Issues that need to be dealt with during the design of CPHS are also presented.

2.8.1 Railway Transportation Systems (RTS) design

Analyzing the relevant literature, one can observe increased interest in studying the operation, structure and behavior of transportation systems. RTS are prominent examples of such systems, playing a key role in providing freedom in transportation, especially in highly urbanized environments. As such environments continuously grow and expand, so does the need for efficient and high quality transportation.

A few approaches propose the development of transportation systems in a generic, model-based manner, emphasizing peripheral issues, such as safety conditions, or environmental impact. In this context, a model-based safety architecture framework for capturing and sharing architectural knowledge of safety cases in safety-critical SoS is proposed in [243], utilizing SysML and the language's diagrams. This framework is applied to the Dutch high speed train lines. In [10], a comprehensive, interdisciplinary approach is proposed in order to analyze urban transportation and its environmental impacts. The wide range of spatial and temporal scales and processes involved, lead to a multi-tiered approach and a cascade of models to describe alternative urban development and transportation scenarios.

Using transportation systems modeling and design solutions, an engineer can define various aspects of each system. However, the vast majority of related works on RTS study tools rely heavily on basic simulation to analyze and evaluate the performance of such systems. A recent approach, Multi-Agent Transport Simulation Toolkit (MATSim), in [106], acts as an open-source agent-based framework for simulating large-scale transportation systems. The Railway Simulation System (RSS) is a general railway simulation platform, involving a railway network data management model in [296]. Moreover, the TRansportation ANalysis and SIMulation System (TRANSIMS) by [253], uses computational and analytical techniques to support models and simulation for transportation systems. In [6], a railway network operation, i.e. Abu Qir railway line in Alexandria city, is studied, optimized, and evaluated, using RailSys 3.0, a German capacity & capability modeling software (simulation program) used to support analysis on rail capacity. The OpenTrack of [191], acts as a user-friendly railroad network simulation program and simulates rail

system operations based on user-defined train, infrastructure, and timetable databases. [39] conducted an empirical study on Taiwan's High Speed Rail (HSR) line, presenting an effective multi-objective model, based on fuzzy mathematical programming to generate the best-compromise train service plan.

RTS are usually analyzed as DES [36], where trains travel according to a fixed or dynamic schedule and passengers are generated using appropriate probability distributions. Therefore, defined simulation models, using either custom or simulator-specific notation, are needed to support the design and operation of modern railway systems. In [90], a custom, event-driven simulator for multi-line metro systems and its application to the Santiago de Chile metropolitan rail network are presented. In [38], the DEVS formalism and a compliant simulator are used for the simulation of magnetically levitated train systems. SIMARAIL, a discrete event simulation environment is proposed in [235] to apply simulation-based timetable optimization for the Iranian railway network. Distributed constraint-based railway simulation is proposed in [241] to achieve efficient testing in large railway networks. In a similar context, a discrete event model for representing RTS, including their dynamic aspects, is proposed in [60], aiming at reducing the computational cost of simulation execution. Simulation performance is also addressed in [294], where the movement of a group of trains on a single railway line can be studied with less iterations and CPU time than the state of the art. In [175], the Simulation Management (SIMAN)/ARENA DES Tool is used in the operational planning of a RTS. Finally, the DEVS formalism is used in the design of LIBROS-II library for specifying railway transportation models in [109].

Issues to explore

According to [77], service quality provided by RTS is a quantitative index for the overall operation and performance of a service, facility, or means of transport (e.g., train), from the viewpoint of the passenger or the service provider. RTS service quality is layered in classes, in a standardized fashion ("F" to "A", according to [67]); the layering is represented by distinct levels, i.e. LoS [270], that is based on characteristics such as available space within trains/stations, waiting time for passengers, etc. This enables the characterization of a large range of structural and operational aspects of the system, in terms of the quality of the provided service, through a relatively small number of grades. Thus, study, development, management, evaluation, or even adjustment of the examined RTS can be effectively performed, based on the required, estimated and delivered LoS.

The main issue that the operators of RTS want to explore before or during system usage/operation, is how to manage and compute LoS for the provided transportation service; such a service can be the comfort of passengers during their transportation inside a train or while waiting in a station. In parallel, there is the need to determine how the computed LoS can be improved to provide services of higher quality to commuters with the current infrastructure; however, a point that is raised is how much will it cost to operate the system, while reaching a high LoS. In practice, passenger comfort is a QoS metric which RTS operators want to optimize under a reasonable budget.

2.8.2 Cyber-Physical Human Systems (CPHS) design

CPS is a special category of systems, comprising of parts that interact with the physical environment and others controlled by devices with computing and communication capabilities, therefore, delivering smart, controllable physical devices that are useful in production environments [108]. CPHS identify the necessity of human participants and their special role in the operation of these systems. Therefore, CPHS development approaches address the important aspects of human interference and, of course, safety [86].

In parallel, the huge and increasing number of small communication devices -also including sensing components- connected to the Internet resulted in the growth of applications based on interconnected peers interacting with the environment, i.e. IoT [45]. Nevertheless, these devices are usually owned and used by humans and operate in favor of them.

These parallel domains –CPHS and IoT– seem to converge, as identified in [35], resulting in many and with incremental importance applications requiring human participation and, at the same time, affecting human life. Medical applications is a domain where this evolution can be decisive for the effectiveness of monitoring and remediation activities.

However, human participants cannot be addressed as plain system components. System components are designed to fulfill specific functional requirements providing a system solution. Such components are selected or built to meet the functional and physical requirements and to be integrated in the developed product. This practice cannot be applied with human participants that exhibit diverse behavior and usually set their own requirements. Let alone the fact that they are a crucial factor for the acceptance and the success of the system.

These concerns have been identified and the role of the human participant has been studied [120, 301, 206, 182, 208, 240, 46, 263, 78, 255, 86]. However, human participants have been approached as a subject under study, overlooking their potential role as active participants in the design process. In this work, we propose that the involvement of human participants in the design of CPHS will increase system awareness and comfort regarding the system and, therefore, result in an increase in their motivation to participate and support the system acceptance and successful operation.

CPS leverage the capabilities of systems composed of mechanical devices and sensors that interact with the physical environment by introducing dynamic communication and control functions offered by computing devices. Such systems offer smart and adjustable production or other physical operation capabilities. The advancement of IoT and the technological leap in communication technologies and portable or wearable smart devices further promoted the application of CPS. Humans are important actors present in physical production environments and CPS often require control or monitoring input from adjacent human participants. Humans may unintentionally affect CPS in unexpected ways, affecting their prescribed operation and possibly causing severe danger for human life. Therefore, CPHS is defined as a special category of CPS, where the role of human actors is thoroughly studied and considered during the design process of CPS solutions. However, model-based CPS and IoT design approaches delivering efficient, effective and secure solutions –often utilizing the SysML– have not sufficiently addressed the distinctive properties of human participants and moreover their willingness, motivation and commitment to participate as integral parts of the system.

Industry 4.0 revolutionizes industrial manufacturing solutions by utilizing cutting-edge technology in innovative ways [120, 304]. CPS are defined and used as the main components in this scheme. The ultimate purpose of using cyber infrastructure (including sensing, computing and communication hardware and software) is to intelligently monitor (from physical to cyber) and control (from cyber to physical) objects in the physical world [108]. A system with a tight coupling of cyber and physical objects is called CPS. CPS engineering in general is a recurring process which integrates the dynamics of the physical processes with software and hardware, provides abstractions and offers modelling, design, and analysis techniques for the integrated whole [120]. When human beings are acting as an integral part of these systems, due to the actual interaction and participation within them, it is important to consider the human factor in the system development process [208] [58]. To this end, the terms of CPHS or Cyber-Human Systems (CHS) have emerged, comprising interconnected systems (computers, devices, and people) that work together to achieve the goals of the system, which ultimately are the goals set by human stakeholders [152] [169]. This integration of humans introduces a large number of challenges and problems to be

solved in order to achieve seamless and solid participation. Human participants, machines, and software systems are required to interact and align with each other in order to work together in an effective and robust way [255]. This interaction has introduced another common term, in the literature for CPHS, Human-in-the-Loop Cyber-Physical Systems (HiLCPS) [86, 263]. Such systems comprise a promising, yet challenging, class of applications with immense potential for impacting the daily lives of many people. They consist of a loop involving a human, an embedded system (the so-called cyber component), and the physical environment. The role of the embedded system is to augment a human's interaction with the physical world [240].

Modeling CPS is a challenging effort [120]. According to Lee [161], although key deterministic models such as differential equations, synchronous digital logic and single-threaded imperative programs have historically proven extremely useful during technology revolutions, CPS combine these models in such a way that determinism is not preserved. Moreover, besides modeling human characteristics and system-related properties, their concerns and conditions in order to accept to become a part of the system should be considered.

Conventional build-then-test practices made CPS not affordable to build. In response, a group of leading industrial organizations have joined the Society of Automotive Engineers (SAE) to define the SAE Architecture Analysis and Design Language (AADL) [64] (AS-5506 Standard), a rigorous and extensible foundation for model-based engineering analysis practices. AADL can facilitate lightweight and rigorous analyses of critical real-time factors such as performance, dependability, security, and data integrity. Additional established and custom analysis and specification techniques can be integrated into the engineering environment, developing a fully unified architecture model that makes it easier to build reliable systems. Therefore, AADL has been widely used in avionics and space areas.

The challenges raised up on how to dynamically test CPS models described in AADL and find design fault at the design phase to iterate and refine the model architecture are addressed in [302]. The properties of flow latency in CPS models are abstracted and translated into Push-Down Automata (PDA), in order to assess the flow latency with simulation. The case study of a pilotless aircraft cruise control system is used to prove the feasibility of dynamic model-based testing on model performances and achieve the architecture iteration and refining aim.

In the automotive industry synchronous reactive models are used by automotive suppliers to develop functionality delivered as Automotive Open System Architecture (AUTOSAR) components to system integrators (Original Equipment Manufacturer (OEM)). Integrators then generate a task implementation from runnables in AUTOSAR components and deploy tasks onto CPU cores, while preserving timing and resource constraints. An integrated synthesis flow is proposed in [52] to support the automotive production process in both sides of the supply chain. On the supplier side, from synchronous models, AUTOSAR runnables are generated, promoting reuse and easing the job of finding schedulable implementations. On the integrator side, the mapping of runnables onto tasks and allocation of tasks on cores, satisfying the timing constraints and memory efficiency requirements, are identified. Three system views are considered for this purpose: the Synchronous (or SR) view, the Runnables (or AUTOSAR) view, and the Task view (or Task model). The SR view and the Runnables view are the input and output of the first synthesis step, while the Runnables view and the Task model are the input and output of the second synthesis step.

INTO-CPS [159] is a project that aims to realise the goal of integrated tool chains for the collaborative and multidisciplinary engineering of dependable CPS. Challenges facing model-based CPS engineering are addressed, focusing on the semantic diversity of models, management of the large space of models and artefacts produced in CPS engineering, and the need to evaluate effectiveness in industrial settings. Semantically integrated multi-models, links to architectural

modelling, code generation and testing, and evaluation via industry-led studies are proposed to this end.

Recently, CPS development relies on service-oriented architectures. The constituent components are deployed according to their service behavior and are to be understood as loosely coupled and mostly independent. In [157], a workflow that combines contract-based and CPS model-based specifications with service orientation, and analyze the resulting model using fault injection to assess the dependability of the systems is proposed. Compositionality principles based on the contract specification make the analysis practical.

In [267], the authors adopt the concept of micro-service and describe a framework for manufacturing systems that has the cyber-physical micro-service as the key construct. The manufacturing plant processes are defined as compositions of primitive cyber-physical micro-services adopting either the orchestration or the choreography pattern. IoT technologies are used for system integration and model-driven engineering is utilized to semi-automate the development process for the industrial engineer.

Modeling and Analysis of Real-Time and Embedded systems (MARTE) UML profile, targeting modelling of real-time embedded systems was one of first to adopt SysML for modeling CPS [61]. Based on this initial work, others followed to establish Model-Based Systems Engineering (MBSE) and Model Based Safety Analysis (MBSA) integration using SysML. [42] is such an example, proposing to perform FMEA and FTA analyses early in the CPS design process in order to identify and mitigate the risks related to some safety critical structures and behaviours.

SysML4IoT is a framework for the design and analysis of IoT applications [45]. The framework is composed of a SysML profile based on the IoT-A Reference Model, and the SysML2NuSMV, a model-to-text translator that converts the model and QoS properties specified on it to be executed by NuSMV, a mature model checker that allows entering a system model comprising a number of communicating Finite State Machines (FSM) and automatically checks its properties specified as Computational Tree Logic (CTL) or Linear Temporal Logic (LTL) formulas. A proof of concept implementation that analyzes the QoS property of reliability in a Building Energy Conservation (BEC) IoT application.

In [44], the authors introduce a MBSE methodology for IoT application development, focusing on the design phase, which aims at helping organizations to develop IoT applications to fully achieve the benefits of this new paradigm. The approach comprises of a method and a tool, which are aligned with proven concepts from Systems Engineering and Software Architecture literature. Development process stakeholders (not human participants) and several viewpoints (virtual entity, functional, information, operational, deployment) are identified.

The challenge of developing IoT systems that adapt at run-time is addressed in [110]. A model-driven approach to ease the modeling and realization of adaptive IoT systems is proposed. First, IoT systems are modelled with SysML4IoT (an extension of SysML) to specify the system functions and adaptations, while environment information and its relationship with the system are modeled with a publish/subscribe paradigm. Second, based on the system design model, code is generated and deployed to the hardware platform of the system. These works confirm the potential of SysML to serve as a modeling language for model-based design of CPS.

Internet of Medical Things (IoMT)

The healthcare domain is one of the first to adopt IoT technologies. Specifically, in this field, the term of Internet of Medical Things (IoMT) [121] has emerged, comprising sensor-based devices integrated with cutting edge IoT mobile technologies, in order to deliver high-quality healthcare services, such as, the collection of a patient's medical data using smart wearable devices, the real-time remote monitoring of her medical status, the robust communication and interaction with

professional caregivers, etc. [94, 223]. The rapid breakthrough of IoMT technologies has drawn the attention of researchers in the direction of intelligent and efficient design for healthcare. The common scope of these advances is that they seamlessly connect sensors and devices to improve information delivery and the care-giving process in healthcare services [56]. One of the prominent example of such systems, increasing the level of independence of the patient, is the REMS. Such a system provides remote monitoring and diagnosis for the sensitive demographic subjects, dealing with the real-time diagnosis of medical incidents. Wearable sensors collect the medical information of patients and transmit them to healthcare professionals for further assessment and recommendations [156, 43].

The implementation of innovative sensing technologies such as wearable or implantable force sensors [41], or flexible e-skin pressure sensors [170] for monitoring human physiological signs provide the infrastructure to build appealing IoMT applications. Low cost and easily implementable IoT based health monitoring systems that offer accurate measurement of heart rate, blood pressure, glucose level and other health parameters of patients are also proposed in [179, 237]. Increasing demands of current IoMT-driven applications require leveraged solutions for effective health monitoring, decision making and data storage [63].

In such a context, the highly dynamic and real-time nature of IoMT systems can be enhanced by Artificial Intelligence (AI)-based methods which can endow IoMT systems with several degrees of intelligence [220] in decision making and provide a higher degree of robustness and accessibility [250]. Moreover, the advantages of AI-driven innovation can extend the boundaries of healthcare outside of hospital settings by transforming the hospital-centric to patient-centric ecosystem [63]. A wide range of AI approaches can be effectively adopted for the IoMT concept, as discussed in [68, 181, 153], in order to enhance decision making, anomaly detection, predictive risk monitoring, treatment support and so on.

All above efforts have contributed significant functional features to create more efficient IoMT systems from a technical point of view. As the adoption of IoMT systems is strongly related to the willingness of patients to employ and use medical devices [172], it is essential for such systems to ensure that patients are the focal point of all healthcare solutions, in order to achieve patient satisfaction and a greater adoption. To this end, recent studies have explored the involvement of the humans to healthcare systems. In [289] both patient and healthcare caregiver perceived facilitators were explored for successfully implementing healthcare applications. Specifically, a number of factors such as usability, flexibility, privacy and data availability were identified as critical barriers to the applications' adoption. Human concerns have also been identified in [190] for the integration of wearable health devices; convenience and possible functionalities were concerns essential to patients, while healthcare providers valued data reliability, device certification, and data transmission risks. The importance of convenience and usability of wearable devices for the elderly have been also recognized in [131, 262].

Patients, especially, become part of the IoMT system, as they often should wear or carry medical devices. As explored in [176], the capability of patients to effectively use medical devices (aka e-skills), and even more their attitude towards IoMT-based services (aka e-loyalty), primary affect their willingness to adopt such solutions. While there are works that attempt to explore the willingness of patient to adopt such services and identify their concerns after system implementation ([190], none of them indicates an effort to take patients' concerns into consideration while designing and implementing a IoMT-based service. In addition, there is not a comprehensive framework that takes into account the activities of patients and their active role in IoMT systems success.

Remote Elderly Monitoring System (REMS)

The REMS is a remote healthcare IoMT monitoring system [14] for elderly individuals. It acts as a platform, enabling the remote real-time monitoring of physiological signs and health parameters (like heart rate) of an elderly individual from professional health caregivers (e.g., doctors), reducing the number of times a patient has to travel for a regular check at the premises of a healthcare facility (e.g., a hospital) [29].

REMS utilizes smart wearable sensors to monitor elderly individuals, and offers real-time medical diagnosis and treatment, without the need to visit the healthcare facility premises [29]. Therefore, the patient is more involved in his own medical treatment; REMS can be considered a CPHS [183], where the human participant is an autonomous and unsupervised physical component that interacts with cocnerns.

Issues to explore

CPHS are composed of multiple mechanical and electronic components, physical processes, as well as human participants, that actively and dynamically interact with them, operating within the constraints of such systems. While several efforts on designing and implementing CPHS have been made in the past, these efforts either (i) do not take into account human users, or (ii) study users in isolation with respect to other cyber or physical elements. However, users, with their perception of the quality of system services, constitute a critical factor of the system. Keeping the human factor out of the loop during the system design/implementation can lead to poor quality and system performance regarding provided services, as perceived by the user, and it can also induce high costs for acquiring/operating the system [25, 28]. Taking human concerns as a starting base, the main issue to address is how a system designer can represent and integrate them within system design or before its implementation; these concerns should be transformed to critical system constraints that affect the system design.

Chapter 3

Integrated MBSD using SysML to serve QoS requirements

> "Here's the simple truth: you can't innovate on products without first innovating the way you build them."
>
> Alex Schleifer

This chapter is structured as follows.

In Section 3.1, we analyze our MBSD framework, heavily based on the work of Friedenthal et al. [72], where a SysML system model is the core, while (i) external models that accommodate QoS-related concepts, decision-making, and analysis aspects, are integrated to it (see Section 3.1.1), and (ii) basic SysML extensions are incorporated into it (see Section 3.1.2). The design activities, accommodated by this framework, form an iterative design workflow (depicted in Section 3.1.3) that can be applied to domain-specific SoS (described in Section 3.1.4).

In Section 3.2, we describe the external analysis models; such models can be either based on simulations (see Section 3.2.1), or closed-form mathematical equations and parametric relationships (see Section 3.2.2).

In Section 3.3, we also analyze the external decision model that provides a system designer with automated decision support capabilities; the designer creates the system model, which in turn is transformed into a decision model (see Section 3.3.1), created within an external decision-making tool (see Section 3.3.2).

In Section 3.4, we create a comprehensive QoS meta-model to serve QoS in SoS design, which is our primary focus; this meta-model is a structure of QoS-related SysML extensions (see Section 3.4.1, and validation rules/constraints that are imposed on its entities to ensure correctness and completeness (see Section 3.4.2).

In Section 3.5, we create a generic cost meta-model for the economic assessment of SoS; this meta-model comprises cost-specific but domain-independent SysML extensions for the description of cost entities (see Section 3.5.1), specializations of these entities (see Section 3.5.2), computations of costs incurred by these cost entities (see Section 3.5.3), cost computation functions that are readily available in an inventory (see Section 3.5.4), and validation rules/constraints that are imposed on the cost entities (see Section 3.5.5.

We conclude the chapter in Section 3.6, by presenting the environment where our integrated framework is implemented.

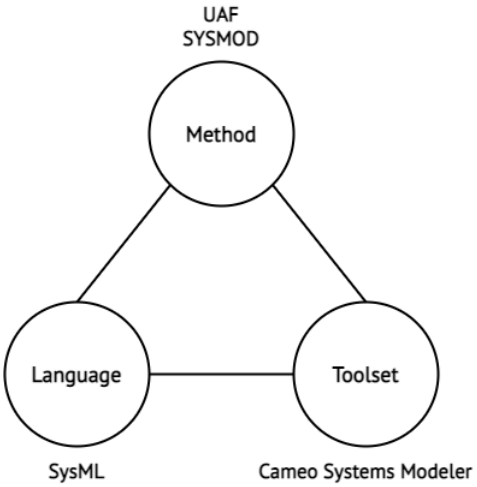


FIGURE 3.1: Requirements for effective MBSD.

3.1 Proposed QoS framework

According to Friedenthal et al. [72] and Weilkiens [290], effective MBSD requires the following: (i) a modeling language to represent and design systems, (ii) a method where design activities and views should be defined and artifacts should be delivered, and (iii) a toolset to implement the modeling language and the method. The combination of these requirements is illustrated in Figure 3.1 and provides a productive infrastructure for applying MBSD to various SoS. In this work, we address them as follows. W.r.t. the modeling language, we employ SysML as the de-facto standard that has been widely accepted by the scientific community, due to its effectiveness in designing complex systems [17]. W.r.t. the method, we build the following integrated MBSD framework to serve quality and QoS-related requirements during the design of complex systems, using the SYSMOD approach [287] and UAF framework [97] as basis. W.r.t. the toolset, we utilize standard-compliant tools, like CSM [37], that provide a suitable, customized MBSD environment, based on SysML, for system designers.

3.1.1 Overview

SoS design is typically based on MBSD approaches, implemented using SysML [95, 293, 37, 72]. Most approaches focus on isolated activities of the design process, such as simulation to explore the performance of a possible system design solution [292]. However, a well-defined framework which integrates structure, system requirements, analysis information, and decision-making to address various aspects of a system in a coherent, complete design model, is required [118]. Such a framework is at the core of the current work.

Friedenthal et al. [72] advocate the integration of SysML system models with external models, mainly analysis ones, to produce an integrated MBSD framework for system development, supporting specific design activities, and exploring system performance and behavior. This integrating framework should "provide a consistent source of the system specification, design, analysis, and verification information, while maintaining traceability and rationale for key system decisions". The system model as a framework for analysis and traceability is depicted in Figure 3.2.

In this work, we treat Friedenthal's schema as an initial base for MBSD, on which further design approaches can be built. However, our objective is to determine whether a certain system

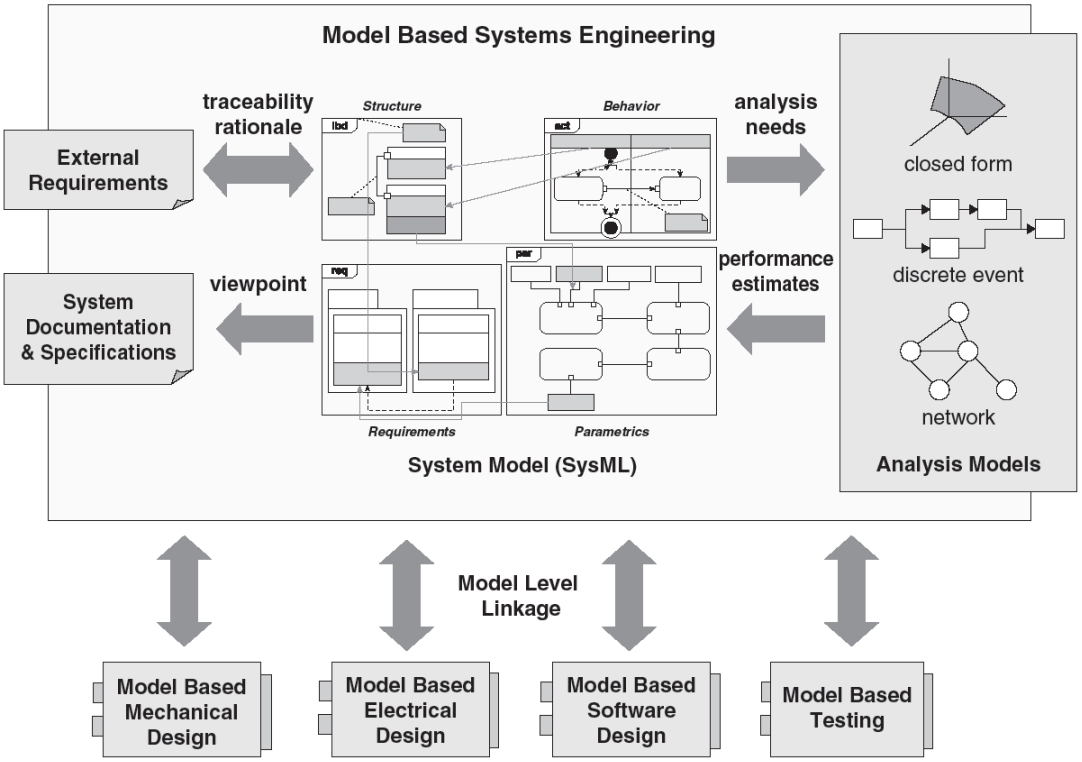


FIGURE 3.2: The system model as a framework for analysis and traceability, presented in the work of Friendethal et al. [72].

design can serve a specific quality of service, desired by the user(s) of the system. This complicates the rationale of the schema and requires certain extensions to be achieved. In particular, QoS requirements need to be specified, expressed, and integrated into the system model, before they can be evaluated. Analysis models are required in order to evaluate the actual performance of the system w.r.t. QoS, i.e. to assess whether associated requirements are satisfied or not. Moreover, satisfying certain QoS requirements in system design, requires a designer to decide upon different key aspects of the system, such as system configuration. Therefore, decision-making should be properly supported within an extensive –depending on the complexity of the system domain– decision space. Finally, serving QoS is directly tied to the associated costs pertaining to acquiring and/or operating the system. Cost models are required to be integrated into the system model to assess costs, and by extension, trade-offs between QoS and costs.

Therefore, in order to apply MBSD emphasizing in QoS, we propose and implement necessary extensions to Friedenthal's basic schema. These extensions are presented in Figure 3.3 as grey-colored rectangles:

(i) *QoS External Requirements*, to serve QoS within the system model through quality requirements. Technically, SysML should be extended to describe the notion of QoS as requirement(s) and associate it with the system's structural entities.

(ii) *Analysis Models*, to evaluate system performance and reliability aspects, after associated needs are fed into them. Specifically, in order for the designer to ascertain whether the aforementioned QoS requirements are satisfied or not, he should be facilitated to study the behavior of the system using various analysis mechanisms, such as simulation, stress-testing, etc. Analysis models should thus be integrated into the system model, while their results

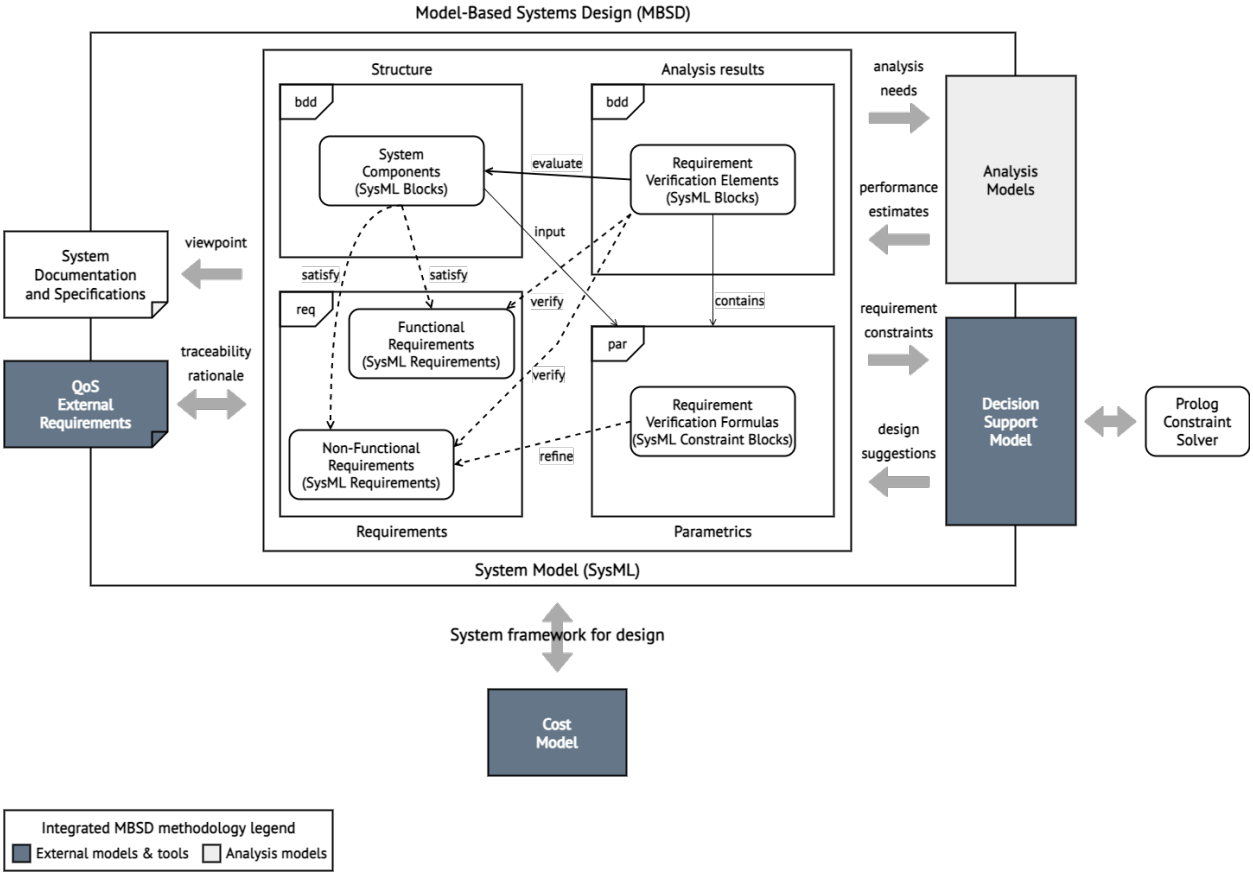


FIGURE 3.3: Integrated QoS-serving MBSD using SysML, utilizing system analysis, decision-making, and cost model.

(i.e. analysis outcomes) should be fed back into this model for QoS evaluation and QoS requirement verification.

(iii) *Decision Support Model (DSM)*, to manage the decision-making process of the designer upon designing the system. Specifically, it facilitates the designer to decide on the configuration of the system so that certain QoS requirements are satisfied.

(iv) *Cost Model*, to account for budgetary constraints of the system since QoS and cost are inter-dependent.

In order to adapt the integrated MBSD framework to specific domains of application, we introduce domain-specific design models, as shown in Figure 3.4. As indicative domains, we examine RTS and CPHS. Based on the figure, our analysis models are properly adapted per domain model; each domain may have different analysis needs. Specifically, on the one hand, an RTS, due to its scale, cannot be modeled by simple functions or closed-form formulas; thus, it needs repeated large-scale simulations to extract e.g., average transportation quality results pertaining to massive commuter populations in urban or other environments. On the other hand, a CPHS, which is a more personalized human-centric system, can be analyzed using customized functions or equations that encode specific needs of its human user(s); thus, a simple mathematical solver suffices to compute needed analysis results. These analysis models are visually depicted in Figure 3.4, by choosing the same respective colors for the analysis model that applies to a specific domain; the

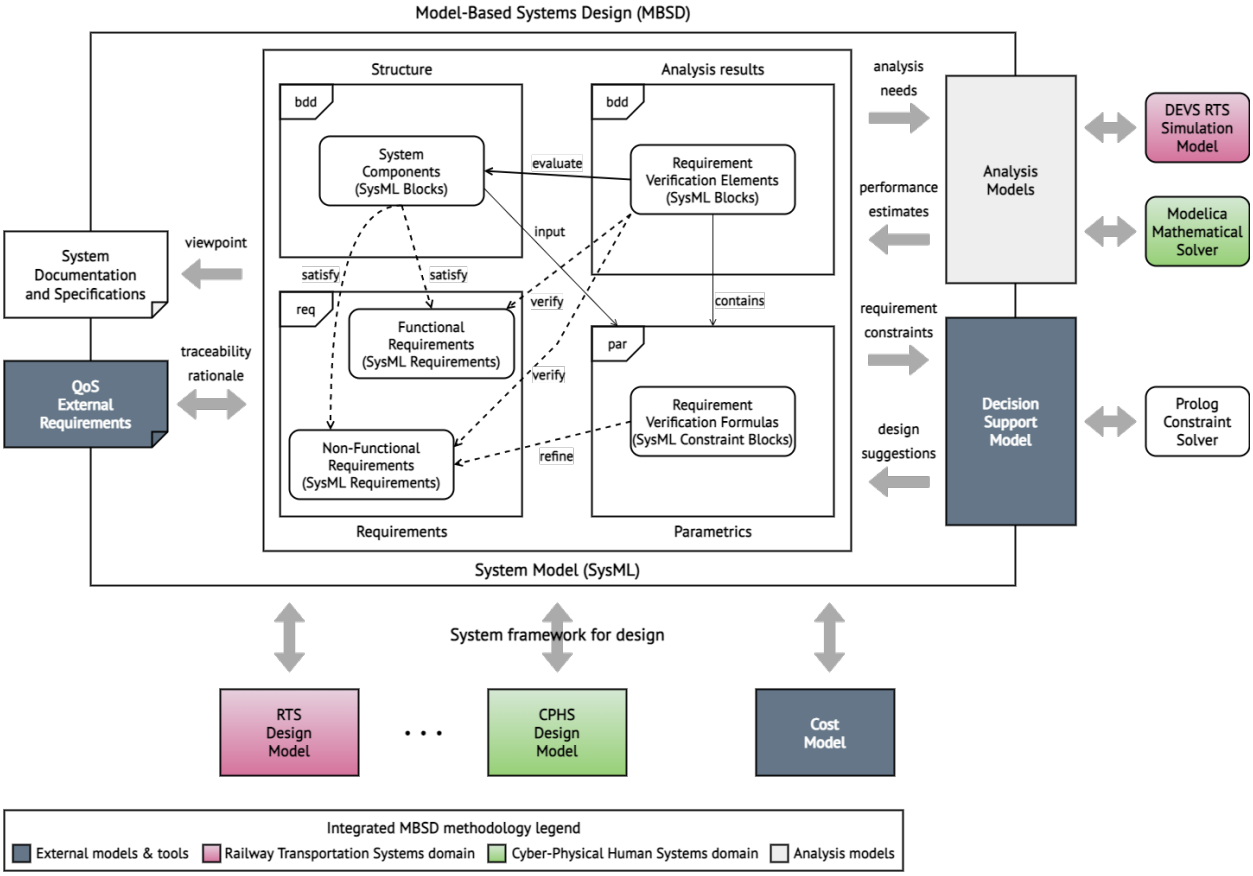


FIGURE 3.4: Integrated QoS-serving MBSD using SysML, adapted to application domains.

purple-colored *DEVS RTS Simulation Analysis Model* [124, 81] applies to the RTS domain, while the green-colored *Modelica Mathematical Solver* [74] supports CPHS. These analysis models are described in detail in Section 3.2.

According to Friedenthal et al. [72], the system model "is a primary artifact of MBSD and an integral part of the technical baseline of the system". Thus, the model is at the center of the MBSD process, covering different aspects of the system. In this work, we view the system model from our perspective, focusing on the following system aspects: (i) structure, (ii) requirements, (iii) parametric constraints, and (iv) analysis/performance results.

SysML is used to capture the system model [51, 72]; it provides specific diagrams (analyzed in Section 2.3.1) that cover the aforementioned aspects, such as the BDD, to present system structure and hierarchy, the RD, for capturing and tracing requirements and their relationships, or the PD [211], to describe requirement constraints and parametric relationships between system properties. In Figure 3.4, the system model is presented as an interconnected set of model entities; each entity is contained in a specific SysML diagram and represents a system aspect. In addition, these entities extend the primitive notion of SysML-provided entities, leveraging the flexibility of SysML w.r.t. tuning or extending native constructs; examples include, but are not limited to, generalizations of the SysML *Requirement* or extensions of the SysML *Block* entity. We delve into basic SysML extensions that comprise our system model in the following.

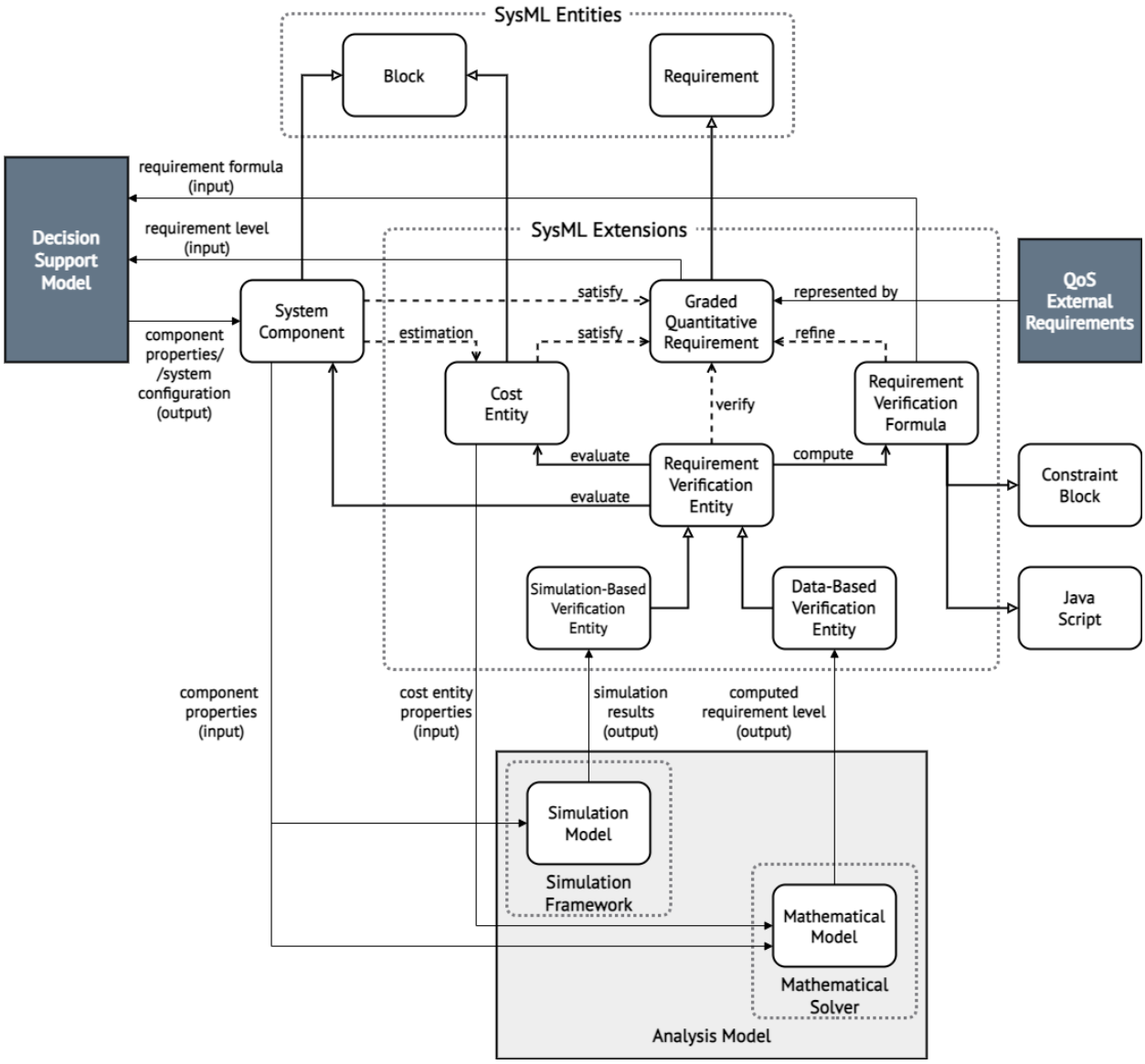


FIGURE 3.5: Basic SysML extensions.

3.1.2 Basic SysML extensions

The system model is presented in Figure 3.5 as an interconnected set of all our SysML-extended entities; the interaction of these entities with the integrated external models/tools (e.g., in terms of input, output) is also depicted.

System Component. It is defined as a SysML block to represent an actual/real entity of a system, building its structure; some components may be composite that comprise others, forming a hierarchy of this structure. Each system component is described by quantifiable characteristics that are represented by value properties [96] of the SysML block. A component is instantiated when specific sets of data values populate its value properties. Such data values (i) can be inserted directly by the system designer during system configuration and/or testing, or (ii) can be produced in an external tool, such as the decision support model, that is integrated with the system model, and then be incorporated into specific properties of system components. The component entity also includes operations as a feature to encapsulate the behavior that it may exhibit.

Graded Quantitative Requirement. It extends the SysML requirement (either an *Abstract Requirement*, or a *Requirement_Constraint_Block*, based on SysML version 1.5 [261]), inheriting predefined properties, such as a unique identifier and a human-readable descriptive statement. The primary goal of this extension is to serve QoS and address QoS concepts; note that the current version of SysML specification [199] does not directly address such concepts. Thus, the graded quantitative requirement is our way to directly relate text-based QoS external requirements to system design, integrating QoS to the system model. It is also worth mentioning that quality metrics are not supported with the standard SysML notation. We cover this limitation by describing the extended requirements in a qualitative fashion, using graded levels of the quality of system services; these levels should be achieved to satisfy the requirements. Based on Friedenthal et al. [72], "the traceability to a text-based requirement includes the design elements to satisfy it, other requirements derived from it, and a test case to verify it." We follow this traceability rationale including system components to satisfy quantitative requirements, sub-requirements to be contained to them or be derived from them, verification entities to verify them, and parametric constraints to refine them.

Requirement Verification Formula. This entity can extend the SysML *Constraint Block* stereotype or be a *Java Script*, produced by a programming language. It is described either as a mathematical equation of specific quantitative indicators or a logical expression of quality indicators (typically, an "if-then-else" formula). A formula refines a requirement, enriching the textual description of the requirement with a formal representation (mathematical or logical expression). In essence, the formula depicts the conditions that must be satisfied by value properties of system component(s) or other model entities, in order for the associated requirement to be verified. It may also be used to compute the proper values of value properties of related system components in order to satisfy requirements.

Analysis models are primarily used in our framework to evaluate quantitative requirements. As described in Section 3.1.1, analysis models are adapted per domain model, while each domain may have different analysis needs. In this work, we focus on two kinds of such models: (i) dynamic, that specify how the state of a system changes over time (i.e. discrete-event simulation), and (ii) mathematical, that represent a system in terms of mathematical equations which specify parametric values and relationships between them.

In terms of events simulation, an external simulation framework is used. DEVS [124, 81] is such a framework that provides a high-level representation of the system; a simulation model is generated by the system model and is executed. When simulation results are generated, they are fed back to the system model and are integrated into appropriate model entities.

Regarding mathematical analysis models, constraint formulas are input to an external mathematical solver, where they are executed, and the computed results are returned back to the system model; OpenModelica [205] is such a core solver that can model mathematical behaviour.

Both analysis models are described in detail in Section 3.2.

Requirement Verification Entity. This is defined as a SysML block and acts as a test/verification case [72] to support the verification of quantitative requirements. In our framework, the primary goal of this entity is to store the results that are produced by external analysis models, i.e. values that are computed either by simulation or a mathematical solver. These values can in turn be checked against requirements, leading to the requirements' verification outcome. Note that the verification entity is specialized in a *Simulation-Based Verification Entity*, where simulation-computed results are stored, or a *Data-Based Verification Entity*, where the computed outputs of mathematical equations are stored.

Cost Entity. It is an extension of the SysML block and is used for the financial assessment of a system component and/or the whole system. This entity includes value properties that are related to costs (e.g., the price of a system component, currency like Euro, etc.) and corresponds

to discrete cost categories such as capital or operational. A cost entity should satisfy associated cost-related quantitative requirements (e.g., requirements which describe that the value of an X system component should not exceed the amount of Y Euro), leading to the economic evaluation of the system. Note that we propose the creation of separate cost entities, instead of overloading a system component as a single entity that combines both structural and cost characteristics. Through this decoupling, a clearer, more accurate, and extensible system model can be defined, where the different perspectives, i.e. structural and economical, are distinguished; a system designer can specify a system component with its structural properties, and connect it with domain-independent, separate cost entities that may hold cost-specific properties of the component. Our cost perspective, comprising the definition and evaluation of specific cost entities within SysML models, is analyzed in Section 3.5.

Decision Support Model. As described in Section 3.1.1, the system model is integrated with an external decision model, the DSM. The latter model enhances the decision-making capabilities of the system designer during system design; in essence, it enables the designer to make system design decisions on the configuration of complex SysML SoS, abstracting away the underlying complexity. Specifically, it facilitates the automated exploration of alternative system solutions and the final achievement of a satisfactory system configuration solution that meets the defined quantitative requirements. It also provides automated suggestions/recommendations to the designer for efficient system design. Quantitative requirements and requirement constraints (i.e. computation formulas) are input to the external decision model; they are transformed into rules within a Prolog-based [24] decision model. This model is then used to generate system design configurations, i.e. provide acceptable combinations of data values of components' properties, conforming to quantitative requirements. The produced results are automatically incorporated back into the system model and are presented to the system designer. Note that a full (all requirements are satisfied) or a partial (some requirements are satisfied) solution can be reached -if feasible-. The automated decision-making process, including the transformation of SysML model to decision model, is described in detail in Section 3.3.

3.1.3 Design workflow

The system model, as the core artifact of MBSD, is at the center of the process, within a design environment. External models/tools are integrated with it, enriching it with additional design capabilities, like simulation and/or decision support. The system designer performs four distinct design activities, operating only within the design environment [147]. Specifically, the designer is enabled to: (i) define the SysML system model, satisfying given structural or other specifications, (ii) integrate quality and economical features into the system model through the definition of quantitative QoS requirements, (iii) analyze the system model using formal methods, such as event simulation or a mathematical solver, via the verification (or not) of the requirements, and (iv) explore and decide on alternative system design configurations and solutions that can cover the defined requirements; this can be applied as an iterative "design, analyze, decide" process during design in a specific system domain. We represent the aforementioned activities as sequential stages, according to Figure 3.6. In each stage the designer (a) provides input features and/or models that comprise the overall SysML system model, and (b) exploits artifacts from it; the stages are described in the following.

Stage 1: Define system structure (grey-colored). The system designer constructs the initial system model, specifying basic components that comprise the system, in an abstract fashion (i.e. simple declaration of system components along with value properties and/or their operations; no instantiation). At this stage, the abstract system components model, containing the structure and hierarchy of the system, is created as a SysML BDD.

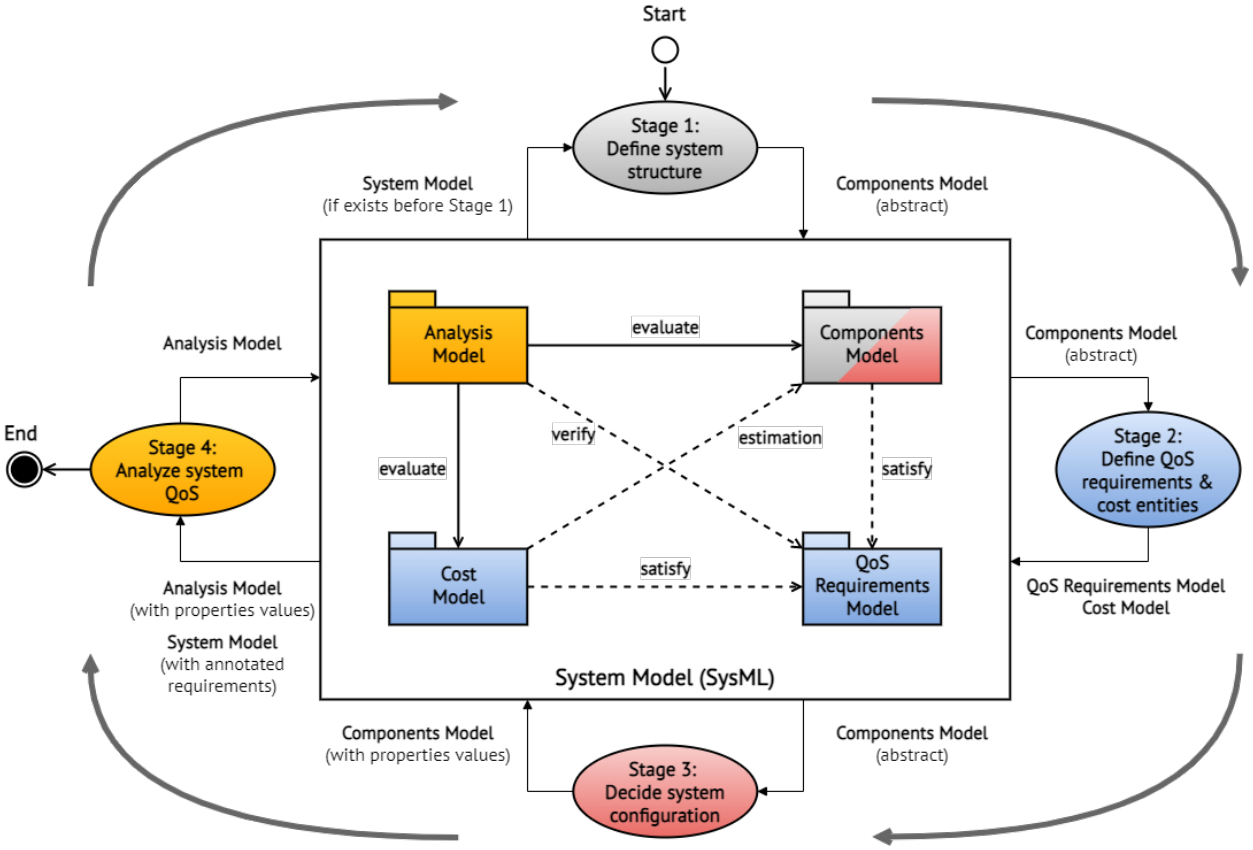


FIGURE 3.6: Proposed end-to-end MBSD workflow.

Stage 2: Define (a) QoS requirements, and (b) cost entities (blue-colored). This is a crucial stage where the designer integrates QoS and cost aspects into the system model. To serve QoS, quantitative requirements are specified within a SysML RD. In turn, the designer provides an abstract cost-related model that contains cost-specific entities; this model is connected to the components model, since system components are estimated by respective cost entities, that are used to measure and hold the expenses of the components. To fully cover cost, the requirements model is enriched with cost-related quantitative requirements.

Stage 3: Decide system configuration(s) (red-colored). In this stage, the designer provides data value ranges for all value properties of the defined system components. These values represent alternative configuration sets of the real system. Different combinations of the values represent different configurations of the system. In this stage, the external decision model comes into play, facilitating the designer to choose the component property values that correspond to the most satisfactory system design. These values may refer to the solution where all defined requirements are satisfied (full solution), or the solution where only some requirements may be satisfied (partial solution).

Stage 4: Analyze system QoS (yellow-colored). Via the analysis of system configurations, the system can be evaluated. The integration of the system model with analysis models, enables the designer to check the correctness and performance of the system design, as well as evaluate the system, assessing the delivered service quality. In case QoS is not satisfactory, the designer can explore alternative system designs, traversing other stages, in order to reach a satisfactory solution. Note that when QoS requirements cannot be fulfilled by the system configuration and design, they are properly indicated within the modeling environment so that the designer can focus on improving them.

Overall, stages 1 and 2 may be considered as the "model entities definition" stages, i.e. the designer defines structure, QoS requirements, cost entities, as well as verification entities. Stages 3 and 4 can be characterized as "properties' population" stages, since they allow the designer to populate these entities with data values, analyzing the system and deciding on an efficient design solution.

3.1.4 Application of proposed framework to specific domains - - Creating domain profiles

Profiles are used to extend and customize SysML and can support a wide range of system modeling domains. In this work, we construct two custom SysML profiles, namely the SysML QoS profile that encodes quality, and the SysML cost profile that encodes our cost perspective. Note that respective QoS and cost meta-models are first defined that capture abstractions and relationships of such concepts; our profiles customize/extend the constructs of these meta-models to adapt them for different purposes. Both profiles are generic, customizable and domain-agnostic, enabling the following: (i) they can be reused across multiple system domains, adjusted to their particular intricacies (e.g., different system design objectives), and (ii) they can be combined with domain-specific profiles which may encode structure, performance, behaviour or other extended system capabilities. The application of these profiles to a system model –or even another profile–, enriches it with quality, analysis and economical features/characteristics, making it more accurate and complete.

Regarding the QoS profile, it facilitates a system designer to incorporate QoS into system models, employing QoS-specific entities (described in Section 3.1.2); such entities are defined within

the QoS profile. Specifically, it primarily contains (i) the graded QoS quantitative requirement entity as the stereotype of the SysML requirement, (ii) the requirement verification entity as stereotype of the SysML block, and (iii) the requirement verification formula entity as a SysML constraint block stereotype. In Section 3.4, we describe our generic QoS meta-model, whose concepts are transformed into the respective profile.

As far as the cost profile is concerned, it allows the system designer to integrate cost-related information into a system model, and estimate the expected incurring costs of services or system components. It includes (i) general cost entities to cover all the financial aspects of a system, such as Capital Expenditures (CapEx) and/or -current- Operating Expenses (OpEx), and (ii) functions that allow the automated computation of financial figures. The model entities of the profile are a direct transformation from the cost meta-model constructs; this generic meta-model is analyzed in Section 3.5.

In this work, the aforementioned SysML profiles are applied to two distinct domains: (i) railway (fixed-track) transportation domain that contains RTS and (ii) human-centric cyber-physical domain or CPHS. Each domain is characterized by its own design aspects and modeling challenges, specific requirements and objectives, as well as distinct structural, quality, economical and analysis characteristics. For both domains, a corresponding domain-specific profile has been created that primarily includes entities for the exact description of system structure. That is, a RTS profile comprises declarations of trains, lines, or stations [148], while a CPHS profile consists of small devices like sensors or actuators [45] or even the human participant [145, 150]. In addition, each profile contains QoS requirements as specific conditions that must be satisfied by the corresponding system that are associated with the structural system entities. These profiles are fully described in Sections 4 and 5, where our QoS-aware SysML-based MBSD framework is applied to RTS and CPHS.

In the following, we outline the specific characteristics of both domains.

(i) **RTS operation based on the spatial comfort of commuters.** Such systems should be designed, analyzed, improved and operated by offering high-quality and efficient transportation services to citizens. The EN 13816 Standard [59], contains QoS standards in public transportation, focusing on eight criteria, i.e. availability, accessibility, information, time, customer care, comfort, security, and environmental impact. In this work, we focus on the specification and evaluation of the quality of the comfort of commuters while they wait at stations or inside trains. This quality is classified into service levels using the concept of LoS [67]; specifically, LoS acts as a quantitative index for the overall performance of a service (here, comfort during transportation) from the viewpoint of the service operator/provider. Employing a relatively small number of LoS grades, the service operator can characterize a wide range of structural and operational system aspects. In practice, this process is implemented by comparing the estimated achievable LoS (for example, LoS that is yielded by modeling and simulation) with the required LoS (e.g., defined by the operator). Deviations may drive re-adjustments and potential improvement of the considered system. In parallel, an economic analysis can be performed for the operating RTS costs (i.e. OpEx) that change as the best LoS is achieved.

(ii) **Engaging human participants in CPHS design taking into account their concerns and quality requirements.** Human participants of a CPHS should be involved from the early stages of the system development process, namely during the system design activity. This involvement is expected to promote the identification of human concerns, while the model-based nature of our framework facilitates the definition of human concerns, their correlation to system components, and the designation of critical aspects of the system. The participants

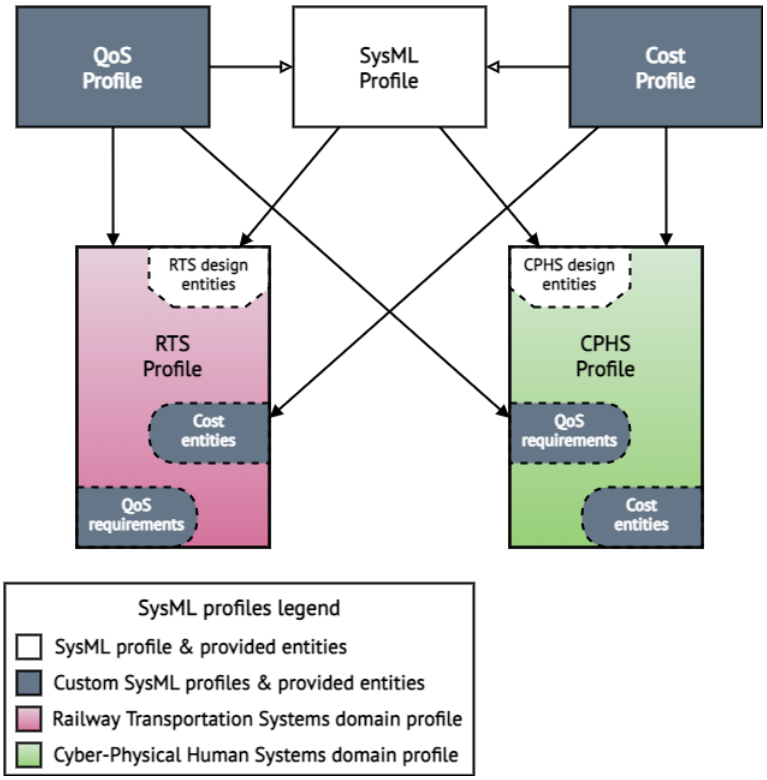


FIGURE 3.7: SysML profiles.

can dive into system details via specific views (e.g., human and/or system view) and modeling constructs, depicting key performance indicators, in order to help them understand what is feasible and how the system can be improved. Well-known capabilities of MBSD [42, 113, 127], like system model validation, automated requirements verification, and traceability can also be utilized. Overall, the application of the proposed framework to a CPHS, aims at (i) understanding human participant quality and budgetary requirements, and (ii) satisfying them with an appropriate system design through a process that continuously develops and maintains the bonds between the human participant and the system. Therefore, humans are expected to establish deep awareness of the CPHS and increased willingness to participate when it is rolled out.

Figure 3.7 shows the relationships between all aforementioned profiles, where the integration of quality (provided by the SysML QoS profile) and cost (provided by the SysML cost profile) characteristics in the RTS and CPHS profiles and fields, is observed.

Specifically, the domain-specific profiles utilize the following.

(i) Basic modular units like the SysML block, provided by the general SysML profile [204]; these are suitable to display design/structural entities in each domain.

(ii) QoS-related requirement and requirement verification entities from our QoS profile; these can be used to describe comfort LoS in RTS or concerns of human participants (for the QoS perceived by them) in a CPHS.

(iii) Cost-related entities from our cost profile; these can be applied for the financial evaluation of each domain, i.e. operating costs (i.e. OpEx) in a RTS, or capital/acquisition expenditures (i.e. CapEx) in a CPHS.

Having all required profiles applied to a system model, the designer can design the system, performing the following.

- Define system components (based on structural entities, applying SysML and domain-specific profiles).
- Define system QoS/cost requirements and associate them to the system components (QoS entities using QoS and domain-specific profiles).
- Estimate cost (cost entities using cost profile).

In combination with analytical and decision-support external models, the design capabilities of designers are enhanced, enabling them to evaluate the system as follows:

- Verify that the system design and configuration satisfies requirements (analysis models).
- Assess the impact of unsatisfied requirements, and reconfigure the system design based on automated suggestions/recommendations (decision support model).

3.2 Analysis models

Having defined structure and requirements, a system designer may need to perform system performance analysis. To this end, analysis models are employed to answer a specific technical performance question or provide outputs for a particular system design configuration and decision. Such analysis models can be either based on simulations, which represent how properties vary over time, using a system of dynamic equations, as depicted in Section 3.2.1, or close-form mathematical equations and parametric relationships, as shown in Section 3.2.2.

3.2.1 Simulation-based testing during MBSD

The objective of an external simulation framework is to employ simulators to evaluate the performance of systems that are highly dynamic and cannot be modeled via closed-form solutions [124]; on the contrary, complex computations are required to model the behavior of such systems, and capture their time-changing state. In our work, the performance of a system is expressed in terms of levels of quality of the services that can be delivered by the system; simulators are employed to calculate these levels over time, for different inputs supplied to the system.

The simulation process can be depicted as a workflow that begins with the definition of the system model, proceeds to simulation, and yields results back to the starting model. This is described in Figure 3.8. System models that are used in the process are depicted in the center, while the three successive layers represent the following: (i) operations that are performed on the models (inner layer), (ii) software tools that execute these operations (intermediate layer), and (iii) parts of the process that require manual effort from the system designer or are performed in an automated fashion (outer layer). Note that the operations of the inner layer are performed in an iterative manner, indicating that the designer can design and test different system configurations as many times as needed. Processing of the models is performed using only standard modeling languages like SysML, meta-modeling concepts based on MOF, and model transformation techniques like QVT, based on MDA [136], to ensure model and tool interoperability. Thus, our simulation process is generalizable and can be applied in various SoS and multiple domains, modeling environments, and simulation frameworks.

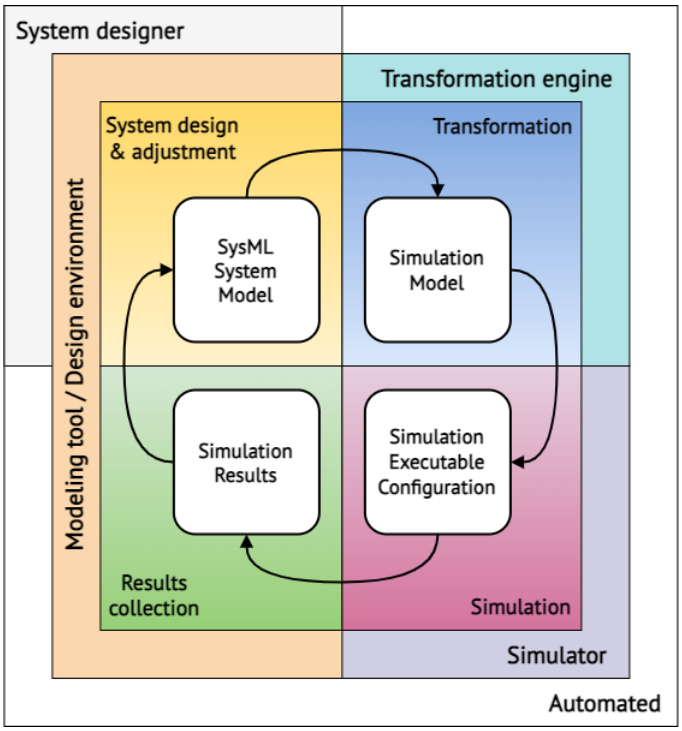


FIGURE 3.8: Iterative process of simulation-based testing during MBSD.

Specifically, the system designer (grey-colored quarter of outer layer) interacts with a modeling tool in a single design environment (orange-colored half of intermediate layer) by developing the core SysML system model. In turn, three tasks are performed, i.e. (i) the system model is transformed to a simulation model, via a QVT-compliant model transformation tool (blue-colored quarter of intermediate layer), (ii) simulation code is generated and executed with the utilization of simulation library components, and (iii) simulation results are produced and are incorporated back into the SysML model; these tasks are automated and transparent to the designer during the design and simulation-based testing of the model.

To enable simulation of simulation-agnostic, domain-specific SysML system models, the DEVSys framework, utilizing domain-specific libraries, is selected in this work [127]. In Figure 3.9, this framework is illustrated, comprising several components; the yellow-colored blocks indicate available technologies/tools (i.e. infrastructure), the green-colored blocks are the DEVS-related parts of the DEVSys framework [125, 194], the blue-colored blocks indicate domain-specific artifacts, and the white blocks refer to models, instances or files.

We follow a top-down approach to describe the simulation process based on this framework. The starting point is SysML system modeling, enhanced with domain-specific modeling aids, i.e. SysML profiles that are tailored to a specific domain (SysML4[Domain]); additional tools (i.e. plugins) may also be required to further facilitate the system modeling. In turn, the domain-specific models are transformed to DEVS models via QVT; such transformations can be executed in compliant tools, like mediniQVT [69] based on the Eclipse environment [69]. Next, high-level DEVS simulation models based on the DEVS MOF meta-model, are generated. Following this generation, Extensible Stylesheet Language Transformations (XSLT) model-to-text syntactic transformations are performed, converting DEVS simulation models to specific DEVS simulator code. Here, a set of available DEVS simulation library components, implementing the simulation behavior of specific elements, is identified and transformed into executable simulation configuration code.

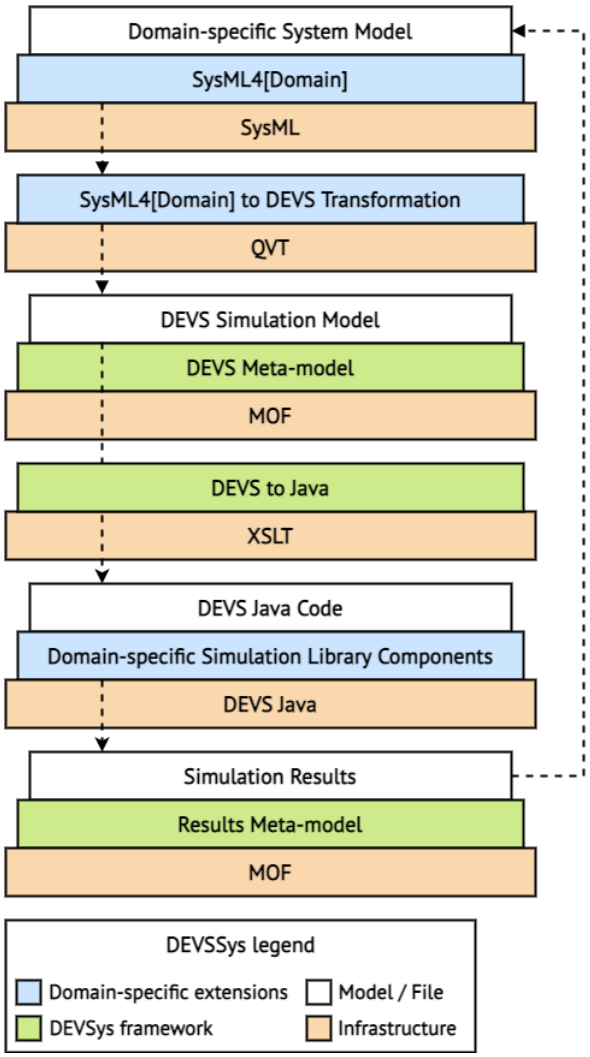


FIGURE 3.9: DEVSys: A framework for simulating SysML models with DEVS.

This code is the execution target of the simulation. After execution, simulation results are produced and stored into a results meta-model. These results are in turn incorporated back into the starting SysML system model and the process can be iterated.

3.2.2 Mathematical equations solver using parametric relationships

In contrast to highly dynamic systems which require large-scale simulations to be analyzed, there are cases where closed-form approaches for analyzing different performance aspects of the system actually exist, and suffice for fine-grained analysis [145]. Where feasible, such approaches are desirable to use in system design, e.g., in the form of mathematical equations which receive specific inputs and compute outputs, while being connected via parametric relationships with the system model. The reason is that in those cases the system designer has more transparent interaction and faces less complexity w.r.t. the behavior of the system than in the case of black-box simulations. In fact, such equations can be tailored to systems that rely heavily on personalization, such as CPHS, since they enable the designer to analyze perceived QoS and cost aspects using custom combinations of the equations, depending on the requirements of system users.

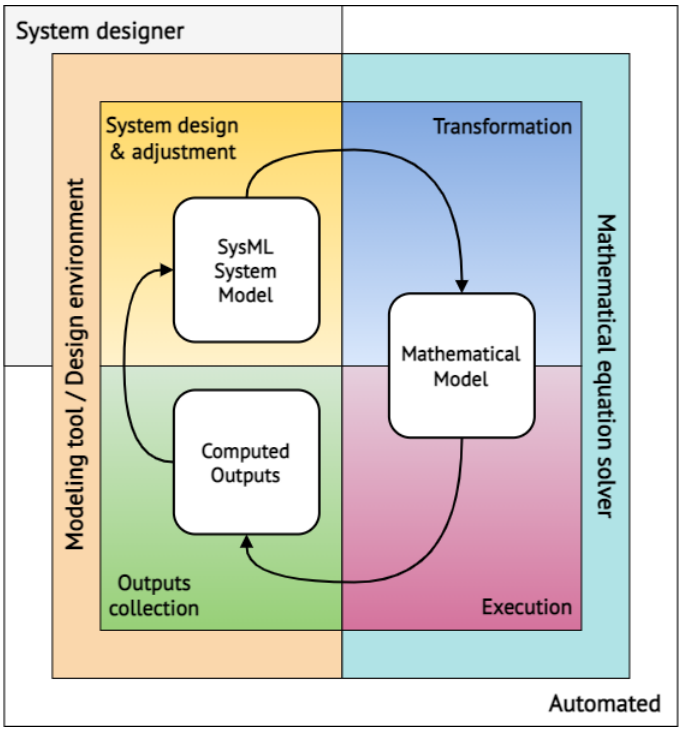


FIGURE 3.10: Iterative process of mathematical equation solving during MBSD.

Mathematical equations are contained within SysML parametric diagrams; there, they are connected to (a) input parameters that are bound to value properties of different system components -or other model entities-, and (b) output parameters where the computed result is stored. Parametric diagrams provide a way to integrate analysis models that are described in mathematical equations and constraints, with design models that describe structural and behavioral aspects of systems. The execution of these diagrams enables the computation and update of the values of the output parameters at any time during execution. Mathematical expressions can be solved using well-known math solvers, like OpenModelica [76]. This solver is based on Modelica [75], a formal, equation-based modeling language. OpenModelica creates a Modelica model stemming from the system model, and executes it. The produced results, i.e. the equation outputs, are incorporated into the system model. This process is illustrated in Figure 3.10.

3.3 Automated decision-making within SysML models

Designing SoS is a complex process that involves, among other activities, the specification of system structure and requirements, analysis of behavior and performance, and exploration and decision of the most appropriate design solution(s). The integration of these activities is advocated by MBSD, where the core SysML system model can be enriched with additional capabilities (see Section 3.1.1), such as analysis or decision-making. In a comprehensive SysML literature mapping study [292], external models that serve the analysis of a system from simulation frameworks or engines to tools for the assessment, analysis, and evaluation of performance characteristics, are presented. However, the integration of SysML models with decision-making tools is also considered crucial for efficient system design [233]. In this Section, we focus on decision-making as a critical design activity.

In practice, a system designer should be able to decide on the configuration of complex SoS that are typically described in SysML. Such systems may comprise several heterogeneous components and discrete requirements that need to be satisfied by these components. Therefore, the system model, containing all these entities, may become so complicated that the designer needs some kind of help/feedback to make related system design decisions [140].

To this end, we provide the system designer with automated decision support capabilities in order to tackle the decision-making complexity. Specifically, the designer creates the system model, which in turn is transformed into a decision model (Section 3.3.1), created within an external decision-making tool; the latter is based on the Prolog language (Section 3.3.2). This transformation is transparent to the system designer and is executed in an automated fashion, dealing with the complexity "under the hood". Via the SysML-to-decision model transformation, alternative system design solutions are explored; these solutions are directly presented to the system designer within the SysML system model using an automated feedback approach. The designer assesses them and designs what is "best". In addition, our decision model expands this ability of the designer to make decisions, based on automated recommendations/suggestions, and thus to produce an efficient and satisfactory system design that fully covers the needs of its users.

Note that this process is based on basic SysML entities; no extensions are needed. Thus, it is generalizable and can be applied to a variety of SoS, described in SysML. The only prerequisite is that the system designer creates the system model, specifying SysML system requirements in detail, using constraints.

3.3.1 Transforming SysML model to decision model

In Figure 3.11, the integration of the SysML system design model and the decision model is presented. The system model, created by the designer, is contained within the design environment. The designer specifies system components and their properties, requirements with their levels, and parametric diagrams that contain requirement constraints along with their formulas. The system model, containing the aforementioned entities, is transformed into the rule-based decision support model to generate system design configurations, i.e. provide acceptable combinations of property values of system components, conforming to SysML requirements. The decision model –here, named DSM– is contained within an external decision-making tool –here, named Decision Support System (DSS)–, completely hidden from the designer. The SysML system model-to-DSM transformation process is automated, by executing the following actions without requiring any involvement from the designer.

(i) **Validate the SysML system model.** A critical prerequisite of the model transformation process is the validation of the correctness and completeness of the SysML system model. To this end, the plugin contains corresponding validation rules that capture important conditions, which in turn must be checked against the system model. For example, every requirement constraint must contain a formula and be connected with a requirement (via a refine relation). Another rule may be that every parametric diagram containing a requirement constraint must also receive required input from system components, which in turn should hold specific properties.

(ii) **Generate the decision model.** This model is implemented using the Prolog language, which enables the creation of knowledge bases, comprising predicates and rules. The decision model is one such knowledge base, which stems from the SysML model. Requirements, constraints and associations between requirements and system components are transformed into Prolog predicates and rules (see Figure 3.11). Requirements are transformed to Prolog predicates. The design constraints are transformed to respective Prolog rules. The

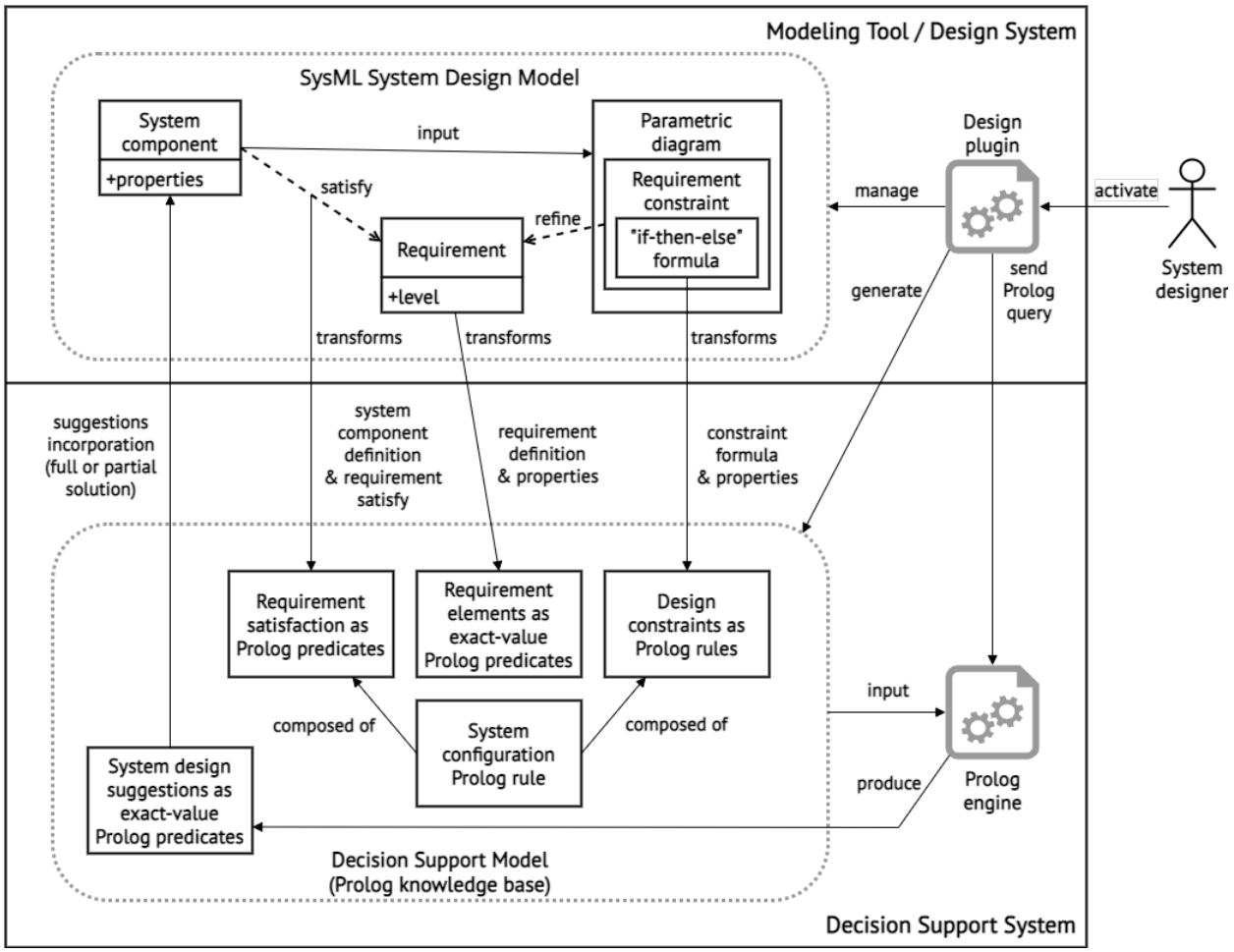


FIGURE 3.11: Transforming SysML system models to decision support models.

requirements-system components associations (i.e. satisfy) are transformed to corresponding Prolog predicates. Prolog elements are then combined into a general system configuration rule, based on which a feasible system configuration solution is computed as a response to a Prolog query, and is finally produced.

(iii) **Ask for a configuration solution.** A Prolog query, "asking" for a feasible system configuration solution, and the decision model, serve as inputs to the external tool, which executes the query. The outputs of the execution are system design suggestions expressed as exact-value Prolog predicates. If a feasible solution is found, the resulting suggestions are incorporated into the SysML system model (i.e., values are fed back into the properties of the associated components). The decision tool strives to satisfy all requirements based on the decision model; however, in some cases there may not be a solution satisfying all requirements. In any case, the plugin displays the yielded (full or partial) solution to the system designer, who can in turn choose if the solution is sufficient or not.

Note that the design plugin can be applied on the entire SysML system model or sub-parts of it (e.g., only a specific component), and does not depend on the actual modeled system. Thus, it is considered generic and can be reused across multiple domains.

3.3.2 Decision model entities as result of SysML model transformation

In Figure 3.12, the predicates and rules that are generated in Prolog as a result of the transformation of the SysML system model entities, are presented. The latter entities are shown on the left side of the Figure; parametric diagrams and their content (i.e. requirement constraints and formulas) are blue-colored entities, requirements are red-colored entities and system components are depicted as grey-colored. Similar-colored arrows indicate which entity provides input to associated Prolog predicates or rules; this input may be the definition of the entity (e.g., its name) or the value(s) of the properties of the entity. A detailed description follows.

(i) **Exact-value Prolog predicates.** These predicates contain the required values of the properties of the system components; these values are required to achieve a specific satisfactory level per requirement. The inputs of the predicates are names of requirements, values of properties of system components, provided by the parametric diagrams, as well as values of levels of the requirements.

(ii) **Value range Prolog rules.** They encode the ranges of the values that can be assigned to the different properties of system components, so that the specific required level is achieved. The value range rules are complementary w.r.t. the exact-value Prolog predicates and act as filters while searching for feasible solutions. The inputs of the rules are the properties (including their respective values) of system components extracted via the parametric diagrams, the constraint formulas that define the ranges of the values, and the names and level values of the requirements.

(iii) **Component-requirements Prolog rules.** These rules connect system components with corresponding requirements that they need to satisfy. In addition, the rules serve to limit the search range of feasible solutions by taking into account only connected requirements and system components. The inputs of the rules are the names and properties of system components and their requirements.

(iv) **Requirement satisfaction Prolog predicates.** They assign the required levels of requirements to system components that need to satisfy them. The inputs of these predicates are

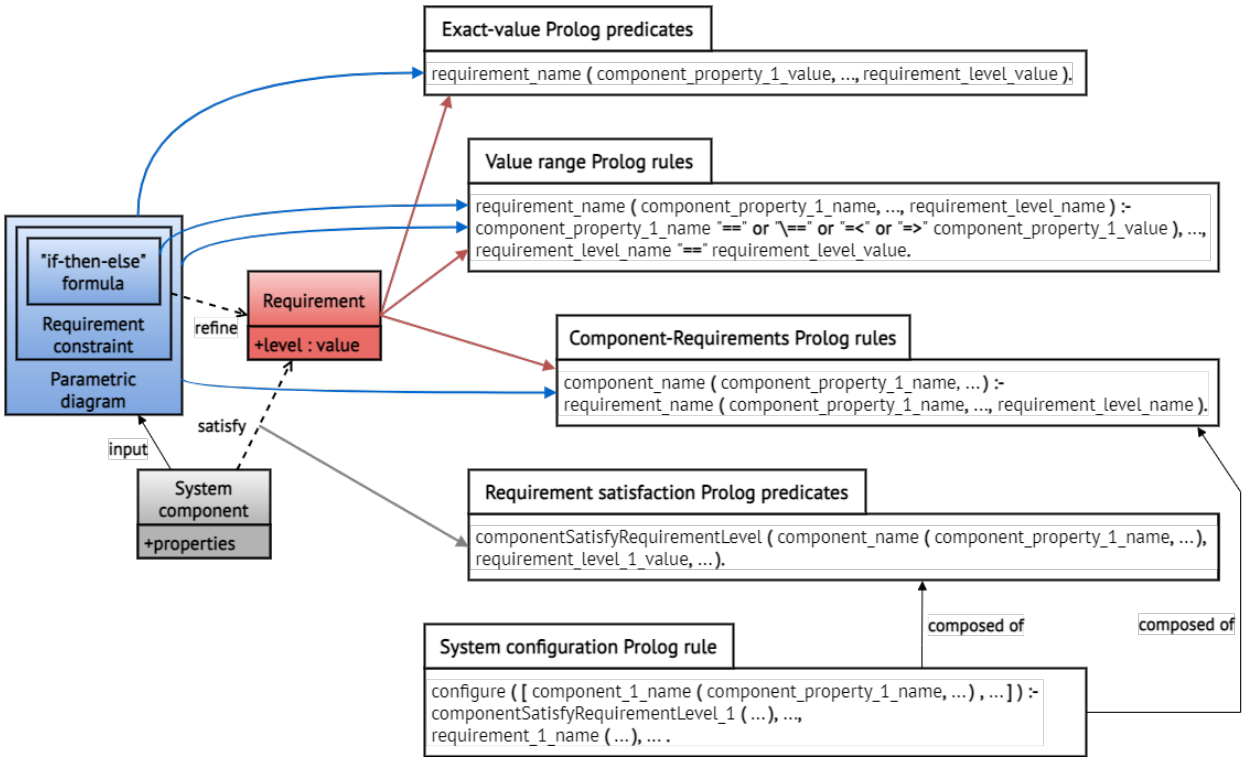


FIGURE 3.12: Construction process of Prolog predicates and rules.

the -satisfy- connections between system components and requirements. Specifically, the predicates hold the names and properties of system components and the values of their required levels.

(v) **System configuration Prolog rule.** This is a unifying rule that produces the final solution (full or partial, if feasible) to the basic Prolog query. It combines rules (iii) with predicates (iv). The returned solution is composed of values which, when assigned to corresponding properties of system components, satisfy the required levels while being compliant with predicates (i) and rules (ii).

3.4 Extending SysML to integrate QoS analysis

In this section, we describe our comprehensive QoS meta-model used to serve QoS, which is our primary focus. This meta-model definition is a structure of QoS-related concepts, capturing generic QoS requirement definitions and analysis concepts as well as relationships between them. This leads to a portable QoS requirements meta-model that can be re-used in various SoS domains. We also analyze validation rules and constraints that are imposed on entities of the QoS meta-model, mainly the quantitative requirement entities, to ensure correctness and completeness.

3.4.1 Domain-independent QoS entities

In Figure 3.13, the QoS meta-model is depicted; it contains entities that are described in Section 3.1.2, as well as relationships between them. While in that section we described the theoretical constructs used in the design framework, here we describe the actual meta-model implementation. We further delve into details of entities that capture QoS and participate in QoS analysis.

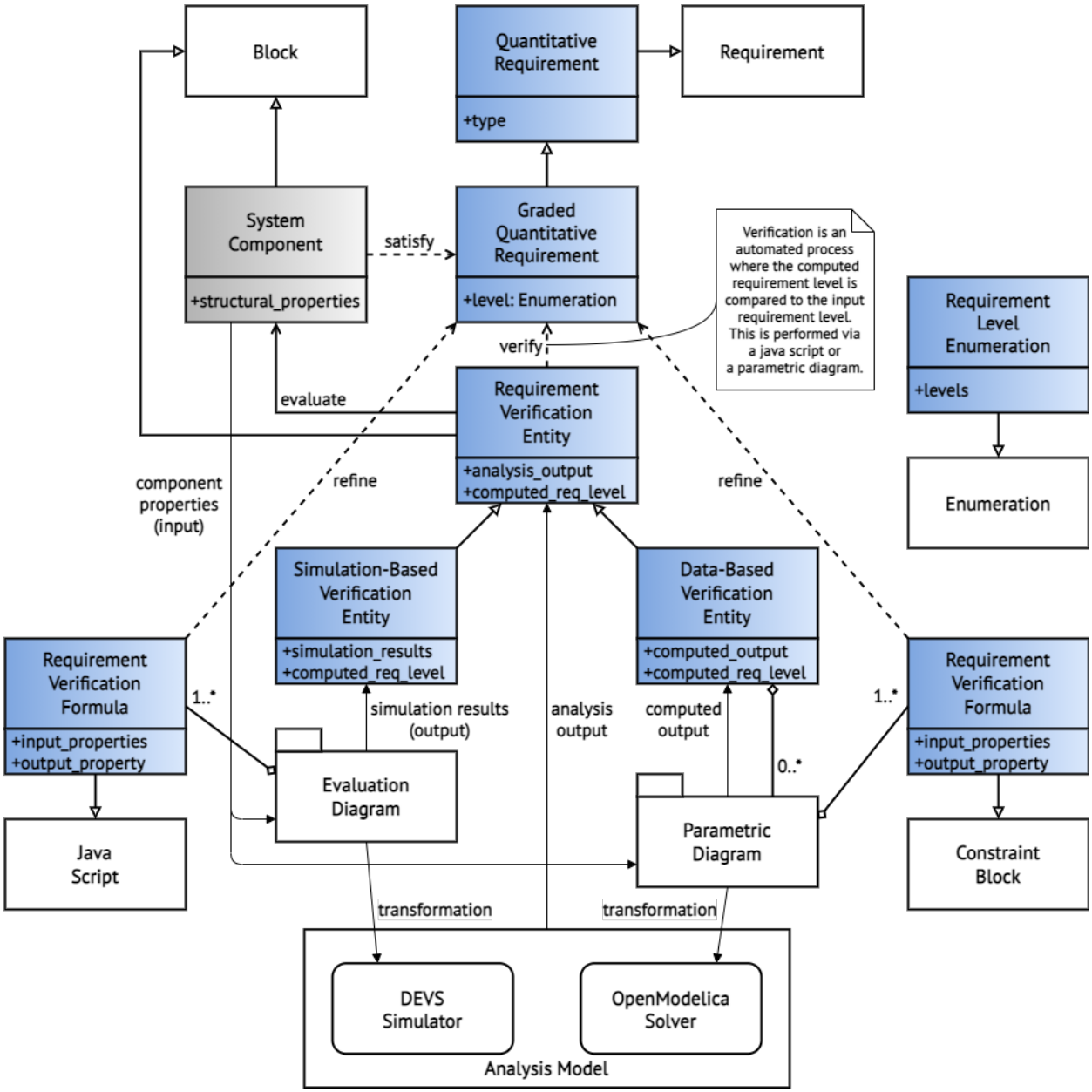


FIGURE 3.13: QoS meta-model for incorporating QoS in system models.

We begin with the *Graded Quantitative Requirement* entity; it is defined as a specialization of a *Quantitative Requirement* entity which in turn is a stereotype of the generic SysML requirement. Note that the quantitative requirement is a generic entity that can be used to facilitate a system designer to specify various requirements of different types. In case additional grading information is available, the graded specialization can be used for more fine-grained design. Both the graded and the quantitative requirement entities inherit basic properties of the SysML requirement (i.e., unique identifier (*id*) and textual description (*text*); however, we specify custom properties in order to cover the targeted domain (i.e. QoS). The quantitative requirement holds a *type* property that indicates various types of requirements that can be defined within a system model. The graded entity holds a custom textual *levels* property; this indicates the level that should be achieved to satisfy the requirement. The latter property is of an enumeration type; the property's value can be selected from a defined enumeration that holds multiple values for the respective requirement levels.

As described in Section 3.1.2, a requirement is satisfied by a *System Component* that holds respective structural (system-related) properties, and is verified by a *Verification Entity*. The latter entity holds output properties that are produced by an external *Analysis Model*; a computed level for the requirement is among those properties. In essence, this level is compared to the required level (specified within a graded requirement during its definition), leading to the actual verification of the requirement (i.e. whether the levels are the same or not).

Specializations of the verification entity are simulation-based and data-based. Each of these specializations is required to address different analysis methods in different domains. A *Simulation-Based Verification Entity* receives simulation results –and computed requirement levels– from an *Evaluation Diagram*; such a diagram was first explored in the work of Tsadimas et al. [275]. The evaluation diagram is transformed to a *Simulator* like DEVS, where it is executed and corresponding results are produced. Note that the evaluation diagram comprises *Verification Formulas* that act as *Java-based scripts* and participate in the execution of the simulation. A *Data-Based Verification Entity* contains a SysML *Parametric Diagram*, which in turn is transformed into an *OpenModelica Solver*. Since our meta-model conforms to the SysML standard, the parametric diagram also contains a verification formula, which is a specialization of the SysML constraint block. Such formulas refine requirements, enriching them with additional functions or expressions that are executed by the mathematical solver. Both specializations, used in their respective domain(s), provide the computed output that is compared to the required levels property of the requirements.

3.4.2 QoS-related validation rules

The QoS meta-model, presented in Section 3.4.1, forms an abstract syntax of a QoS profile that enables the system designer to specify QoS requirements within a SysML system model. However, during the development of this model, several defined requirements may need to be related to several different model entities. This heavily increases the design complexity, introducing potential errors, like overlooking a crucial connection between a component and a requirement. For this purpose, the system model should be validated against restrictions that must be satisfied by the system design. The QoS meta-model imposes such restrictions as sets of SysML constraints; such sets include specific validation rules that constrain defined requirements and/or their relationships, and characterize their completeness, correctness, and validity of use. SysML constraints can be enforced to the system model at the time it is constructed; although when needed, the designer can manually run a model-checking routine that provides information about any constraint violation. During the validation process, requirements are checked against the validation rules and either violate them and are thus marked as invalid/incorrect, or are successfully validated, indicating a correct design. The implementation of these constraints can be binary, i.e. implemented

TABLE 3.1: QoS meta-model validation rules.

Stereotype	Base class	Properties	Constraints
Graded Quantitative Requirement	SysML Requirement	-id -text -levels	Completeness: -participates in SysML satisfy relationship with exactly one System Component. -participates in SysML verify relationship with exactly one Requirement Verification Entity.
Requirement Verification Formula	SysML Constraint Block	-input properties (depend on defined equation/expression) -output	Completeness: -participates in SysML refine relationship with exactly one Graded Quantitative Requirement. -contained within one or many SysML Parametric Diagram(s). Correctness: -have defined input properties that bound value properties of other model entities (e.g., system components), based on defined equation/expression.
Requirement Verification Entity	SysML Block	-computed output (bound to output property of Formula) *or* -simulation results	Completeness: -have analysis output from simulation or computation of exactly one Formula. -participates in custom evaluation relationship with exactly one System Component.
System Component	SysML Block	-structural properties	Completeness: -participates in SysML satisfy relationship with one or many Graded Quantitative Requirement(s). -participates in custom evaluation relationship with exactly one Requirement Verification Entity.

in a Java environment or OCL-based [258]. In addition, they can be global, i.e. executed for the whole SysML system model, or local, i.e. employed for sub-parts of it, even a single model entity. In table 3.1, the entities of the QoS meta-model and the validation rules that are imposed on them, are presented.

3.5 Extending SysML to integrate cost analysis

According to Wymore [293], there are several prominent parameters that should be taken into account when designing a system: besides provided services (system output), available technology and performance, cost is also a crucial factor that constrains design decisions. It should be noted that Wymore adopts the term performance to describe all non-functional requirements the system should conform to, also including availability, security, quality of service, etc. In fact, during system design, capital (initial investment) or operating costs drastically affect (and are affected by), as well as constrain, the services a system provides to its end-users [260] and should be leveraged against performance metrics, taking into account available technology. This necessity has grown even more during the recent recession, where many systems were redesigned in order to accommodate cost reductions during their operation.

However, popular engineering methodologies [168] are not focusing on exploiting cost parameters while designing a system, since they are typically superseded by performance [129]. In the cases where cost is actually taken into account, not all its different dimensions (e.g., cost categories like operational expenses, equipment costs, etc.) are explored; only the design process costs or the costs incurred during the system's architectural changes (capital costs) are studied [47]. Common pitfalls of related efforts are the lack of generality, scalability, dynamicity and simplicity, while cost exploitation is customised for a specific category of systems or case studies [82, 271, 248, 158].

A framework to overcome these difficulties should be generic enough to successfully address different system categories and enable cost analysis at both a high-level (e.g., on the level of overall capital or operational expenses of a system) or fine-grained (e.g., the individual cost of a device). In practice, the system designer should be facilitated to model, design and evaluate a system

from a cost perspective as well, and be able to perform both performance and cost analysis of the explored design alternatives in a single modeling environment.

There are numerous efforts to enhance the exploration of system performance, as a design parameter, using SysML [292, 274]; however there is a lack of cost exploitation is a similar manner. To this end, we aim to integrate cost properties and restrictions within SysML models at a generic level, and accommodate the designer to explore design alternatives under specific cost restrictions, e.g. evaluate either acquisition or operating costs for alternative design decisions. This way we integrate cost analysis in a popular modeling language employed for MBSD, so that it may be explored for any system designed using SysML. The employment of the approach can lead to an efficient system design and enhance the decision-making support of system designers, facilitating them to identify areas for improvement; ideally, a designer should minimize or reduce the costs of the provided system services, while maintaining their high quality for greater satisfaction and usage by the end-users.

To achieve our goal, we extended available SysML modeling entities in a standardized fashion by constructing a generic SysML profile that encodes cost design parameters. This is a custom cost-related SysML meta-model extension, termed *cost profile*. To enable computations related to this perspective (e.g., to compute the acquisition and operational expenses of a system), we introduce a cost functions library, which is readily available to the designer during design time, along with the profile. The profile is general and extendable enough to be applied in any system, leveraging the expressiveness and flexibility of SysML. To automate computations, we employ mathematical-based parametric relationships [211]. The end results (i.e. the computed cost entity values) are in turn used to populate the parameters of Cost profile elements; related requirements (e.g., conformance to budgetary constraints) are then verified by comparing the computed and desired values.

In essence, we aim to facilitate generic cost evaluation, enabling different solutions to be explored until the designer can reach a satisfactory system design that remains affordable to its users (in the context of the desired domain). Such an effort entails description and computation of economic parameters such as the TCO [278], CapEx or OpEx [188], etc. Such parameters are important in/during system design [192] and should thus be explored and estimated, in order to enable a high-level (e.g., on the level of the TCO) or a fine-grained (e.g., the acquisition cost of a single system component) cost analysis of the examined systems.

To understand the challenges of such an analysis, consider the case of a designer who models a system from a cost perspective, employing the widely used SysML modeling language [199]. Without a generic cost-oriented design approach, the designer would have to resort to custom extensions of a domain-specific structural profile [72] of a system by adding cost properties (along with the structural ones) on each and every system component, that should be analyzed cost-wise. This heavily increases complexity, introducing more potential errors (e.g., overlooking a critical cost property), and is neither scalable nor generalizable. Moreover, it would reduce clarity and transparency regarding cost integration, analysis, and estimation, especially when used across multiple system domains.

Drawing an example from the healthcare domain, assume that a system designer is tasked to model a medical monitoring system for every patient room in a 500-room hospital. After selecting the needed components (for example, medical sensor devices and sensor-data aggregators), the designer will need to compute the overall CapEx for the hospital. Without integrating this process into the system model and automating it, the designer will have to do this manually for each system component. Moreover, should an upgrade on the equipment take place, the process needs to be performed anew; this is rigid and constrains dynamicity (for example, changing component prices depending on operational properties). It is also important to note that in a heterogeneous system (e.g., a hospital with different medical equipment per room), such a cost analysis becomes

untenable without proper automation in place. The same observation applies to other systems/-domains; for example, consider a complex train transportation system that is composed of tens or hundreds of stations, traversed by thousands of trains and boarded by millions of passengers per day. Ideally, system designers should be able to apply their domain expertise without worrying about other cost analysis complexities.

To deal with the aforementioned challenges, we employ and extend SysML, integrating cost-related parameters into SysML system models, facilitating their analysis. The proposed extensions are the following.

(i) Domain-independent cost entities (Section 3.5.1). For example, an OpEx cost entity that represents the overall expenses incurred during system operation is domain-agnostic. This extension deals with the challenge of generality, as well as clarity and transparency.

(ii) Automated (re-)computations of costs (Section 3.5.2) incurred by the cost entities. For example, the summation of sub-cost entities, like energy and labor, to produce the value of the overall OpEx cost, can be performed at design/operation time. This extension deals with the challenge of scalability and dynamicity.

(iii) Cost computation functions that are readily available in an inventory (Section 3.5.3). This extension makes our approach more designer-friendly and augments the automation features of point (ii).

We encapsulate these extensions in a comprehensive SysML cost meta-model that is described in the following section.

3.5.1 Extended domain-independent cost entities

Cost-related entities comprise our SysML cost meta-model that is depicted in Figure 3.14. Specifically, the illustrated white elements are standard SysML-provided constructs, blue-colored elements illustrate the proposed cost extensions, and grey-colored elements are specializations of these extensions.

The depicted *Cost* element is the centre of our profile, having the following characteristics.

- It contains cost-specific properties, i.e. a *value*, used for the assessment of the system components' worth, a *measurement unit* that represents the currency (e.g., "Euro"), and an *instances* property that represents the number of occurrences of each system component connected to a cost entity; note that a system may contain multiple instances of the same component. The latter property differentiates from the others since it is derived; the / symbol, preceding its name, indicates that the value of this property is produced from another element (here, from the structural components of the system).

- It estimates various structural system components, since it holds the cost values that characterize them. Conceptually, all these components are represented by the SysML block as depicted in Figure 3.14; in order to illustrate their connection, there is a custom *estimation* relationship between the cost and the block elements. Note that a system can have a hierarchical structure; a "parent"-"children" hierarchy can be formed by the system components. The same hierarchy applies to the cost entities connected to the structural components.

- It represents a general cost entity that has specific cost specializations; these specializations form hierarchies (as mentioned in point (ii)) and inherit all the characteristics of the Cost element, as described in Section 3.5.2.

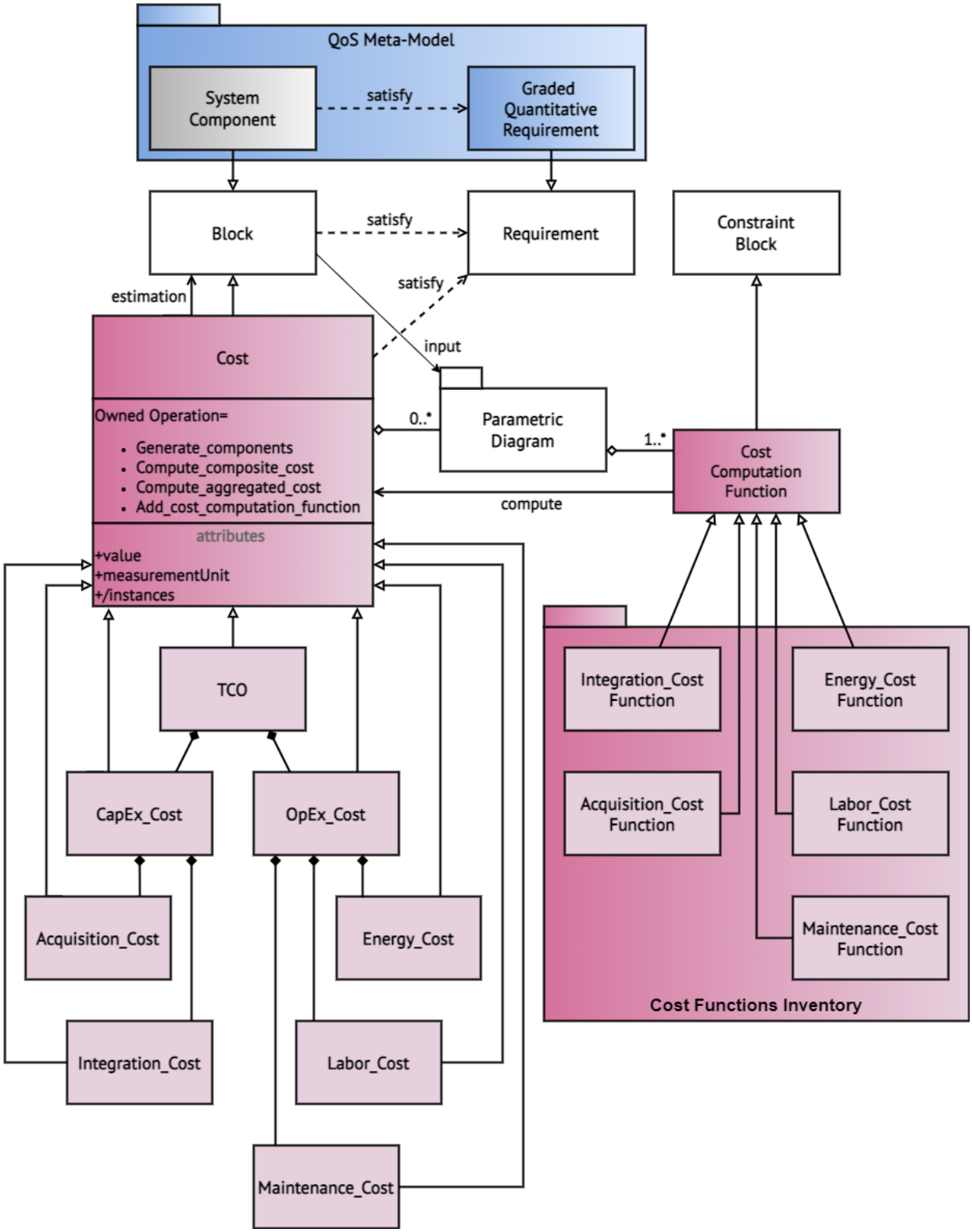


FIGURE 3.14: SysML cost meta-model.

- It owns specific operations, referring to cost computations. The value of a cost entity can be computed either by automatically summing up the cost values of its children entities as described in Section 3.5.3 or by custom computation functions. Regarding the latter, described in detail in Section 3.5.4, they are represented by the *Cost Computation Function* entity (as shown in Figure 3.14) which is the stereotype of the SysML constraint block. Each function holds a specific mathematical expression that generates a cost value. These expressions are computed in a standardized fashion, i.e. via the graphical SysML *Parametric Diagrams*; the system designer inserts the expressions within the diagrams along with appropriate input/output properties, executes them (parametrically) and the computed cost value is extracted for assessment.
- It satisfies –cost-related– requirements, that specify certain conditions that the system must conform to. This is needed to complete the cost analysis of a system. Specifically, via a SysML *satisfy* relationship (depicted in Figure 3.14), the cost entity can meet and fulfill requirements. When the system is actually evaluated (cost-wise) such requirements are either satisfied or not.

3.5.2 Cost-related specializations

The first specialization of the cost element (see Figure 3.14) is the *TCO* cost entity that represents the financial assessment and cost accounting tool [278, 173], used to determine the total economic value of an investment, including capital and operating expenses. Typically, organizations and systems have two types of expenses, i.e. CapEx and OpEx [188]. Both cost categories refer to money paid by the organization, although each one is managed differently for accounting or taxation purposes [256]. CapEx is the funds provided by the stakeholders (or end-users) of the system to purchase or improve physical assets, such as system components. OpEx is an ongoing cost, required to keep a system operational on a daily basis. Usually, organizations try to reduce the expenses that are incurred during system operation, without affecting its quality or performance. Typical OpEx examples are energy consumption costs and personnel salaries. We represent the CapEx and OpEx categories via the *CapEx_Cost* and *OpEx_Cost* specialization entities, accordingly.

Note that both entities are considered composite, i.e. they comprise different categories of sub-cost entities. The CapEx_Cost contains the *Acquisition_Cost* entity that represents the acquisition cost of a newly purchased system component, and the *Integration_Cost* entity, related to the integration expenses of the acquired components into the system. OpEx_Cost is the sum of the *Energy_Cost*, associated with the energy consumption costs of the components, the *Maintenance_Cost*, related to the expenses incurred to maintain the components operational and in good condition, and the *Labor_Cost* entity, i.e. the salary of system participants that operate the components and provide services. A system can be analyzed cost-wise using either or both CapEx_Cost and OpEx_Cost entities, depending on the system design focus.

3.5.3 Cost-related computations

It is worth mentioning that we categorize the cost entities into composite, aggregated, and individual, based on the way their values are computed. These entities and their computation process are included in horizontal and vertical tree-like hierarchies as depicted in Figure 3.15. Indicatively, we employ CapEx as a composite cost entity, while its sub-cost categories, i.e. Acquisition and Integration, represent aggregated and individual costs.

Traversing the hierarchy from the left to the right side, we have design entities connected to composite cost elements that are linked to aggregated (or individual) cost entities trees. This signifies that each system design entity is estimated by respective composite cost elements (such as

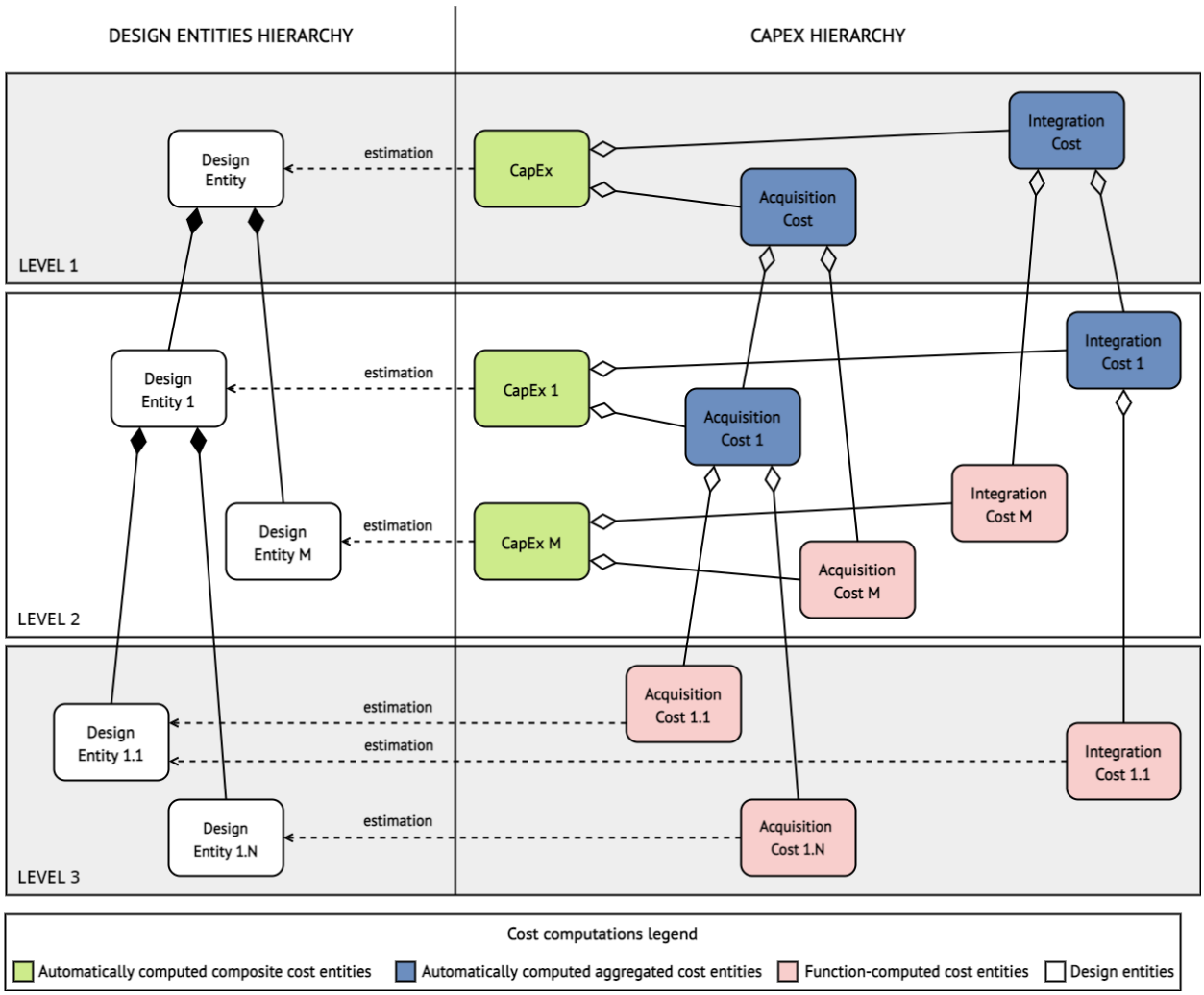


FIGURE 3.15: Automated computation of CapEx.

TCO, CapEx or OpEx), which in turn are composed of aggregated (or individual) cost elements of different types (e.g., CapEx comprises acquisition and integration cost entities). Note that the estimation (or not) of certain design entities by corresponding composite cost elements can be designated by the designer; that is, certain design entities can be directly estimated by corresponding aggregated or individual cost elements only.

Traversing the hierarchy top-down, we essentially map the structure of the design entities to the corresponding structure of the aggregated and individual cost entities. The computations of the values of the different cost entities comprising this tree, start from the leaf nodes and traverse the tree upwards, as described in the following.

The leaf nodes are individual cost entities of system components, which may correspond to different cost categories (e.g., the acquisition cost of a system component or its energy consumption cost). The values of these cost entities are computed by cost functions included in an inventory, as described in Section 3.5.4. Each individual cost entity is computed by a specific function inserted by the designer.

By summing up the individual –function-computed– costs of a level, the aggregated costs of its parent level (corresponding to more complex design entities) can be computed automatically per type. Specifically, an aggregated cost entity obtains its value by the automatic summation of the values of its children cost entities (corresponding to the respective design entity hierarchy), conforming to the same cost category. For example, the acquisition cost of the part of the system *W*, comprising the components *X* and *Y*, is the summation of the individual acquisition costs of X and Y. This can be done automatically for each level of the hierarchy; note that it is permitted to have a mixture of aggregated and individual costs on a level.

At the composite cost entities of the tree (including the top-level root), the same-level aggregated and individual costs are finally summed up, across the different types, to produce the overall cost value of the system (root composite cost entity) or its subsystems (intermediate composite cost entities). Specifically, the value of a composite cost entity is computed by automatically summing up the values of all its same-level children entities.

Figure 3.15 depicts the design entity hierarchy (left side) as well as the corresponding cost entity tree (right side). The aggregated cost entities whose values are automatically computed via hierarchical summation are blue-colored, the composite cost entities are green-colored, and individual cost entities, whose values are computed by designer-applied functions, are pink-colored. The white elements represent the design entities of a system.

In Figure 3.15 we present the hierarchy of cost entities; this hierarchy pertains also to the order of computations. In order to show how aggregated and composite cost entities are computed, we employ the following functions (in pseudocode format).

```
Data: design Entity (dE), cost Entity (cE), cost Type (cT), cost Value (cV)
function computeAggregatedCost(dE, cT):
    foreach cE : {dE, cT} do
        if dE has children then
            foreach cE_i : {cE, cT} do
                dE_i : {cE_i};
                cE.cV += computeAggregatedCost(dE_i, cT);
            end
            return cE.cV;
        else
            return cE.cV;
        end
    end
end Function
```

Algorithm 1: Computing the values of aggregated cost entities.

Function 1 is used to compute the value (cV) of an aggregated cost entity (cE) -of a certain type (cT)- corresponding to a specific design entity (dE). It thus receives as input the design entity and cost type, and generates as output the corresponding value of the aggregated cost entity. It is worth mentioning that we exploit the custom relationship (i.e. estimation) that is uniquely used to connect design entities to cost entities.

Specifically, based on the estimation relationship, a cost entity that has the input cost type and is connected to the input design entity is first found and set as our current cost entity. In case the associated design entity has no children design entities (cDE_i), i.e. it is not composed of others, the function terminates and the cost value of the current cost entity is returned. In case the design entity has children, we apply the function recursively on each of the cost entities (cCE_i) which estimate them, as long as they are of the same –input– type; the cost value is updated via summing up the return values of the recursive functions calls.

```
Data: design Entity (dE), cost Entity (cE), estimation Relationship (eR), cost Value (cV)
Function computeCompositeCost(dE):
    cE : {dE}
    foreach eR_i : {dE} do
        if cE_i : {eR_i} != cE then
            cE.cV += cE_i.cV;
        end
    end
    return cE.cV;
End Function
```

Algorithm 2: Computing the values of composite cost entities.

Function 2 is used to calculate the value (cV) of a composite cost entity (cE) corresponding to a specific design entity (dE), which is the only input of the function. The generated output is the cost value of the corresponding cost entity. We employ the estimation relationship (eR_i) that the design entity has with cost entities, in order to locate our current cost entity (cE_i). By traversing all the rest of the estimation relationships (i.e. between the design entity and the corresponding aggregated or individual cost entities), we extract their cost value and sum them up to compute the value of the composite cost entity.

3.5.4 Cost computation functions inventory

The inventory of the cost computation functions is essentially a collection of various mathematical or logical expressions that transform input (e.g., structural property values of a system component) to output (i.e. the cost value of this component). Extracting the values of individual cost entities is equivalent to computing these expressions.

Note that multiple functions may be readily available within the inventory, loaded to our custom Cost meta-model (see Section 3.5.1). Forming such an inventory brings the following benefits to the system designer. First, functions are readily available in a drag-&-drop form for connection with appropriate cost entities. This enables function re-use without having to re-implement them from scratch. Even if system designers are not cost-analysis experts, they can utilize functions to perform a useful cost analysis. Second, the inventory can be easily extended with more functions at will (by designers themselves or other experts), e.g., to perform more fine-grained cost analysis. Third, it can also be used across several system domains, containing computation functions for various cost categories. Last but not least, during system design, the inventory remains a part of the modeling environment, meaning that all computations are defined and conducted within a single tool, without requiring time-consuming interactions with external cost-computation tools.

3.5.5 Cost-related validation rules

The cost meta-model, presented in Section 3.5.1, forms an abstract syntax of a cost profile that enables the system designer to specify cost entities within a SysML system model. However, during the development of this model, various and several cost entities may need to be related to several different model entities and connected to specific requirements. This heavily increases the design complexity, introducing potential errors, like overlooking a crucial connection between a cost entity and a requirement.

For this purpose, the system model should be validated against restrictions that must be satisfied by the system design. Similar to the QoS meta-model, the cost meta-model imposes such restrictions as sets of SysML constraints; such sets include specific validation rules that constrain defined requirements and/or their relationships, and characterize their completeness, correctness, and validity of use. SysML constraints can be enforced to the system model at the time it is constructed; although when needed, the designer can manually run a model-checking routine that provides information about any constraint violation. During the validation process, requirements are checked against the validation rules and either violate them and are thus marked as invalid/incorrect, or are successfully validated, indicating a correct design. The implementation of these constraints can be binary, i.e. implemented in a Java environment or OCL-based [258]. In addition, they can be global, i.e. executed for the whole SysML system model, or local, i.e. employed for sub-parts of it, even a single model entity. In table 3.2, the entities of the cost meta-model and the validation rules that are imposed on them, are presented.

3.6 Implementation environment

During the system workflow design, a system designer typically needs to impose constraints and apply validation rules on the system model. These are required for error-checking and model verification in terms of correctness and completeness. Custom SysML profiles, while expressible in the context of a system domain and its design, do not natively support such actions. We implement them using additional software modules, which take the form of model plugins written in a general-purpose programming language (like Java). These plugins effectively accompany our custom SysML profiles, extending their constructs with additional functionality; this may involve

TABLE 3.2: Cost meta-model validation rules.

Stereotype	Base class	Properties	Constraints
Cost	SysML Block	-value -measurement unit -/instances	Completeness: -participates in custom estimation relationship with System Component(s). -participates in SysML satisfy relationship with SysML Requirement(s). -participates in custom evaluation relationship with Requirement Verification Entity. -participates in containment relationship with sub-cost entities (if it is a composite Cost entity). Correctness: -contains a Parametric Diagram. -value property is populated after execution of Parametric Diagram. -its ``value'' property is derived by (automatic) summation of values of sub-cost entities (if it is a composite Cost entity).
Cost Computation Function	SysML Constraint Block	-input properties (depend on defined equation/expression)	Completeness: -is contained within one or many~SysML Parametric Diagram(s). Correctness: -has defined input properties that bound value properties of other model entities (e.g., system components), based on defined equation/expression.
Parametric Diagram	SysML Block	-computed output (bound to output property of Formula) *or* -simulation results	Completeness: -is contained in corresponding Cost entity. Correctness: -contains at least one Cost Computation Function. -has ``value'' property from Cost entity and input properties from estimated System Component(s).

the automatic generation and validation of graphical objects within a system model. Plugins can be integrated directly into a design tool and are transparent to the designer, who simply activates them. Since plugins accompany the defined profiles, they can be divided into general, i.e. applicable to various SoS (or their sub-parts), or domain-specific.

In our work, we have implemented plugins to provide the following functionality.

- **Cost plugin.** It is applicable to multiple SoS, accompanying our generic Cost profile and providing additional functionality to it. This plugin performs two actions, each one activated by the designer within the design tool.

  (i) It automatically checks whether defined system components are properly connected to appropriate composite, aggregated or individual cost entities (see Section 3.5). In the case where a system component has a relation with a composite cost entity (like CapEx), the plugin also facilitates the creation of all its different-category sub-cost entities (i.e., acquisition and integration) to the same component, in an automated fashion; all cost entities are initialized with default values. The same applies to an aggregated cost entity (like acquisition) that is connected to a component, although same-category cost entities are created and connected only to other components that comprise the primary component.

  (ii) The plugin automatically creates a SysML parametric diagram for each created cost entity; the designer then manually inserts a cost computation function and specific input/output properties. When the function is computed, the resulting cost value is automatically stored within the associated cost entity.

- **QoS plugin.** It accompanies our generic QoS meta-model to serve additional QoS-related functionality. This plugin performs the following actions.

  (i) Configuration of a SoS. This action allows system designers to choose a specific system configuration. Such configurations facilitate the decisions of the designers by giving them the freedom of option, abstracting away the complexity of property values generation. When a designer chooses a certain configuration among others, specific properties of system components are populated with pre-defined values. Our plugin supports the exhibition of specific system configurations in the form of a menu. It is worth mentioning that the decision support configuration (Section 3.3) is encoded into the configurations menu and can help the designer choose the property values that correspond to the "best" system design.

  (ii) Analysis of system QoS. This action accommodates the verification (or not) of QoS requirements. In essence, the output of the analysis models (either simulation or mathematical equations) is the computed level of QoS requirements. When our plugin is activated, these levels are automatically compared against the desired ones (i.e. those defined with the requirements). When an obtained requirement level does not verify the desired one, the frame of the "defective" requirement as well as its connection with the respective system component(s) becomes red-colored, notifying the designer that there is a problem. In addition, the design tool exhibits appropriate information to system designers, as well as adjustments they can make. The latter may include suggested actions or solutions (like reconsidering the structure of the system). This process can occur several times –dynamically– in the system's design process, adjusting to changing operational circumstances. Thus, by assessing the design tool's automated feedback, the designer can further improve system design.

Domain-specific plugins are presented in the following sections, where our framework is applied to RTS and CPHS; these plugins accompany respective domain-specific profiles and provide additional functionality.

W.r.t the practical implementation of the proposed integrated design framework, we employed Dassault Systémes's NoMagic MagicDraw [178] and Catia CSM [32] modeling suites, since they are two of the official tools, endorsed by OMG, for UML/SysML model development. Plugins were developed using the Java programming language within Netbeans [21], a Java-based Integrated Development Environment (IDE), to further facilitate model configuration and execution.

Regarding the development of analysis models and their integration into the modeling tools, the DEVS-Suite from Arizona Center for Integrative Modeling and Simulation (ACIMS) [236] and the Cameo Simulation Toolkit (CST) [196, 207] were used; the DEVS-Suite is suitable for executing discrete-event simulations, while the CST uses OpenModelica [205] as the basis for computing mathematical equations and closed-form expressions. To implement the external decision-support system, we employed the Prolog language [24] to define predicates and rules, while we combined these rules with an Eclipse Java-based IDE to develop the respective SysML-to-Prolog plugin.

The tools and software, used to develop our framework are depicted in Figure 3.16. As shown in this figure, the system designer interacts only with the modeling environment and the SysML models, designing the system and activating the plugins. The designer does not need to deal with the execution of the analysis models, the decision-support model, as well as the incorporation of their results back into the system model; these are hidden from her and are performed automatically in the background, shielding her from complex actions and procedures.

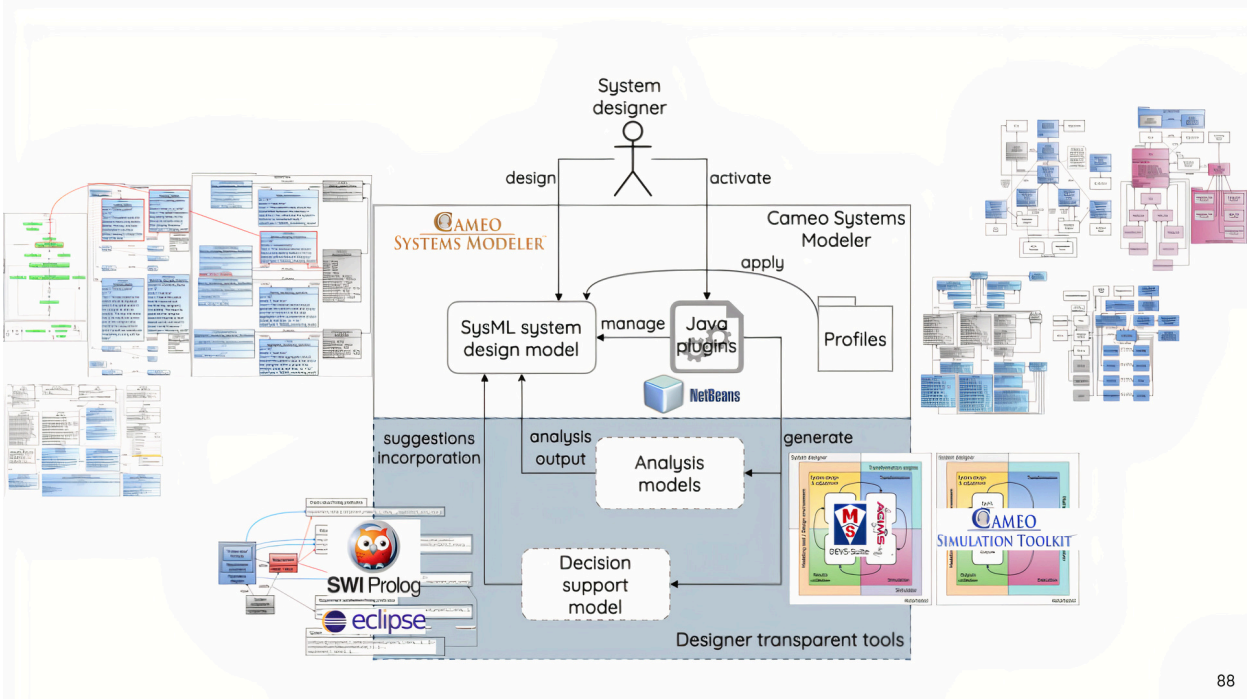


FIGURE 3.16: Implementation of integrated QoS-aware MBSD SysML framework.

Chapter 4

Applying proposed framework in Railway Transportation Systems

[...] All of the sights of the hill and the plain
Fly as thick as driving rain;
And ever again, in the wink of an eye,
Painted stations whistle by. [...]

Robert Lewis Stevenson

In this chapter, we present the application of our proposed MBSD framework (described in Section 3) to explore the QoS-related design of RTS under restrictions of operating costs.

In Section 4.1, we first describe the problem in RTS design. To address this problem, we focus on the spatial comfort of passengers in moving trains or while waiting at stations as a key metric of QoS in RTS (see Section 4.1.1); note that we classify the quality of passenger comfort using the term of LoS. We also discuss the need to perform a cost analysis in a railway system, balancing passenger comfort LoS and operational costs (see Section 4.1.2).

In Section 4.2, we apply MBSD to RTS, providing the basic concepts needed to create a RTS-specific SysML profile (see Section 4.2.1). We also describe how an effective exploration of LoS metrics using SysML can be enabled in RTS design (see Section 4.2.2).

In Section 4.3, we apply our QoS meta-model (presented in Section 3.4.1) to RTS. Based on this meta-model, we build the RTS SysML profile to explore passenger comfort LoS (see Section 4.3.1), while further analyzing the process of defining, estimating, and evaluating such LoS (see Sections 4.3.3–4.3.5).

In Section 4.4, we also apply our Cost meta-model (introduced in Section 3.5.1) to RTS to enable the design and evaluation of such systems from an operational cost perspective.

As a case study, in Section 4.5 we present the improvement of passenger comfort LoS under operating cost constraints in the RTS of Athens Metro. The study is based on real-data provided by the metro operators and provides useful results and insights w.r.t. its operation during the morning rush hours (see Section 4.5.3 where the comfort LoS is evaluated and improved, and Section 4.5.4 where a trade-off between comfort LoS and operating costs is evaluated). Finally, in Section 4.6 we discuss observations that emerged from the analysis of the design problem of the Athens Metro case study. We also comment on how the proposed framework contributes to address the specific problem.

4.1 Problem statement: QoS-related RTS design under restrictions of operational costs

An RTS involves several physical parameters associated to moving dynamic entities, like passengers and trains, as well as static infrastructure, like lines (fixed-tracks) and stations. Typically, the primary objective of urban RTS is to provide commuting/transportation services to massive passenger populations; passengers may be crammed inside train wagons, change lines, and enter/exit the system dynamically.

An interesting key QoS-related metric in that context is the spatial comfort of passengers (in terms of surrounding free space) either while on a train or while waiting to board one at stations. This metric needs to be evaluated and improved, if needed, during system design and/or operation. To that end, the LoS concept can be used. LoS is a term in the context of transportation planning [270] and represents the layering of QoS in classes, as a quantitative index for the overall measured performance of a service, facility, or means of transport, from the viewpoint of the passenger or the service provider [141, 144, 148]. In essence, employing a relatively small number of LoS grades, the service operator can characterize a wide range of structural and operational aspects of the RTS, like comfort.

However, optimizing the RTS design or operation solely from the QoS/LoS perspective is not sufficient. Such systems are run either by private companies or as public services, which need to account for the underlying operational costs. In fact, costs are directly connected to comfort-related metrics through the system settings that satisfy the latter; e.g. putting more trains on the line to account for increased passenger load may alleviate discomfort, but at the same time it requires more equipment, drivers, and in general, increased operational costs.

Therefore, achieving a certain level of spatial comfort entails making a strategic decision that involves the achievement and improvement of comfort under a certain operational budget. This is essentially the problem that we consider in RTS design.

4.1.1 Passenger comfort as a key Level of Service (LoS) indicator in RTS

LoS, in the context of the RTS, can define service levels at critical associated locations, like trains or platforms/stations (while passengers are commuting). Typically, LoS is divided into six levels or classes, from "A" to "F", with "A" representing the best quality of service, and "F" corresponding to the worst situation. These definitions are included in [67] and are based on the Measures of Effectiveness (MoE) [257]. The latter quantify the obtained results and can be expressed as probabilities that the system will perform according to the given requirements.

The EN 13816 Standard [59], constituted by European Committee for Standardization (CEN), contains QoS standards in public transportation, focusing in eight main criteria, i.e. availability, accessibility, information, time, customer care, comfort, security, and environmental impact; LoS levels have been standardized for some of them. In particular, significant importance have the (i) passenger comfort, e.g., the average available space for individuals (passengers) in the trains or on the platforms, as a MoE [135], and the (ii) train frequency, e.g., the departure frequency of trains and their on-time arrival to stations, as the primary measure of availability and time [40].

In [222], an overview of capacity definitions, alternative analysis approaches, and toolsets suitable to evaluate capacity, are provided, relevant to the evaluation of a RTS, in terms of LoS. Moreover, a capacity evaluation process for the lines, using the numerical values/percentage of passengers and unit trains, as well as, the required LoS of each train type, is proposed [249]. In a similar context, three approaches for decision support are addressed [299], illustrating the impact of high-speed passenger trains on freight trains in shared corridors. Last but not least, the main

TABLE 4.1: Passenger comfort LoS (a) at platforms & (b) in trains.

LoS	Definition	Passenger space (m^2) at platforms		Passenger space (m^2) inside trains	
		Greater-than or equal-to	Less-than	Greater-than or equal-to	Less-than
A	Free movement	1.21	-	1	-
B	Restricted movement	0.93	1.21	0.85	1
C	Personal comfort	0.65	0.93	0.60	0.85
D	Occasional contact with others	0.27	0.65	0.27	0.60
E	Contact with others	0.19	0.27	0.19	0.27
F	Frequent contact with others	0	0.19	0	0.19
		(a)		(b)	

TABLE 4.2: Train frequency LoS.

LoS	Definition	Average headway (min)	Trains/h
A	Passengers do not need schedules	8-10	7-8
B	Frequent service	10-14	5-6
C	Maximum desirable time to wait if a train has been missed	15-20	3-4
D	Service unattractive to choice commuters	21-30	2
E	Service available during the hour	31-60	1
F	Service unattractive to all commuters	60	1

objective of [216] is to investigate the effect of a provided passenger inter-modal facilities LoS to the behavior of the passengers/commuters.

According to railway transportation standards, comfort LoS is used to relate the transportation quality to a given flow rate of commuting passengers [77]. Thus, it is thought to be closely related to the capacity of the platforms and the trains. The platform acts as a "buffering" area for passengers, on which they wait for the next train, while the train is the means of transport, moving between the stations and their respective platforms, affecting the "buffer". Comfort is measured via the available space around passengers either while waiting at a platform to embark on an incoming train or while standing inside a train, commuting to another station (and thus, another platform).

The comfort LoS, in terms of available space, is based on Fruin [77] principles for LoS measurement, corresponding to different levels of crowding. Table 4.1 summarizes the mapping of the different comfort LoS to their respective definition, i.e. the degree of allowed mobility of passengers while waiting at platforms or standing inside a train. Part "a" of Table 4.1 depicts the graded (set by upper and lower bounds) classification of comfort LoS at the platforms, i.e. the available passenger space at platforms, measured in square meters (m^2) per passenger. Note that the passenger comfort LoS in trains is also formed according to the Fruin scale, albeit with a different gradation due to the nature of the limited space of a train (shown in part "b" of Table 4.1).

In the Transit Capacity and Quality of Service Manual (TCQSM) [135], a table where LoS is graded based on different headway classes for incoming trains, is defined (see Table 4.2). The headway is the time between the trains, indicating the frequency of the transportation service, thus showing how long the passengers wait on average to embark on a train.

Since the comfort of passengers is considered a key metric of the effectiveness of RTS, it is selected to appear in the examples presented in the following sections. However, different QoS criteria such as the ones identified in [59] (like trains frequency - time) could also be classified with

the same logic and be integrated into the system, provided that corresponding LoS is standardized.

4.1.2 Cost analysis in RTS

In the RTS context, system operators aim to provide efficient transportation services to the commuters, in multiple dimensions [59]. One of those dimensions is the spatial comfort of passengers while waiting at a stop or within a moving train; this comfort is quantified in terms of LoS (see Section 4.1.1), and can be measured under varying traffic conditions.

However, altering the operational conditions of a RTS to achieve better LoS, e.g., increasing the frequency of train routes during peak business hours, comes at an extra –operational– cost. The increase in operational costs is associated with the additional number of train routes which may involve more drivers (and thus, paid shifts), more energy or fuel consumption, and more equipment or maintenance costs [103]. Therefore, RTS operators require the evaluation of the benefits in comfort versus the –relative– cost increase, needed to yield these benefits. Typically, a RTS operates under specific budgetary constraints; therefore, cost drives company-wide decisions. Both LoS and cost need to be studied in concert with each other during system design [260]. To this end, we focus on addressing a fundamental challenge in large-scale RTS, which is the need for adapting operations to satisfy commuters (e.g., in terms of the available space within trains and stations), while minimizing the additional operating costs that this adaptation involves. This can be achieved via a comprehensive MBSD cost analysis that facilitates a balance between the provided service quality and operating costs that occur, at the scale of large commuting populations.

4.2 Applying MBSD in RTS

4.2.1 Basic concepts

In order to apply MBSD to the RTS domain and build domain-specific models, SysML is adapted to accommodate domain-specific aspects; this is done via the profiling mechanism. In particular, based on our proposed framework (described in Section 3), we construct a RTS SysML profile that includes QoS entities, cost entities, and structural entities, all related to the domain under study. The way the RTS profile was built to facilitate MBSD in the transportation domain is shown in Figure 4.1. First, we applied the general SysML profile (white-colored) to RTS, which contains basic

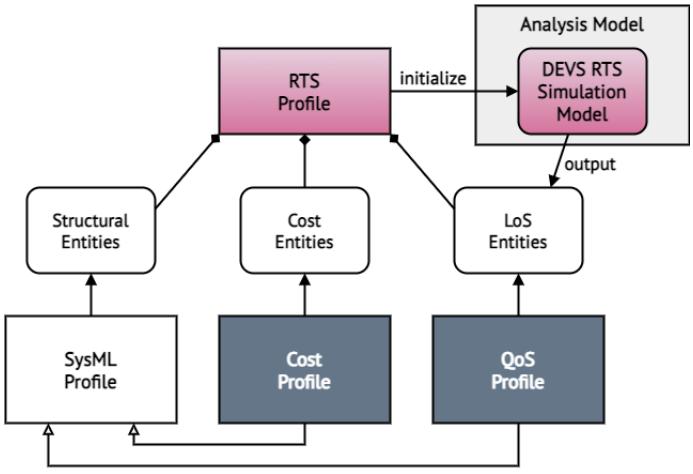


FIGURE 4.1: Applying profiles to RTS.

stereotypes such as blocks to describe structural RTS entities (such as trains, stations, etc.). Second, we applied our QoS profile (grey-colored), which contains QoS-related concepts such as graded quantitative requirements to define and manage LoS. Third, our cost profile (grey-colored) is applied to the RTS profile to define and manage cost entities, which in turn should satisfy cost-related requirements. In Section 3.1.1, we pointed out that a RTS can be analyzed using simulations. In this case, a DEVS simulator with DEVS libraries for RTS is used (see Section 3.2.1).

The way the RTS profile has been constructed to facilitate MBSD in the RTS domain is shown in Figure 4.1.

The process for LoS exploration is presented in the following. First, we provide an overview of the process, then we describe specific modeling artifacts and their interrelations, and finally we summarize its outline.

4.2.2 Overview of LoS exploration process

SysML was introduced as a standard to serve MBSD [72], e.g. become the primary tool to built models and interact with the system engineer. However, the concept of LoS is not directly addressed by the current version of SysML [261]. Moreover, quality metrics are not supported with standard notation, i.e. specific SysML elements. Thus, additional design/modeling elements may be needed to represent specific concepts of system design, like QoS/LoS requirements. For this purpose, SysML itself provides an extension mechanism to create new elements (see Section 2.3). The need to enhance and extend SysML features to enable QoS modeling and virtual exploration of achievable QoS is characterized as a critical issue [83]. Note that relevant efforts are recorded, such as [129], where an attempt to define guidelines for modeling quality of service parameters and the way they might be represented within SysML, is provided.

To enable effective exploration of LoS metrics using SysML, we propose that all related information is defined, computed and stored within the system modeling environment, utilizing both static properties and dynamic aspects of the system-under-study, as depicted in Figure 4.2). Hence, properties such as coherence, synchronization and traceability between LoS aspects and the main system model, are retained. This provides the capability to (i) define LoS properties, based on specific elements of the core system model, (ii) estimate expected LoS of specific system model configurations, and (iii) explain LoS estimation results in regard with specific elements of the core system model.

Apart from the static description of the system (structural and behavioral definition), the system model should also include attributes to store estimated indicators, related with the dynamic aspects of the system-under-study (actual expected behavior, e.g. performance) under specific operational conditions. This is depicted in the *System Model* dotted rectangle in Figure 4.2 and is expected to reside in a single, standard, and machine-usable form, i.e., a SysML model, using an appropriate domain-specific SysML profile, as explained in Section 4.3.1.

LoS aspects of the system should be managed on top of an intermediate layer, containing simple and composite graded quantitative requirements on static and dynamic properties of the system model. This is depicted in the *Level of Service* and *Graded Quantitative Requirements* dotted rectangles in Figure 4.2. Traceability from LoS to specific system model properties is also illustrated, through the respective requirements.

Estimation of specific indicators related with the dynamic aspect of the system is achieved by applying the methodology presented in [124]; an *Analysis Model*, here, an executable simulation model (i.e. *DEVS Model*) is generated from the SysML system model and, subsequently, simulation results indicating the performance of the system components are generated with the execution of the corresponding simulation model. As illustrated in the lower part of Figure 4.2, the obtained results are propagated to the respective attributes of the system model.

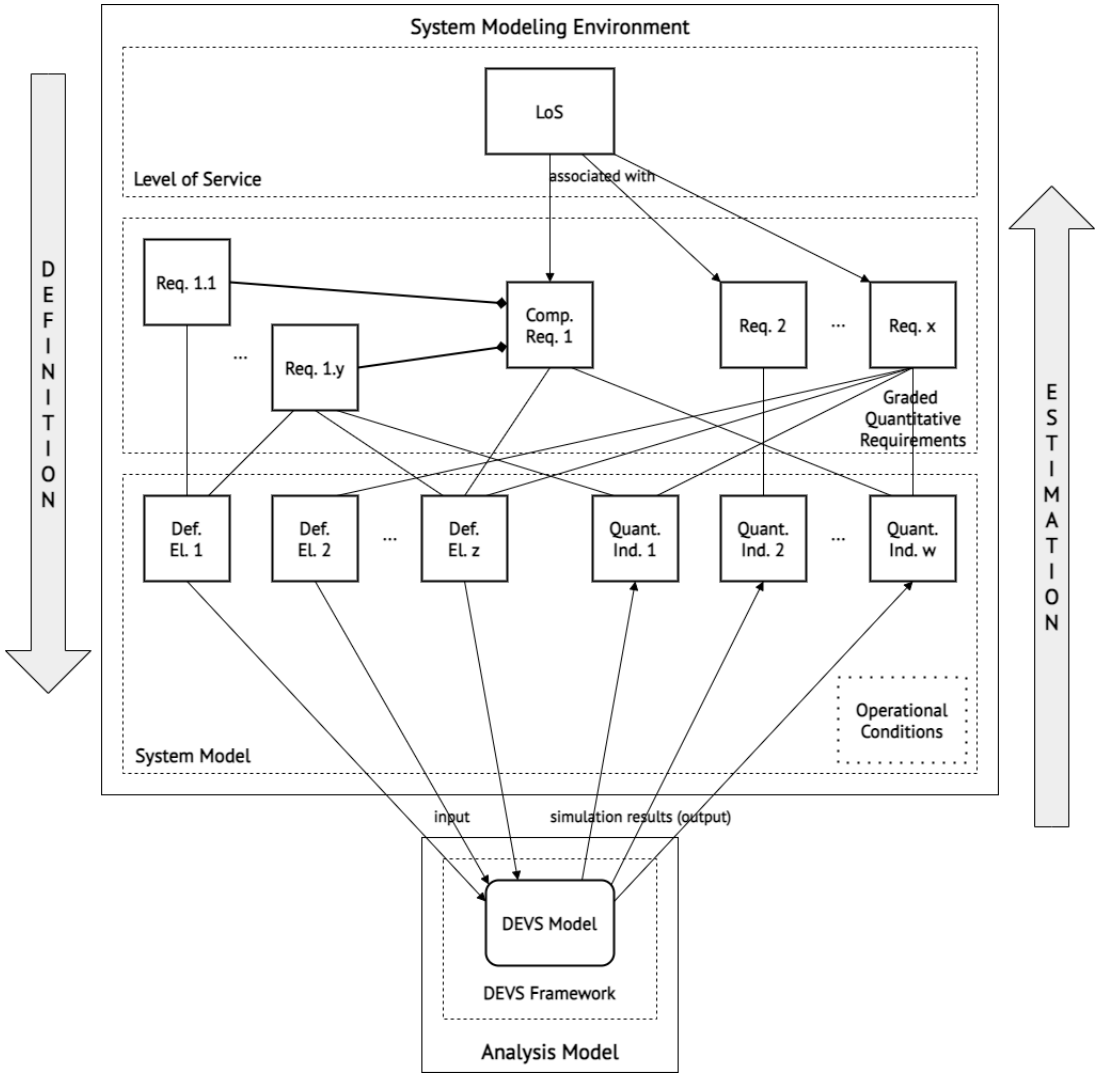


FIGURE 4.2: LoS modeling & exploration using SysML.

4.2.3 Modeling artifacts and process

Delving into a more detailed description of the process with the aid of Figure 4.2, the system model is considered as a set of *Definition Element* ("Def. El. i") nodes and a set of *Quantitative Indicator* ("Quant. Ind. i") nodes. These provide the system definition and a quantitative estimation of the system behavior, given specific *Operational Conditions*. A simple example of a definition element is the maximum capacity of a train, while its estimated average capacity is a relevant quantitative indicator. Definition elements can provide input to the analysis model in order to be executed. Indicators are not expected to be known a-priori, but can be the outcome of measurements, e.g., via simulation of the considered system model. Operational conditions of the system (e.g., "trains depart every 4 minutes") form the context in which the level of the provided service is investigated, estimated and evaluated.

In a top-down reading of the scheme, LoS is associated with specific graded quantitative requirements ("Req. i") or *Composite Requirements* ("Comp. Req. i") imposed on specific system model nodes (either definition elements or quantitative indicators). An example of a requirement is "the available space around a commuting passenger inside a train should not be less than 1 m^2". Composite requirements, like "Comp Req. 1", contain sub-requirements (e.g., "Req. 1.1", ..., "Req. 1.y"), in terms of a requirements hierarchy. An example of a composite requirement is "the comfort of the passenger in terms of space depends on the available space when the passenger moves with a train or stands on a platform".

In a bottom-up view, once simulation results are incorporated back into quantitative indicators of the system model, an estimation of the expected LoS is feasible, based on the verification of requirements, also enabling LoS verification.

4.2.4 Outline of process

Having introduced the considered modeling artifacts and their interrelations, the outline of the proposed process is summarized as follows:

1. The system model is defined using a domain-specific SysML profile. Operation conditions and quantitative indicators are also included.
2. Automated simulation execution is enabled applying the approach of [124].
3. LoS is defined based on specific simple and composite requirements.
4. Once the system definition and operation conditions are baselined, simulation results are fed back into the system model after the automated simulation execution.
5. Requirements are automatically verified, enabling LoS verification.
6. In case the desired LoS is not achieved:
    - appropriate information regarding system model elements that degrade LoS is provided,
    - suggestions to improve LoS are provided,
    - the process is iterative, i.e. it can start over after adjustments in system definition and/or operational conditions.

4.3 Applying QoS meta-model

In this section, we apply our QoS meta-model to conduct LoS analysis of RTS, ensuring that the required passenger comfort level can be achieved. In essence, using this meta-model, an RTS engineer employs SysML to efficiently model LoS requirements that a RTS (already described in SysML) should satisfy, within a single modeling environment (e.g., CSM), and runs simulations to compute comfort LoS under different conditions. This corresponds to the RTS exploitation, focusing on performance parameters and technology (e.g. infrastructure) constraints, according to Wymore [293]. The benefit in this case is that the engineer only interacts with SysML RTS models within the modeling tool and should not familiarize with the simulation tool interface or the simulation models; tool integration and model transformation are hidden from the engineer and are performed automatically.

In Section 4.3.1, we illustrate and describe an overview of a generic RTS SysML profile. We present basic structural elements of a RTS model, corresponding to real-world RTS entities in Section 4.3.2. In Section 4.3.3, we present the constructs used for denoting LoS requirements in SysML models. We analyze LoS estimation, based on RTS simulation results in Section 4.3.4. Finally, we describe the definition and verification of the desired LoS in Section 4.3.5.

4.3.1 RTS SysML profile to explore LoS

Profiling is the basic extension mechanism to customize SysML for a specific domain [5], in this case, RTS. A SysML profile for the description and modeling of RTS is proposed. This profile focuses on the definition of structural RTS elements, for example trains or stations, as well as the specification, estimation and verification of LoS, exploiting simulation analysis models and simulation results.

The profile diagram of Figure 4.3 illustrates stereotypes of the RTS profile, used to define the system model for RTS. These stereotypes are used in SysML diagrams during system design to develop RTS system models with LoS capabilities. Note that we apply our QoS meta-model to this profile, thus, concepts of the meta-model are transformed into specific stereotypes in the profile. The profile includes three groups of entities:

- Structural entities, defined as stereotypes of the QoS meta-model's *System Component*, that represent the real entities of a RTS (e.g., trains). These components correspond to the structural elements of Figure 4.2, and are depicted in Figure 4.3 as grey-colored. They have two types of value properties: (i) descriptive, which are used to describe the corresponding entity (e.g., capacity of a train), and (ii) others that hold LoS estimations and are computed based on simulation results. These components are the following: *Line*, *Passenger Generator*, *Station*, *Stop*, *Train*, and *Route*; they are described in detail in Section 4.3.2.

- LoS metrics, defined as stereotypes of the *Graded Quantitative Requirement* entity, the primary concept of the QoS meta-model (see Section 3.4.1), for the definition of LoS. In the RTS domain, composite LoS requirements, comprising sub-requirements in terms of requirement hierarchies, can be defined. *LoSFrequency* and *LoSComfort*, described in section 4.3.3, are the LoS metrics of the profile. In Figure 4.3, they are depicted as blue-colored and are associated to lines and stops. LoS metric estimations, which are computed based on simulation results, are stored within corresponding entities, while desired values of LoS metrics (e.g., levels) are defined using corresponding quantitative requirement stereotypes.

- Simulation-based verification entities, defined as stereotypes of the *Simulation-Based Verification Entity* of our QoS meta-model (see Section 3.4.1), that are used to evaluate their respective model components (e.g., the *SimStop* evaluates stops). They are also blue-colored

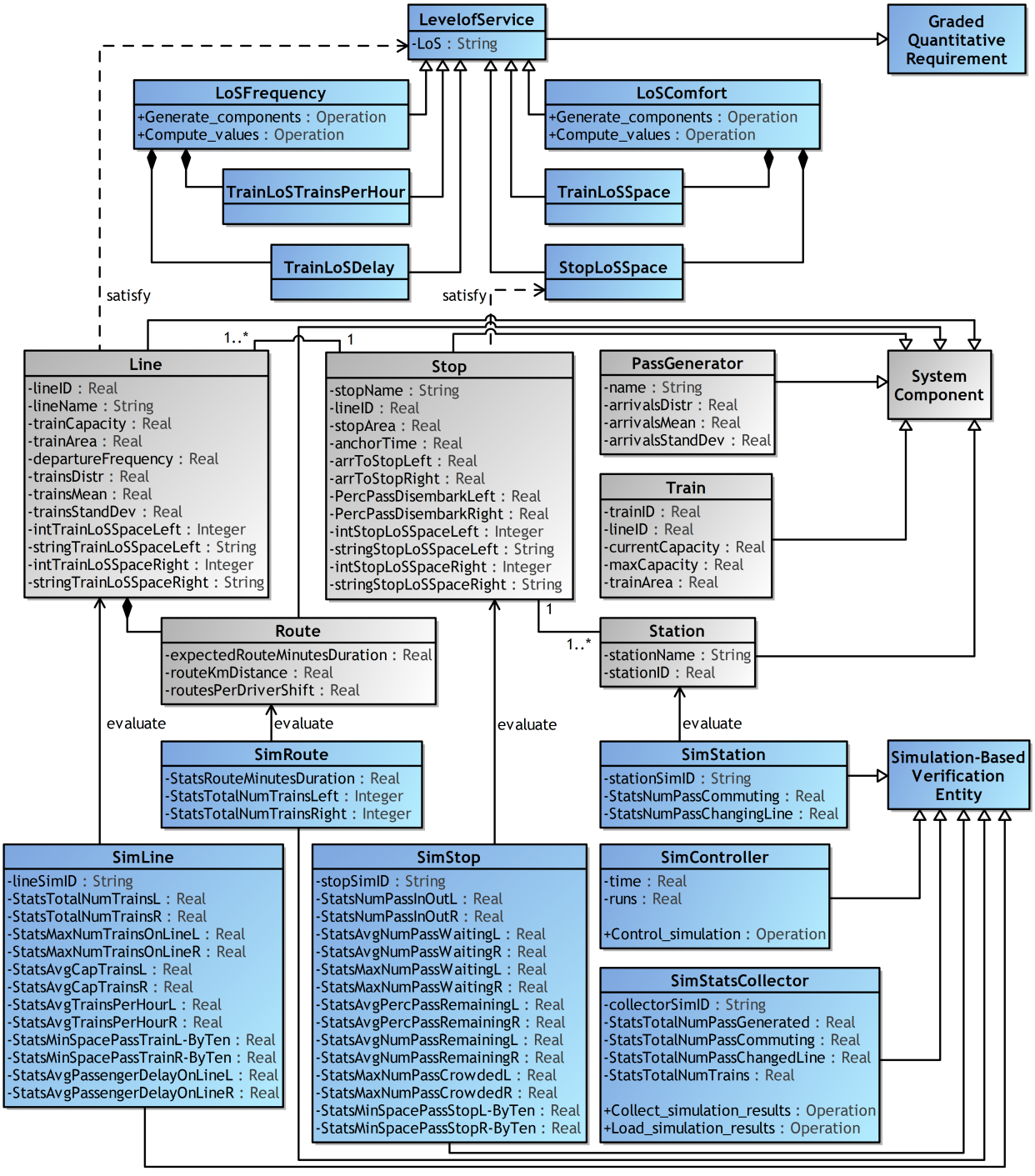


FIGURE 4.3: RTS SysML profile.

entities, as depicted in Figure 4.3, and they either hold simulation results or simulation input parameters. They facilitate the interaction with an external RTS simulator, like DEVS. They correspond to the quantitative elements of Figure 4.2. These elements are described in detail in Section 4.3.4. After completing the simulation of a specific RTS model, desired LoS requirement properties are compared to LoS estimations stored within corresponding RTS entities to explore whether LoS quantitative requirements are met and fulfilled.

4.3.2 Structural RTS Stereotypes

In this section, we describe the role and functionality of each of the structural stereotypes, and explain their most important properties regarding this functionality.

Line. It comprises a sequence of stops; it is also the element where trains are considered to be generated at and move along the routes of the line. When a train is generated, it is driven and positioned to the first stop of the line. Afterwards, it follows the line's sequence of stops – running along the rail track and following specific routes– in its assigned direction until it reaches the respective stop of a terminal station. Lines have input attributes, e.g., *name* and *id*. Since lines serve also as train generators, and trains move along the lines, the line entity also holds attributes relevant to the specifications of trains, e.g. maximum capacity or departure frequency.

Passenger Generator. It represents the actual entrance of the passengers at the stations. Given that in the real system, the number of passengers and the time they enter is undetermined, passenger arrivals are created randomly by the generator, according to a probability distribution.

Station. It refers to the area where passengers are validating tickets before entering a stop's platforms or exiting the RTS. The primary input attribute of the station is its name. Based on [270], stations are categorized as:

- initial or terminal that comprise a single stop where all new generated passengers embark and all passengers arriving with the train disembark (first or last stop in the line sequence).
- ordinary, comprising a single stop that is intermediary within the line sequence where a random number of the passengers arriving with the train disembark and all waiting passengers embark, provided that there is sufficient space in the train.
- transit that contain more than one stops, connecting multiple lines. Upon reaching one of these stops, passengers can (a) move to other stops and, subsequently, change lines, apart from (b) leaving the RTS through a station's exit.

Stop. It corresponds to the platforms of the RTS. Each stop represents a set of two opposite-direction boarding platforms (left and right). Therefore, a stop is the area where passengers are accumulated in order to board an incoming train, or disembark, in either direction of a line. In the case of a terminal station, the stop is used only for disembarking. Passengers disembark from an incoming train, leave the platforms, enter the corresponding station, and either exit the system or change line on a transit station. Each stop belongs exclusively to a specific line. Among stop input attributes are its name, the available area in square meters (m^2), the time interval for a train to arrive from the previous stop for each direction (left and right), as well as a train anchor time. All times are expressed in seconds.

Train. Trains, carrying passengers, are moving parts of the RTS, providing the transportation service. Specifically, after the passengers embark, the train transports them to the next adjacent stop, following a specific route (line). Its attributes, e.g., *trainID*, *lineID*, maximum capacity or current capacity, and available area in square meters (m^2), are used to characterize it. Note that the current capacity of a train is updated according to the number of passengers embarking or

disembarking at a stop. Passengers are not identified and treated as individual, autonomous entities; instead, their aggregate behavior is modeled in stations, stops and trains via respective numeric values that denote the result of overall passenger entrances and exits. When a moving train reaches a stop, attributes storing numbers of passengers are updated accordingly, i.e., the aforementioned train capacity or the number of gathered people at the stop are increased or decreased.

Route. This entity represents the course of trains, traversing a line, from its starting stop to the final one. It is worth mentioning that routes "glue" together lines, trains and stops, and are important both for a LoS and a cost analysis, as shown in the following sections.

4.3.3 Passenger comfort LoS definition

LoS metrics are modeled as SysML requirements; they are individual or composite text-based model entities that form hierarchies and can be satisfied or verified by specific model entities or other requirements, indicated through respective relationships (see Section 3.1.2).

In the proposed RTS SysML profile (Figure 4.3), LoS requirements are outlined as stereotypes, extending the graded quantitative requirement stereotype (which in turn is a SysML requirement stereotype, based on our QoS meta-model). In this profile, the SysML LoS requirements are classified, forming a hierarchy.

The *LevelofService* requirement is the abstract, top-level requirement that indicates the overall level of the RTS services. The addition of a LoS attribute is used to maintain the desired LoS value. LevelofService has two sub-requirements, namely, *LoSComfort* and *LoSFrequency* requirements. LoSComfort represents the general LoS for the comfort of the passengers, based on the available passenger space at a stop or within a train. Thus, it is composed of the *StopLoSSpace* and the *TrainLoSSpace* sub-requirements. The LoSFrequency requirement represents the LoS that is associated with the departure frequency of trains, and is composed of the *TrainLoSDelay*, i.e. LoS related to the remaining time for a train to arrive at a stop, and the *TrainLoSTrainsPerHour*, i.e. LoS related to the number of trains moving on a specific line and between the line's corresponding stops during an one-hour interval.

The basic operations of both the LoSComfort and the LoSFrequency requirements are their decomposition to corresponding sub-requirements, as well as specific LoS computation methods. All of the LevelofService's aforementioned sub-requirements inherit:

- a (sub-)identifier (e.g., LevelofService has id = "1"; its direct sub-requirements will have id = "1.1", "1.2", etc.),
- the same LoS, while the computation method and the defining metric bounds of each LoS requirement may vary.

4.3.4 Passenger comfort LoS estimation

The RTS SysML model, developed using the RTS profile, is utilized for the initialization of executable simulation models. After simulation execution, results are imported into the system model, utilizing simulation-related entities. For example, the simStop entity is used for the computation of the estimated LoS for all the corresponding structural entities, i.e. stops. This way, the computation of LoS metrics is performed within the system model, independently of the simulation process or tools, as prescribed in Figure 4.2. In the following, we explain the calculations performed during the collection of the simulation results, in order to obtain appropriate performance indications. We also present the proposed process for the computation of LoS metrics,

LISTING 4.1: Excerpt of the simulation results in XML format.

```
<results>
  <RESULT id="_17_0_58f01fa_..._131" name="Omonia L1" stereotype="Stop">
    <VALUE>
      <name> StatMinSpacePassStopLeft </name>
      <value> 0.18 </value>
    </VALUE>
  </RESULT>
</results>
```

based on these performance indications, stored within the system model in simulation-related entities for individual RTS elements.

RTS simulation-based verification entities

The RTS engineers interact only with a single design environment, i.e. a SysML modeling tool like MagicDraw [178] or its successor CSM [37], while the simulation environment is completely hidden from them [124]. Applying the simulation-based testing process as presented in Section 3.2.1 to the RTS, the engineers can define the RTS model using SysML (in particular the RTS SysML profile), specifying the RTS system model within the design environment. In the context of MBSD, the RTS model is used for the automated generation of a DEVS simulation model, defined via a standards-based QVT mapping. Eventually, an executable DEVSJava simulation code is produced, exploiting existing simulation library components [124]. The library components, suitable for the generation and execution of the simulation, as well as the management (e.g., collection or compilation) of the produced results, act as auxiliary entities, supported by external tools. For simulation purposes the framework presented in [124] is utilized.

The *Simulation Controller*, shown in Figure 4.3, is used to control the simulation, in particular its time and repetitions. In addition, it can initiate or terminate the operation of all the other entities via signals. A typical example is the signal sent, at the beginning of the simulation, from the controller to lines and passenger generators, informing them to start generating trains and passengers, respectively. The *Simulation Statistics Collector* holds its own *Stats* attributes, i.e. the total number of passengers that were generated, changed a line, and commuted inside the RTS, as well as the total number of produced trains. After the simulation execution, the collector is responsible for compiling, extracting and storing the simulation results in XML and Comma-Separated Values (CSV)-format documents; the XML format is suitable for the easy manipulation of data (e.g., the results) and their conveyance to the design environment. The collector supports the automatic complement of the SysML RTS model with the simulation results; its main operation is to *Load simulation results* to the system model.

The incorporation of the results in combination with defined requirements, related to LoS, allow the RTS engineer to figure out the entities and evaluate the system. The results, compiled after the simulation execution, represent total, average and minimum/maximum numbers about the commuting passengers and the moving trains, important for LoS verification. An excerpt of them, related to a specific stop (*Omonia L1*), is presented in Listing 4.1.

The XML format shows that inside the *results* element (that comprises all the collected data), a specific *RESULT* holds some characteristics of the Omonia L1 stop and contains multiple *VALUE* elements that represent simulation results. For example, the *StatMinSpacePassStopLeft* is the minimum available space around a passenger, waiting at the left platform of the stop.

After the execution of the simulation, all simulation results are collected and incorporated into simulation-related entities, populating their attributes. The statistical attributes of *SimLine* (depicted in Figure 4.3) are related to the number of generated and moving trains, as well as their average capacity. Moreover, this entity contributes the computation and validation of LoS, providing the average or minimum available space of the passengers inside trains. *SimStation* holds the amount of commuting passengers, i.e., the number of passengers entering the corresponding station (*StatsNumPassCommuting*), and the number of passengers that change lines (i.e. they move to different stops inside the station). The simStop produces more than ten key performance indicators (stats output attributes) with the average available space for passengers while waiting at the platforms of the stop, being the most important for the LoS computation and validation. Note that all the simulation/evaluation entities hold a "simID" attribute, referring to the identifier of the respective model component they evaluate. The *SimRoute* simulation-based verification entity evaluates route entities, holding the statistical values of the performed train routes (e.g., the total number of routes, performed on the left direction or the exact duration of the routes in minutes). This entity's statistics are informative for the cost analysis that will follow; e.g., the simulated number of routes is factored in while computing the total OpEx.

Passenger comfort LoS computation

LoS values are calculated based on the simulation results stored in simLine and simStop entities. Passenger comfort is calculated based on the space available for each passenger at trains and stops for each line. Average, minimum and maximum values of available space in stops are calculated based on simulation results stored as attributes, for example *StatsMaxNumPassCrowded*, *StatsAvgNumPassRemaining*, of the simStop simulation entity (see Figure 4.3) and stored as attributes of the same entity, for example *StatsMinSpacePassStop* storing the minimum space available for passengers in each stop. Corresponding passenger comfort LoS value in stops is calculated via the comparison of *StatsMinSpacePassStop* values, with the corresponding LoS bounds, defined in Table 4.1. For example, in the case of Omonia L1 stop, presented in Listing 4.1, the corresponding passenger comfort value is calculated as "D" . Available space for passenger within trains is calculated in a similar fashion.

The relevant algorithm (in pseudocode), applied after importing the simulation results, in order to calculate passenger comfort LoS values, is the following:

foreach *(LoS* ***in*** $A, ..., F$*)* **do**
  if $((PS \geq LB(LoS))\ \&\&\ (PS < UB(LoS)))$ **then**
    $estimatedLoS = LoS$;
    break;
  end
end

Algorithm 3: Calculation of space LoS.

where PS represents the available space for each passenger. The LB and UB, symbolize the lower and upper –Fruin– defined LoS bounds [77]. In the following, we present how PS at a stop is calculated. All instances of the passenger populations present at each platform of each stop (including the passengers boarding an incoming train, disembarking or remaining at the stop in case they do not fit) are accounted for whenever a train T reaches a stop S, traveling in the left or right direction d, at time $t\ \epsilon\ \Delta t$ (i.e. the simulation window of discrete time points under study).

Passenger Space (PS) at stop:

$$PS(S,d,t) = A(S,d)/SP(S,d,t) \tag{4.1}$$

where A stands for the area of the stop, measured in m^2 and SP is given by Eq. (4.2) below.

Stop Passengers (SP) upon train arrival:

$$SP(S,d,t) = PB(S,d,t) + PD(S,d,t) + PR(S,d,t) \tag{4.2}$$

where PB is given by Eq. (4.3), PD by Eq. (4.4), and PR by Eq. (4.5).

Passengers boarding (PB) the train:

$$PB(S,d,t) = \min(WP(S,d,t), TC(T,t) + PD(S,d,t)) \tag{4.3}$$

where WP represents the passengers waiting for the train to come, TC denotes the current train capacity, and PD is given by Eq. (4.4).

Passengers disembarking (PD) from train :

$$PD(S,d,t) = [MTC(T) - TC(T,t)] * P(S) \tag{4.4}$$

where MTC stands for the maximum train capacity, TC denotes the current train capacity, and P is the probability of a train passenger disembarking at this stop.

Passengers (potentially) remaining (PR) at the stop due to capacity constraints :

$$PR(S,d,t) = WP(S,d,t) - PB(S,d,t) \tag{4.5}$$

where WP represent the passengers waiting for the train to come and PB is given by Eq. (4.3).

Having passenger comfort LoS of each stop or commuting train at hand, the minimum and average passenger comfort LoS of each line may be calculated.

4.3.5 Desired passenger comfort LoS selection and verification

RTS engineers are facilitated (i) to select the desired LoS for a line (trains move along a line), and (ii) to define its computation method, i.e. every obtained stop or train LoS is compared with the average or minimum (worst) LoS. The latter is important, since LoSComfort is a composite requirement decomposed to StopLoSSpace and TrainLoSSpace requirements, satisfied by all the stops and trains, respectively, associated to a corresponding line. The outcome of the estimation process described above is utilized at this stage.

Individual RTS model elements that fail to deliver the required LoS, degrade the overall level of delivered service quality. Such elements are indicated in the modeling environment so that RTS engineers can focus on improving their performance. Automated suggestions are also provided to further facilitate this activity.

4.4 Applying cost meta-model

In previous sections, QoS and service levels are investigated, integrating and evaluating LoS properties within RTS models. In this section, SysML is extended, to enable the design and evaluation of a RTS, from a cost perspective. Specifically, we construct a generic SysML profile that encodes

the cost perspective, based on the Cost meta-model of Section 3.5. In essence, all Cost meta-model concepts are transformed into stereotypes of a custom *Cost profile*. To enable computations related to this perspective, we introduce a cost functions library, which is readily available to the designer during design time, along with the profile. To automate such computations, we employ parametric diagrams. The end results (i.e. the computed cost entity values) are in turn used to populate the parameters of cost profile elements; related requirements (e.g., conformance to budgetary constraints) are then verified by comparing the computed and desired values.

Applying the cost profile to system models can lead to an efficient system design and enhance the decision-making support of a system designer, facilitating her to identify areas for improvement; ideally, the designer should minimize or reduce the costs of the provided system services, while maintaining their high quality level for greater satisfaction and usage from the end-users. The profile is general and extendable enough to be applied in any system, leveraging the expressiveness and flexibility of SysML. This claim is further supported by applying the proposed SysML cost-related extensions, to the RTS domain.

Typically, in the context of the RTS, operators aim to achieve better LoS, e.g., increasing the frequency of train routes. However, more routes mean more trains, and thus more operating costs, while the RTS may operate under specific budgetary constraints; therefore, cost drives company-wide decisions. This is reflected with a specific threshold for acceptable costs beyond which increasing LoS is simply not viable. Thus, altering route frequencies requires two things: (i) a LoS analysis to ensure that the change achieves the required passenger comfort level, and (ii) a comprehensive cost analysis, in terms of incurred operating costs or OpEx.

RTS engineers use the RTS profile to apply MBSD on a RTS. Specifically, by employing SysML within a modeling environment, and running simulations, she can identify the required system parameters and conditions under which the desired LoS can be achieved. With this at hand, engineers employ the cost profile and cost computation mechanisms, which facilitate the assessment of OpEx incurred during peak hours for different train route frequencies.

The outline of the cost analysis is summarized as:

1. Cost entities are generated within the system model using the generic cost profile.

2. Following the definition of the cost entities, engineers can add and use functions, that are readily available in the cost inventory, to compute costs. In order to compute each cost value, SysML parametric diagrams, are employed. The equation takes input properties whose values are used to compute it; these properties may belong to different entities. The equation is solved within the parametric diagram (using OpenModelica [76] as the math solver) and the output cost value is extracted for further cost assessment. Note that using a parametric diagram within the modeling environment, engineers do not have to go through the complexity of writing additional code for the computations. The corresponding input values are simply provided while the engineers define the functions that will be used; after the computations are complete, the final required cost values are easily extracted and may be used for the cost analysis of the RTS.

3. Composite costs are computed. Since the aforementioned functions refine a respective cost entity, the computed output value is stored within these cost entities. A composite cost entity may obtain its value from the sum of the values of other cost entities via automatic summations.

4. Costing requirements are verified. In the aforementioned LoS analysis, LoS and LoS-specific requirements are used to evaluate the system. Here, the same principles are followed for the evaluation of the RTS from a cost perspective. Defined cost requirements are either

satisfied or not, leading to the system's cost evaluation. Specifically, a computed cost value is compared against the specified threshold and a result is extracted, i.e. "true" or "false", leading to the satisfaction of the requirement. The problematic –non-verified– requirement is annotated with a red-colored frame; this way, the engineer is informed about the defective element(s).

4.5 Case study: Passenger comfort in Athens Metro RTS

Athens Metro RTS, operated by the ATTIKO METRO, S.A. [9] and Urban Rail Transport, S.A. [277] provides public transportation in the metropolitan area of the capital city of Athens in Greece, populated with approximately five million people. The Athens Metro operators were mainly concerned about the provided LoS during operation, and more specifically the comfort of passengers. This triggered their interest in studying the RTS's behavior under different traffic conditions.

At the time of the study, Athens Metro comprised three lines, namely Line 1 "Piraeus–Kifissia", Line 2 "Anthoupoli–Elliniko" and Line 3 "Doukissis Plakentias–Aghia Marina" with sixty-one stations in total. Six of the stations are initial/terminal, i.e. all the passengers embark/disembark, four are transit, connecting multiple lines, and the rest of them are ordinary.

According to data provided by the Attiko Metro operation company, the six-car metro trains carry an average of 900.000 passengers during a typical working day. The highest volume flow of commuting passengers (nearly one third), is usually between 8 a.m. to 11 a.m. and during this period there is congestion in the Athens mass transit system, most notably in the metro network. Stops that serve popular areas have increased passenger accumulation. Hence, passenger space is restricted, causing frequent contact between the commuters. On the other side, stops that serve a low volume of passengers, allow movement with only occasional contact between them. Please note that the platforms of a stop have critical holding capacities; each platform contains nearly 300 m^2 of available space for the passengers. In case these capacity constraints are exceeded, passengers are at risk of falling to the rails. In addition, if a train's capacity is exceeded, passengers will have to remain at the stop waiting for the next. Each metro train has around 240 m^2 available passenger space in total, since it is composed of six cars/wagons and each car/wagon has 40 m^2 space. In order to adequately serve the passengers, while trying to prevent overcrowding and congestion in stops and trains, the Athens Metro operators estimated and set the average train departure frequency to "7" minutes in Line 1, and "6" minutes in Lines 2 & 3, respectively.

Despite the fact that the operators hold such compiled data, our approach offered them the following capabilities: (a) Computation of the actual LoS in stations and trains for all the lines, and (b) exploration of the desired LoS, in terms of existing conditions and possible recommendations to improve existing LoS.

4.5.1 Athens Metro modeling with SysML

First, the RTS model for the network was developed with the SysML notation using the RTS profile. This activity resulted in the generation of a valid and complete model, representing the transportation network and enabling further exploitation. An Athens Metro RTS model excerpt is illustrated in Figure 4.4, where the RTS's basic entities (white-colored), i.e. *Line*, *Passenger Generator*, *Station* and *Stop*, are defined. Specifically, a line (*Line 1*), three indicative stations (*Thissio*, *Monastiraki*, and *Omonia*) and their respective passenger generators and stops (*Thissio L1*, *Monastiraki L1*, and *Omonia L1*) are connected with each other. Note that, in this case, the station and stop entities belong to *Line 1* and they are adjacent; *Thissio* to *Monastiraki* and *Monastiraki* to *Omonia*. In

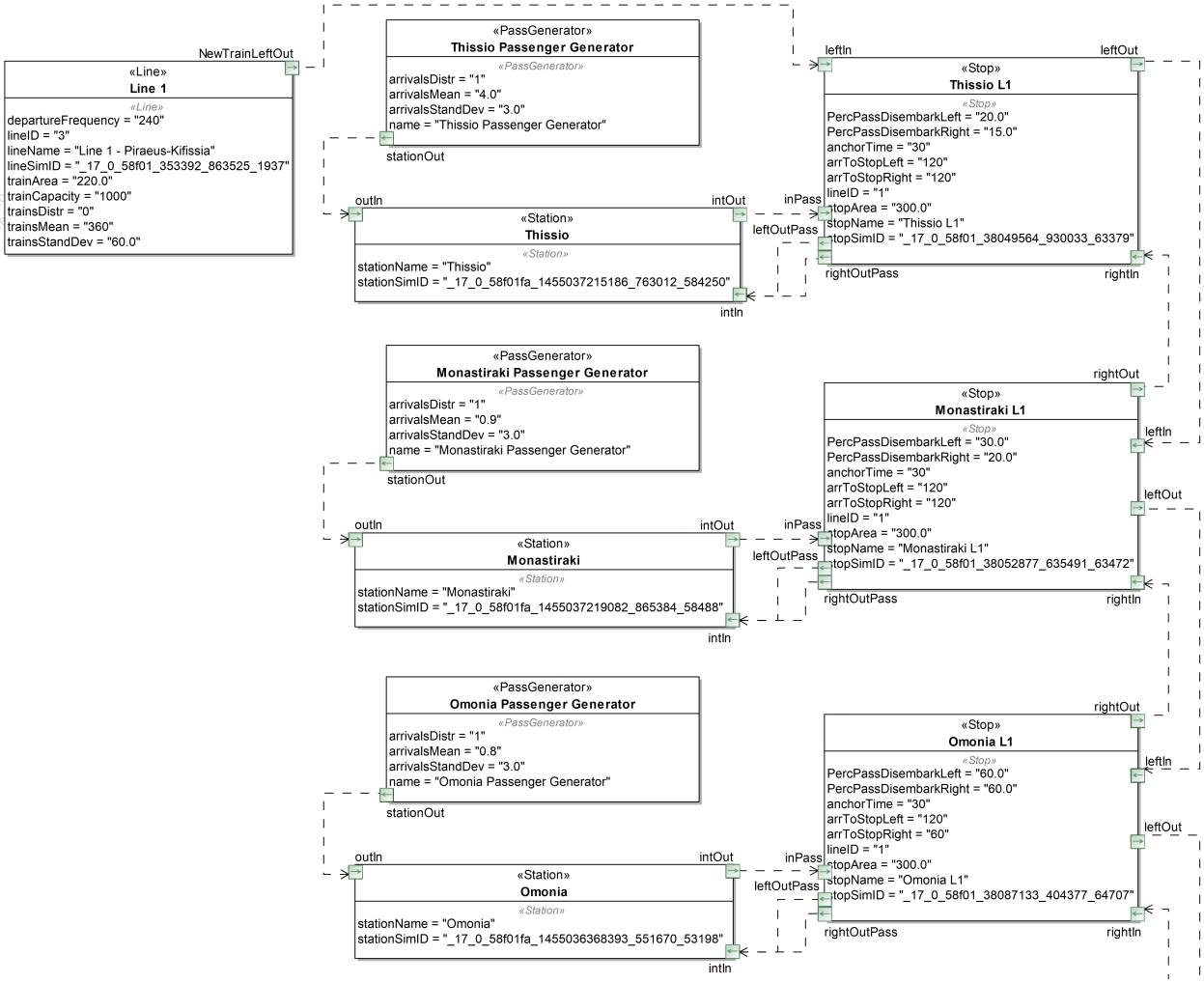


FIGURE 4.4: Excerpt of the Athens Metro RTS model.

addition, both *Monastiraki* and *Onomia Stations* are transit, connecting *Line 1* with *Line 3* and *Line 1* with *Line 2*, respectively; *Lines 2* and *3* are not depicted in the figure.

As shown in the excerpt, SysML *ports* are used as connection points of RTS model elements. They are connected via *information flows* that model the conveyance of material or information, establishing the overall connection between the entities. Connected *ports* among a *Line* and a *Stop* (e.g., *NewTrainLeftOut*, *leftIn*) or these between *Stops* (e.g., *leftOut*, *rightIn*) represent the rails where the *Trains* are moving, i.e. enter and exit a *Stop*. Additionally, *ports* connecting a *Passenger Generator* to a *Station* (e.g., *stationOut*, *outIn*) and the *Station* to its respective *Stop* (e.g., *leftOutPass*, *intOut*) are used to implement the flow of commuting passengers from one entity to the other (e.g., enter a *Station*, exit a *Stop*). Moreover, each entity has specific *attributes* suitable to characterize it, e.g., the *lineID* is the *Line*s unique identifier, *stopArea* holds the overall available space at a *Stop*, etc. These attributes are populated with appropriate values, within the design environment, by the RTS engineer.

4.5.2 Athens Metro LoS estimation

Estimation of the LoS of the RTS with its current architecture, configuration and operation parameters requires static model information and accurate operation performance indications. The

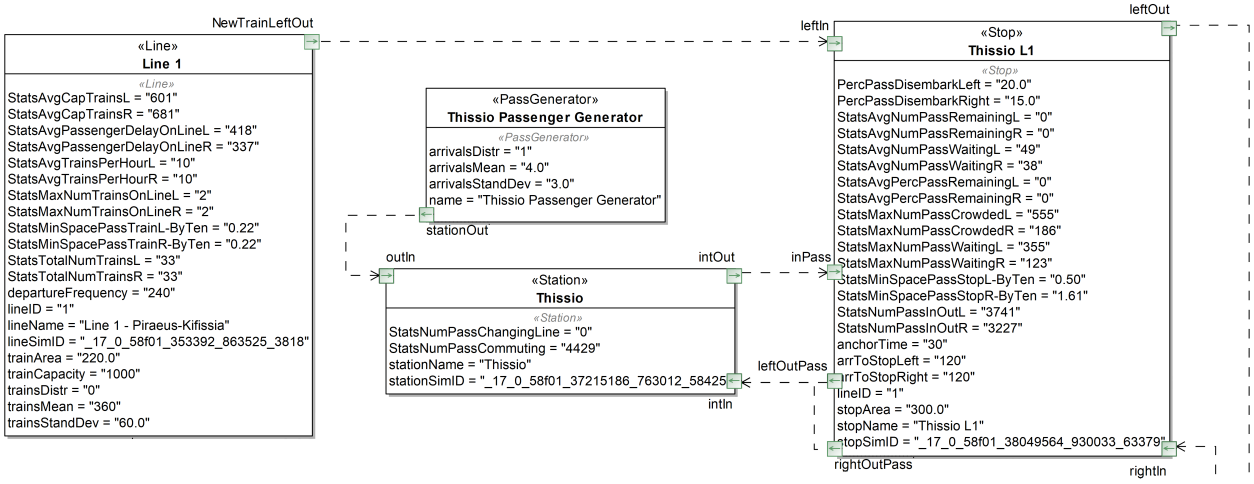


FIGURE 4.5: Excerpt of RTS elements with incorporated simulation results.

latter are calculated from measurements obtained during experimental simulation execution.

Simulation for Athens Metro model

The RTS engineer can configure the simulation parameters via the simulation controller element, defining the simulation's duration and repetitions. The executable simulation model is generated based on system model elements and their interconnections. Available auxiliary simulation components are also utilized. The *Transportation Plugin*, developed to support this research, supports the automatic incorporation of the results, obtained after the simulation completion, into the SysML RTS model. The engineer needs only to exploit the *Statistical Data Collector* via the design environment, in order to load the simulation results in specific attributes of the system model entities.

Cumulative results are applied to the corresponding entities as new attributes, specified by the profile. Figure 4.5 shows a RTS model excerpt, where new specific attributes have been generated and populated with results obtained for the *Line*, *Station* and *Stop* entities (e.g., *StatsMaxNumPassWaitRight*, etc).

Athens Metro LoS computation

The completion of the incorporation of simulation results in the system model elements triggers computation of the RTS LoS, which is performed automatically, as a background process. The result of LoS computation is available to RTS engineers. Specifically, the automated process provides the measurement and classification of the LoS of every *Stop* and *Train*. The average LoS, i.e. the *Train* or *Stop* LoS sum divided by their total number, and the worst LoS, i.e. the minimum LoS, are also computed. After the LoS computation, new attributes generated in the corresponding entity (e.g., *Line*, *Stop*), are populated with the service levels and shown to the engineer (e.g., *stringStopLoSSpaceLeft* in *Stop*).

Indicative result: Current LoS

The computed passenger space LoS for *Line 1 Stops* during rush hour is depicted in Figure 4.6. A two-column Heat Map shows the different service levels on the left and right direction platforms of the *Stops*. An interesting observation is that *Stops* that serve populous areas (e.g., Monastiraki

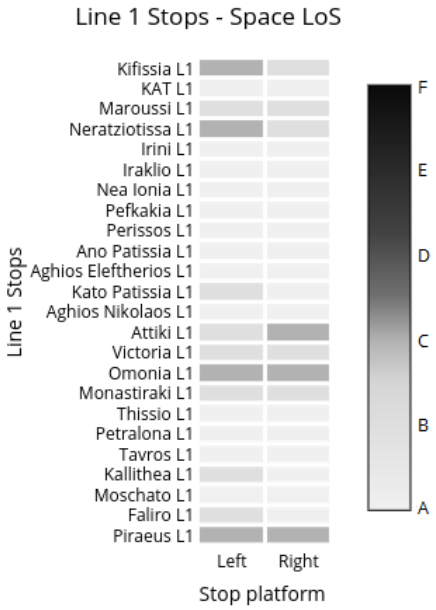


FIGURE 4.6: Computed space LoS of all stops in line 1.

L1), are overcrowded, resulting to the degradation of the passenger comfort LoS (dark-colored). Similar *Stops* like *Omonia*, etc, and their respective platforms (left or right) hold level "C" or worse, while others have LoS "B" and higher (light-colored).

Additionally, when a *Train* arrives at a *Stop*, the gathered passengers board it, increasing its occupancy. The diagram of Figure 4.7 shows the percentage of the *Train* commuting instances, taking into account their average service level ("A" to "F"), during the peak hour. Note that an instance is the commuting of a *Train* through a specific *Line*.

According to the Attiko Metro, *Trains* on *Line 1* are generated every "6" minutes, while the departure frequency of *Trains* in *Line 2* and Line *Line 3* is set to "4" minutes. We can observe that the majority of *Trains* traversing *Lines 2* and *3* hold high levels of LoS "A" and 'B" (light-blue- and dark-blue-colored), e.g., the right platforms of *Line 2* and *Line 3* hold LoS "A" and "B" close to 70%. On the contrary, *Line 1* -the oldest Athens Metro route with increased *Train* departure frequency- holds low LoS "A" or "B" percentages (nearly 30%) and increased "C" and "D" service levels (up to and including "D" level exceeds 80%). Based on the observed results, the worst is LoS = "C" at *Station Omonia*, while the average is LoS = "A" and "B".

We deduce that the current Athens Metro infrastructure and its operational parameters are quite modern and well-provisioned, requiring only a few improvements at focal *Stations* of the network in order to reach excellent LoS for the average passenger during the a.m. peak hour. In the following Section we illustrate these improvements, guided by the solutions proposed to the RTS engineer by the enhanced functionality of the design environment.

4.5.3 Athens Metro desired LoS verification

LoS verification presupposes the definition of the required LoS through the declaration of the respective requirements. The required LoS can then be verified against the estimated LoS, providing information regarding both individual problematic RTS model elements and aggregate performance indications.

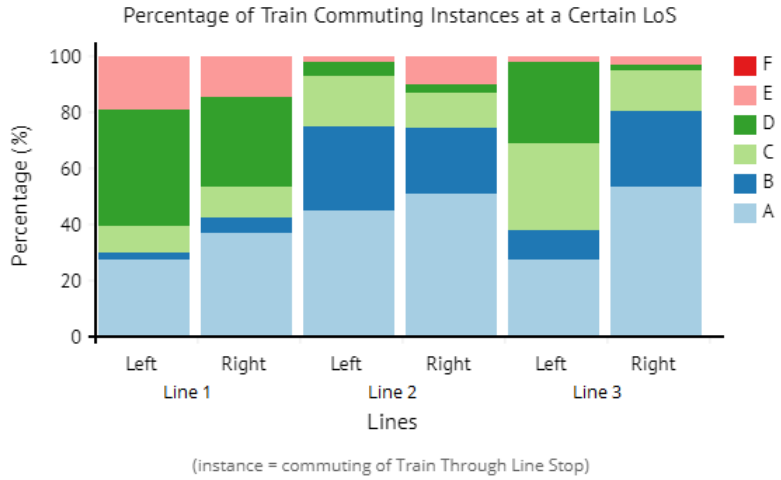


FIGURE 4.7: Percentage of train commuting instances at each LoS for all lines and directions. Departure frequency: "6" minutes for line 1 and "4" minutes for lines 2–3.

Athens Metro LoS requirement definition

In order to specify the RTS's services levels, the Athens Metro RTS model was extended. According to our approach, the LoS for passengers' comfort is modeled as a SysML LoS requirement. This specific comfort requirement is characterized by a unique id and a short textual description, while a *LoS-Comfort* attribute indicates the overall comfort LoS that the system engineer desires for the RTS. For example, the desired LoS for *Line 1* can be set to "B", while it is calculated as the average of LoS of all the *Stops* on this *Line*. *LoSComfort* is a composite requirement that can be analyzed to the *StopLoSSpace* sub-requirement, satisfied by all the *Stops*, and the *TrainLoSSpace* sub-requirement, which is satisfied by all the commuting *Trains*. Figure 4.8 illustrates the decomposition of the general comfort requirement. General LoS requirements are defined and connected with *Lines* by the RTS engineer, while the corresponding LoS requirements for *Stops* and *Trains* are automatically generated, as described in the following.

The model elements, e.g., the *Line*, are connected to the requirements, e.g., *LoSComfort* via *verify* relationships. Moreover, *containment* relationships are used to connect the *LoSComfort* to its sub-requirements, indicating the decomposition and the LoS requirement hierarchy. When the *StopLoSSpace* and *TrainLoSSpace* sub-requirements are generated, they acquire a unique id attribute based on their parent's id. For example, the composite *LoSComfort* has *id* = "6", thus, the *StopLoSSpace* that was created first obtains *id* = "6.1", while the *TrainLoSSpace* sub-requirement, created immediately after, will have *id* = "6.2". In addition, the sub-requirements inherit the desired LoS and store it in specific attributes (e.g., *LoSComfort* has LoS "B", thus, its *TrainLoSSpace* child has a LoS attribute that holds "B"). The *LoSComfort* requirement has attributes for the average and the worst LoS values, obtained during LoS calculation.

Due to the fact that a *Stop* is owned exclusively by a specific *Line* (e.g., the *Monastiraki L1 Stop* is owned by *Line 1*), the *Line* can verify the composite *LoSComfort* requirement. Additionally, the *Line* acts as a *Train* generator. When a *Train* is generated, it is positioned on the first *Stop* of the *Line* and then follows its sequence of *Stops* via the rails. Thus, the *Line* can verify the *TrainLoSSpace* requirement. In summary, the *Line* verifies both the *LoSComfort* and the *TrainLoSSpace* while the *Stop* verifies only the *StopLoSSpace* requirement.

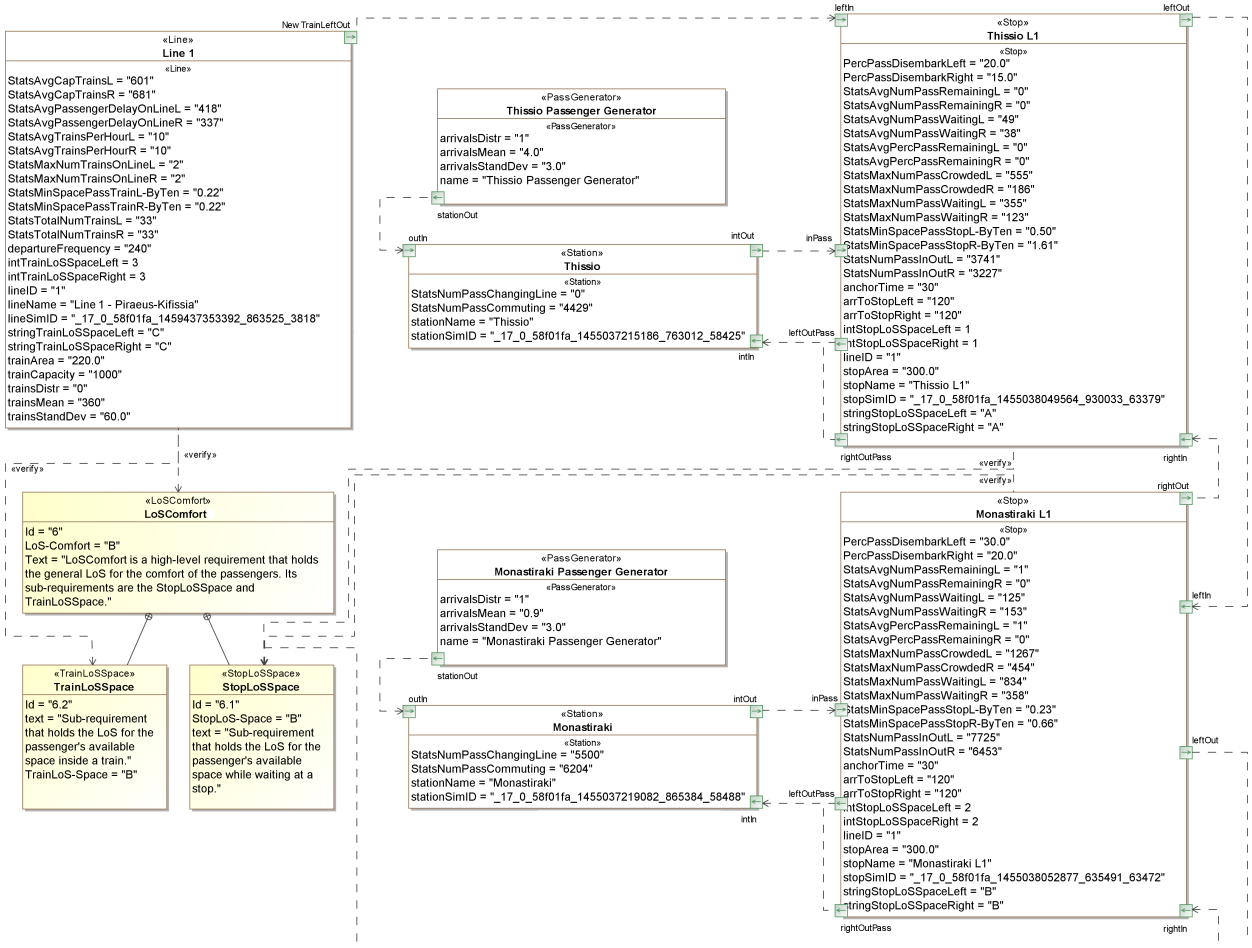


FIGURE 4.8: LoSComfort decomposition to sub-requirements (i.e. TrainLoSSpace and StopLoSSpace).

RTS LoS verification and improvement suggestions

Once LoS requirements have been defined and simulation execution has been completed, LoS verification can be performed. In Figure 4.9, an example of LoS verification is depicted. When the desired LoS cannot be verified by the delivered LoS of a *Line* or a *Stop*, the frame of the "defective" entity becomes red-colored, indicating that a problem occurred. In this case, the desired values of *LoSComfort* was set to "B". Due to the LoS requirement hierarchy, the *TrainLoSSpace* sub-requirement inherits the same LoS from its parent (composite *LoSComfort*). Moreover, *Line 1* entity has space LoS "C". As a result, the *Line* is annotated, notifying the engineer that its LoS does not substantiate the desired and thus, the entity cannot verify both the requirements. Additionally, *Omonia L1 Stop* is also annotated, since it does not verify the desired LoS of the *StopLoSSpace* requirement. On the contrary, *Thissio L1* and *Monastiraki L1* hold LoS better than or equal to the desired values (e.g., LoS "A" and "B", respectively) and they do not undergo any visual change.

When the entity is marked as "defective", the design environment facilitates the engineers to check additional information about the problem, making them aware of the degraded LoS of the entity and the specific requirement(s) that could not be verified. A pop-up window illustrates messages and further details about the current problem. As mentioned above, *Line 1* is annotated. Trains commuting on both its directions have LoS "C", while the *LoSComfort* requirement holds required/desired LoS "B". Thus, the message is the following: "Regarding Line 1 element, there is a problem with both Train directions; Current LoS (Left) = 'C'; Current LoS (Right) = 'C'; Required LoS = 'B'." The "Transportation LoS Plugin" provides automated recommendations for actions ("Suggested Solution(s)") to the RTS engineer for the improvement of the provided LoS. A suggested solution for LoS improvement is the following: "Passenger absorption from the passing *Trains* can be achieved by decreasing the duration between the *Train* departures and thus, increasing their flow. Here, the *Train* departure frequency can be set to "2" minutes."

As a practical example, the possibility of improving the LoS by increasing the *Train* frequency was explored. Changing the *Train* departure frequency to "2" minutes, we followed the described steps, i.e. execute simulation, convey simulation results to the RTS model, calculate and verify the LoS. As a result *Line 1* achieved LoS "B", equal to the desired LoS of both the *LoSComfort* and the *TrainLoSSpace* requirement. Thus, the *Line* is no longer annotated, as it can verify the LoS requirements. The *Omonia L1 Stop* was also able to reach the desired LoS.

Results

Using data provided by the Attiko Metro operation company (e.g., the departure frequency for *Line 1 Trains* is set to "6" minutes, etc), passenger comfort service levels were produced (e.g., Line 1 has LoS "C"). Exploring the possibility of improving the LoS, the RTS engineer can easily change the *Train* departure frequency of the *Lines* and compile the corresponding results.

Figure 4.10 shows the percentage of the *Train* commuting instances, at a certain service level (A to F), during the rush hour. In this case, the LoS values were obtained with departure frequency 1 train every 2 minutes are presented. We can observe that the majority of the *Trains* traversing *Lines 2* and *3* hold LoS "B", equal to the desired LoS. *Line 1*, a problematic entity with degraded LoS, now holds better LoS, e.g., "B" LoS percentage over 50%. This is an important improvement of the LoS, compared to the one presented in Figure 4.10.

Specifically, the improvement of LoS for *Line 1* can be observed, in the bar-chart (a) of Figure 4.11. Using a "6" minute time interval for the *Train* generation, the measured LoS was "C". On the contrary, changing the departure frequency to "2" minutes, LoS "B" was reached. As depicted in the diagram, the approach was also tested for "3" minutes. At that time interval *Line 1* remained "C"-level.

FIGURE 4.9: Indication of RTS elements, failing to meet required LoS.

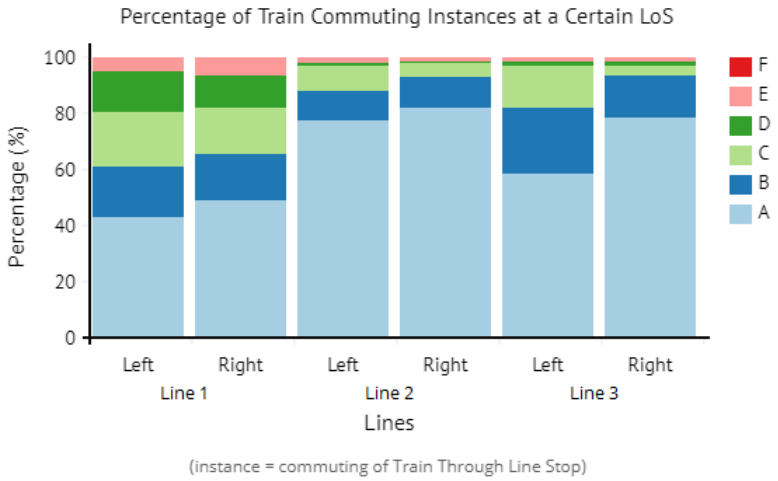


FIGURE 4.10: Percentage of train commuting instances at each LoS for all lines and directions. Departure frequency: "2" minutes for all lines.

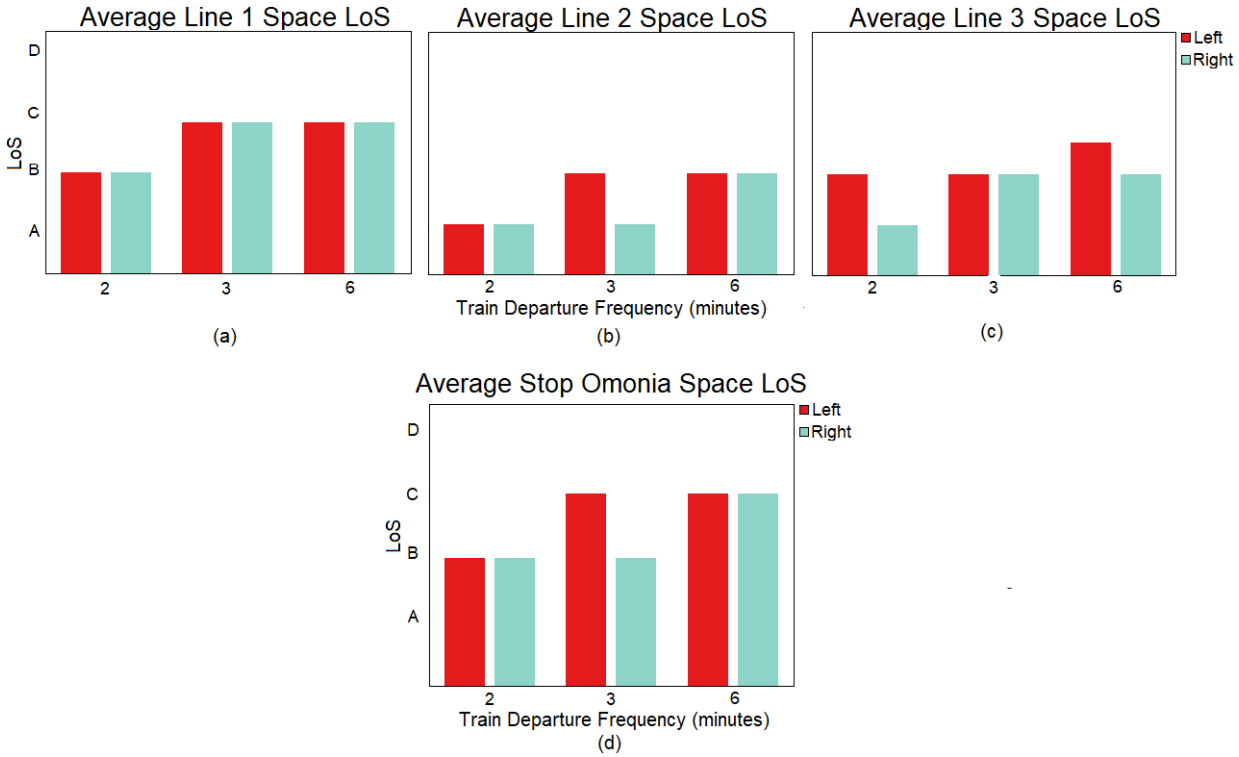


FIGURE 4.11: Average space LoS for a) line 1 trains, b) line 2 trains, c) line 3 trains, and d) Omonia line 1 stop. Train departure frequencies: "2", "3", and "6" minutes.

Similar observations can be made for the rest of the *Lines* in the same Figure. Diagram (b) shows the change from "B" to "A" in *Line 2* LoS, while diagram (c) illustrates the improvement of the average *Line 3* space LoS from "C" at the left platforms and "B" at the right platforms to "B" and "A", respectively. Another observation is that the LoS of *Omonia L1 Stop* was also improved. Diagram (d) indicates that, exploiting the proposed approach and adopting the aforementioned "3" and "2" minutes scenarios, LoS "C" switched to LoS "B".

The examples and results presented in this section provide a practical perspective of the proposed approach for Comfort LoS assessment by RTS engineers. Similarly to comfort, other well-known types of quality factors, such as delay and availability, as well as the inter-dependencies of RTS entities with respect to these factors on an end-to-end basis, could be supported by our approach.

4.5.4 Athens Metro cost computation

In this section, we focus on addressing a fundamental challenge in large-scale RTS, which is the need for adapting operations to satisfy commuters (e.g., in terms of available space within trains and stations), while minimizing the additional operating costs that this adaptation involves. The Athens Metro is also employed as case study. First, we describe the basic design problem of this case study, cost-wise. In addition, the Athens Metro model is described, based on previous sections, containing basic structural components, requirements and verification elements, as well as composite cost entities like the *OpExCost*. Second, we apply our developed Cost profile to the Athens Metro, facilitating the engineer to perform a cost analysis. We start with the generation of the cost entities connected to the structural system components. We then describe the computation of the values of the cost entities. In turn, the Athens Metro is evaluated cost-wise via the satisfaction (or not) of associated cost-related requirements. Finally, in Section 4.5.4, we discuss the application of our framework to the Athens Metro, by studying related results and benefits.

We demonstrate the design problem with the following scenario, based on the data of Section 4.5. During peak hours, i.e. 8-11 a.m. (business times), Athens Metro train routes take place every 7 minutes, yielding a LoS "D". This is a barely acceptable LoS (assuming "A" as the best and "F" as the worst). The operators want to improve this level to "C". To achieve this, a viable solution is an increase of the frequency of train routes from one train every "7" minutes to one train every "6" or "5" minutes. However, more routes mean more trains, and thus more OpEx. The Athens Metro operates under specific budgetary constraints; therefore, cost drives company-wide decisions. This is reflected with a specific threshold for acceptable OpEx beyond which increasing LoS is simply not viable. Thus, altering route frequencies requires two things: (i) a LoS analysis to ensure that the change achieves the required passenger comfort level, and (ii) a comprehensive cost analysis, in terms of incurred OpEx.

Having identified the two facets of the problem, an Athens Metro engineer comes into play. First, the engineer uses a developed RTS profile to apply MBSD on the Athens Metro system. Specifically, by employing SysML within a modeling environment, and running simulations, the engineer can identify the required system parameters and conditions under which the desired LoS can be achieved. This deals with the first facet and is already addressed in previous sections. With this at hand, the engineer employs our Cost profile and cost computation mechanisms, which facilitate the assessment of OpEx incurred during peak hours for different train route frequencies; this deals with the second facet, and is the primary focus of the following sections.

In Section 4.3.1, we introduced a domain-specific profile for the Athens Metro. This profile contains (i) LoS-related quantitative requirements that are satisfied by (ii) basic structural entities of a RTS, accompanied by (iii) simulation elements that facilitate the interaction with an external RTS simulator and enable the experimental calculation of LoS under different system parameters.

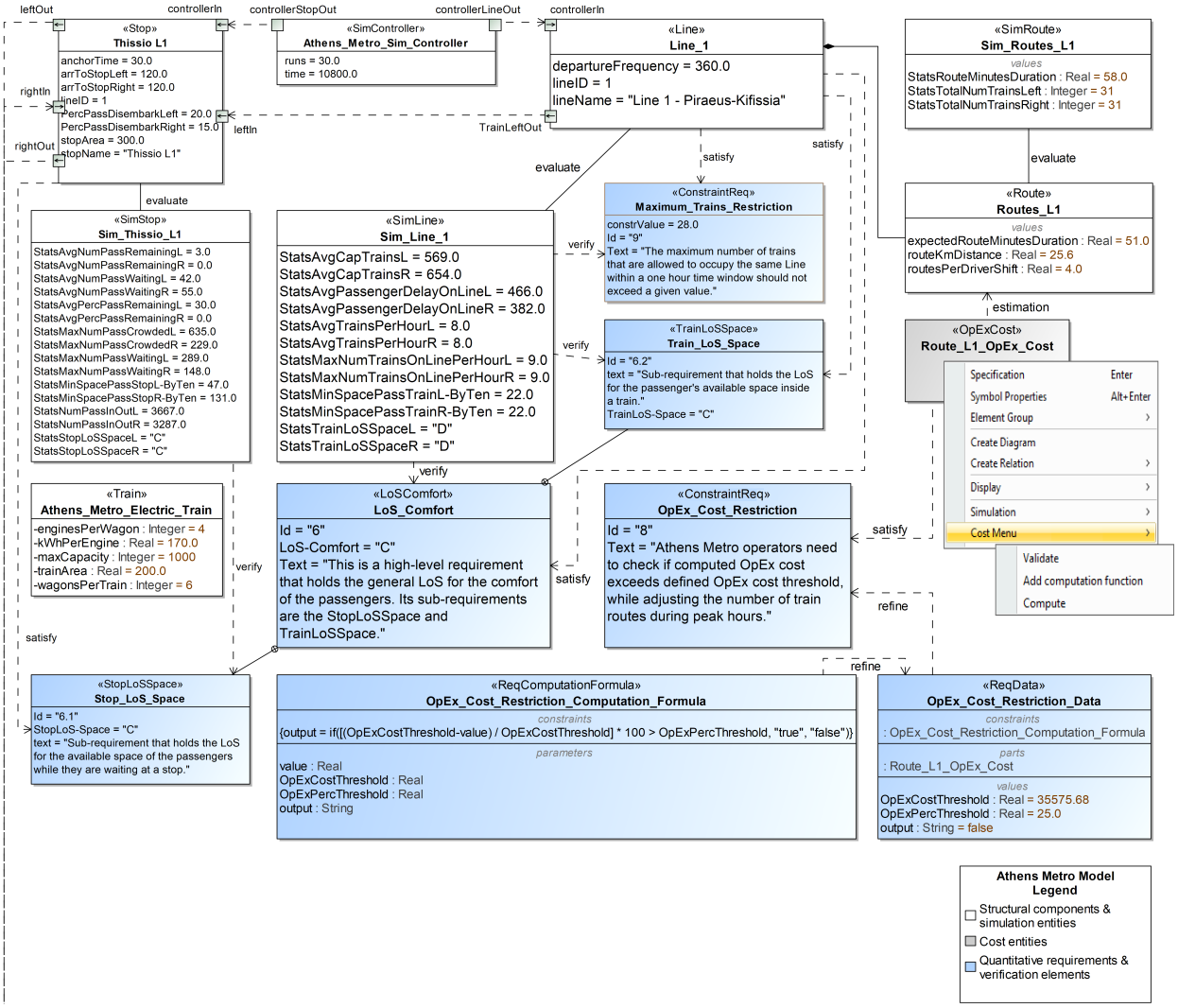


FIGURE 4.12: Athens Metro system model.

Specifically, the engineer can define the system structure, test different RTS configurations, simulate their behavior and extract statistical results (e.g., computed comfort LoS) that lead to the verification of LoS requirements and consequently, LoS evaluation.

In Figure 4.12, we present another excerpt of the Athens Metro system model where the aforementioned RTS profile was applied and LoS analysis was conducted. The depicted transparent (white) elements represent structural entities of the Athens Metro system; for demonstration purposes we consider one entity per type (e.g., line, stop).

Specifically, *Line_1* is a primary entity of this system, that contains the routes along which Athens Metro electric trains move. Since LoS needs to be evaluated on a line-by-line basis, the LoS requirement, referring to the general level of passenger comfort inside a train or at a stop, is connected to the Line_1 entity. Line_1 also comprises a sequence of stops. *Thissio_L1* is such a stop, corresponding to the platforms of the Athens Metro system and representing the area where passengers are accumulated in order to board an incoming train, or disembark.

The Thissio_L1 stop is connected to a LoS-comfort sub-requirement, holding the comfort level of passengers waiting at the platforms of the stop. The *Athens_Metro_Electric_Train* entity represents the trains that carry passengers, providing the transportation service(s). Specifically, after the passengers board a train (e.g., at the Thissio_L1 stop), the train transports them to the next adjacent stop, following a specific route (dictated by Line_1). The *Routes_L1* entity represents the course of the Athens Metro electric trains, traversing Line_1, from its starting stop to the final one.

Note that regarding cost, the Routes_L1 is the primary contributor of the Athens Metro OpEx; Athens Metro electric trains that traverse Line_1 routes incur energy consumption, maintenance, and driver crew costs, which compose the biggest part of the OpEx of the Athens Metro, induced during peak hours. Note that each system component holds its own structural properties; some of them may be descriptive (e.g., the ID of a line or the name of a stop), while others may be critical for conducting a cost assessment of the current system operation (e.g., the kilowatt-hours that an electric train consumes or the distance and expected duration of a route).

We further include a *Sim Controller* (i.e. simulation controller), used to control the system's simulation, in particular its time window (180 minutes, representing the 8-11 peak hours) and repetitions. In addition, simulation-related entities that evaluate respective system components are also depicted. The *Sim_Line_1* evaluates the defined Line_1, holding statistics about the number of the moving electric trains, their average capacity, as well as the minimum available space of the passengers inside the trains and average comfort LoS. The *Sim_Thissio* holds indicators related to average or maximum number of commuting passengers, as well as the minimum available space around them while waiting at the platforms of the stop. The *Sim_Routes_L1* simulation-related entity evaluates the Routes_L1, holding the statistical values of the performed train routes (e.g., the total number of routes, performed on the left direction or the exact duration of the routes in minutes). This entity's statistics are informative for the cost analysis that will follow; e.g., the simulated number of routes is factored in while computing the total OpEx.

With the Athens Metro structural model at hand, including the Routes_L1 structural element, the engineer focuses on the calculation of the OpEx that the routes induce. To this end, she inserts cost by defining a separate composite cost entity, and connects it to respective structural components. In Figure 4.12, the Routes_L1 entity is estimated by the *Routes_L1_OpEx_Cost* composite cost entity that represents the overall OpEx of the routes. Note that within our modeling environment the OpExCost entity is accompanied by a Cost menu; this menu contains the same buttons as in the REMS case study, enabling the engineer to follow specific steps to automatically generate cost entities, add computation functions, and automatically compute costs. These steps are described in the following Sections.

In pevious sections, different levels (LoS) of the comfort of the passengers were studied using the SysML requirement element. In particular, quantitative requirements regarding comfort

were defined, holding a property that described the desired levels. The Los_Comfort, depicted in Figure 4.12, is such a requirement that contains a specific id and self-description text, while the *LoS-Comfort* property represents this desired level. In our current work, we dive in the costs that an operational change that alters this property induces. The *OpEx_Cost_Restriction* requirement is defined, representing the need of the system operators to keep in check the relative differences in operational costs while adjusting the number of train routes during the Athens Metro peak hours. This is important since, assuming a congestion event at a platform (which degrades LoS), system operators may decide to increase the number of serving trains (and thus, routes) to bring LoS back to acceptable levels. However, more routes mean more OpEx (e.g., energy consumption, drivers, etc.). In order to check whether this requirement can be satisfied or not, this OpEx cost is compared against a Metro-operator-defined threshold. The relative difference in OpEx costs, upon adjusting the route frequencies, can be defined as follows: $\frac{OpExCost(routes_{final}) - OpExCost(routes_{init})}{OpExCost(routes_{init})}$
Thus, the requirement translates into the following inequality check:
$OpEx\_Cost\_Rel\_Diff(routes_{init}, routes_{final}) > OpExThreshold$,
where the threshold for the relative cost differences is provided by the Athens Metro operation company. This inequality is depicted in Figure 4.12. When the defined equation of the *OpEx_Cost_Restriction_Computation_Formula* is computed, it results in a "true"/"false" output value, indicating that the final operational cost value exceeds (or not) the defined threshold.

The *OpEx_Cost_Restriction_Data* is used to refine the costing requirement; it is primarily used as the holder of two values provided by the system operators, i.e. the existing (or initial) OpEx cost and the desired threshold. It also contains the computed result of the computation formula; this result indicates whether the related requirement is satisfied or not, as described in the following.

(i) Generate cost entities

The OpExCost entity, defined in the Athens Metro model (see Figure 4.13), is composed of the following individual cost entities: (i) the *Routes_L1_Energy_Cost*, which refers to the energy consumption cost incurred by the electric motors of the Athens Metro electric trains moving along the Line_1 route, (ii) the *Routes_L1_Driver_Cost* that is associated to the wage of the operating driver(s) (working in shifts), and (iii) the *Routes_L1_Maintenance_Cost*, which refers to the expenses incurred by maintaining (e.g., repairing if needed) the traversing Line_1 electric trains. Note that each cost entity holds cost-specific properties, critical for their cost value computation (e.g. the wages of a train driver is reflected in the *driverSalaryTotalCost* property of the Routes_L1_Driver_Cost entity).

(ii) Add cost computation functions

Following the definition of the cost entities, the engineer can use functions, that are readily available in the cost inventory (see Figure 4.14), to compute energy consumption, labor and maintenance costs for the electrically-powered trains that move along the Athens Metro routes. In Figure 4.13 we illustrate these functions within the Athens Metro model.

The energy cost value of the electric trains traversing the route is the result of the multiplication of the (i) number of wagons per electric train, (ii) the number of engines per wagon, (iii) the kilowatt-hours that electric train engines consume (per hour), (iv) the duration (in hours) of a complete route, (v) the cost of each consumed kilowatt-hour, and (vi) the total number of performed routes (the summation of the number of routes, performed in both directions). Regarding the cost value of a train driver, it is the result of the multiplication of (i) the total number of routes divided by the number of routes that a driver performs on her shift, and (ii) the total salary cost of the driver. Finally, the maintenance cost value for electric trains is the result of the summation of (i) the maintenance crew cost per hour, multiplied

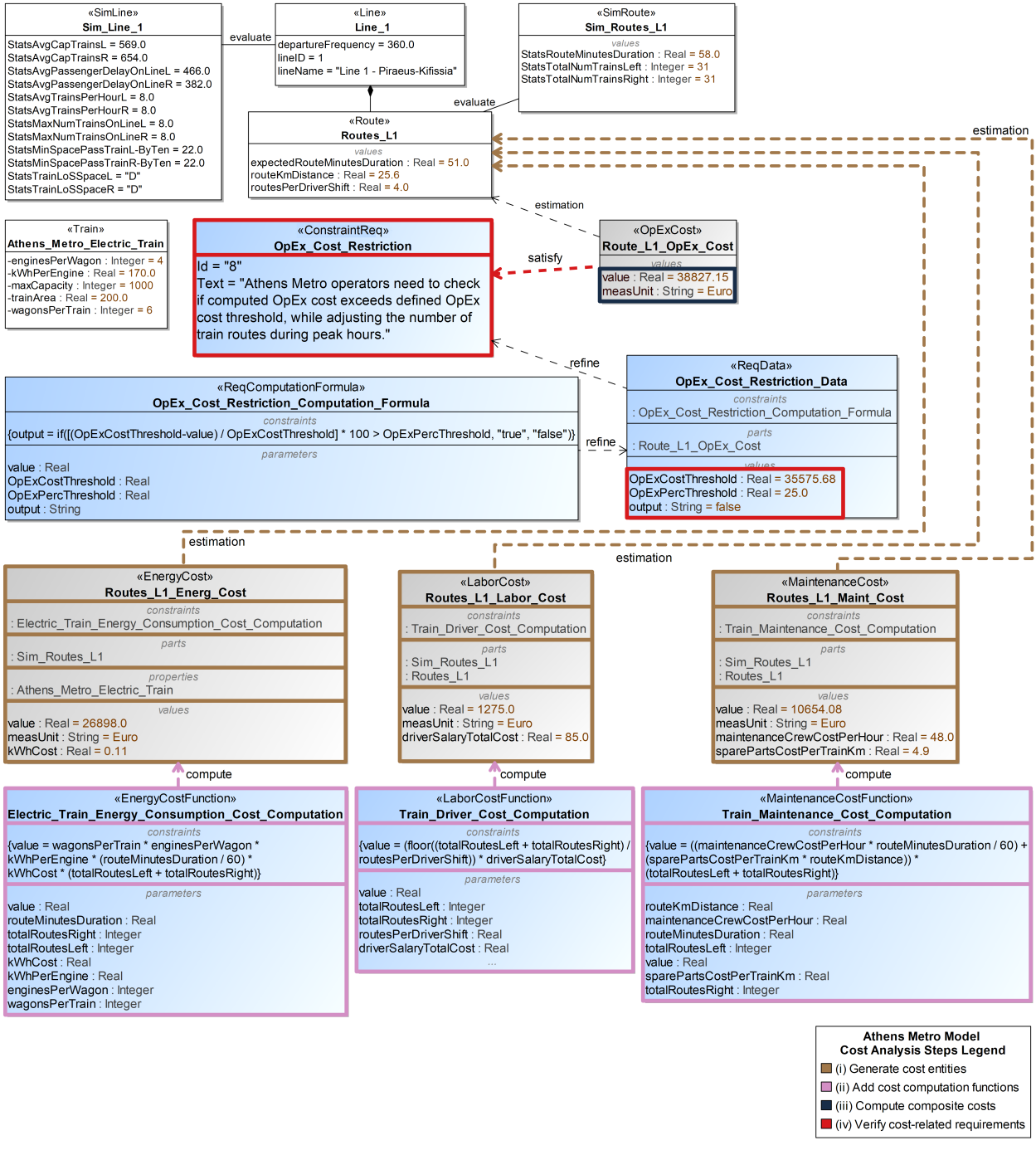


FIGURE 4.13: OpEx cost computation in the Athens Metro system model.

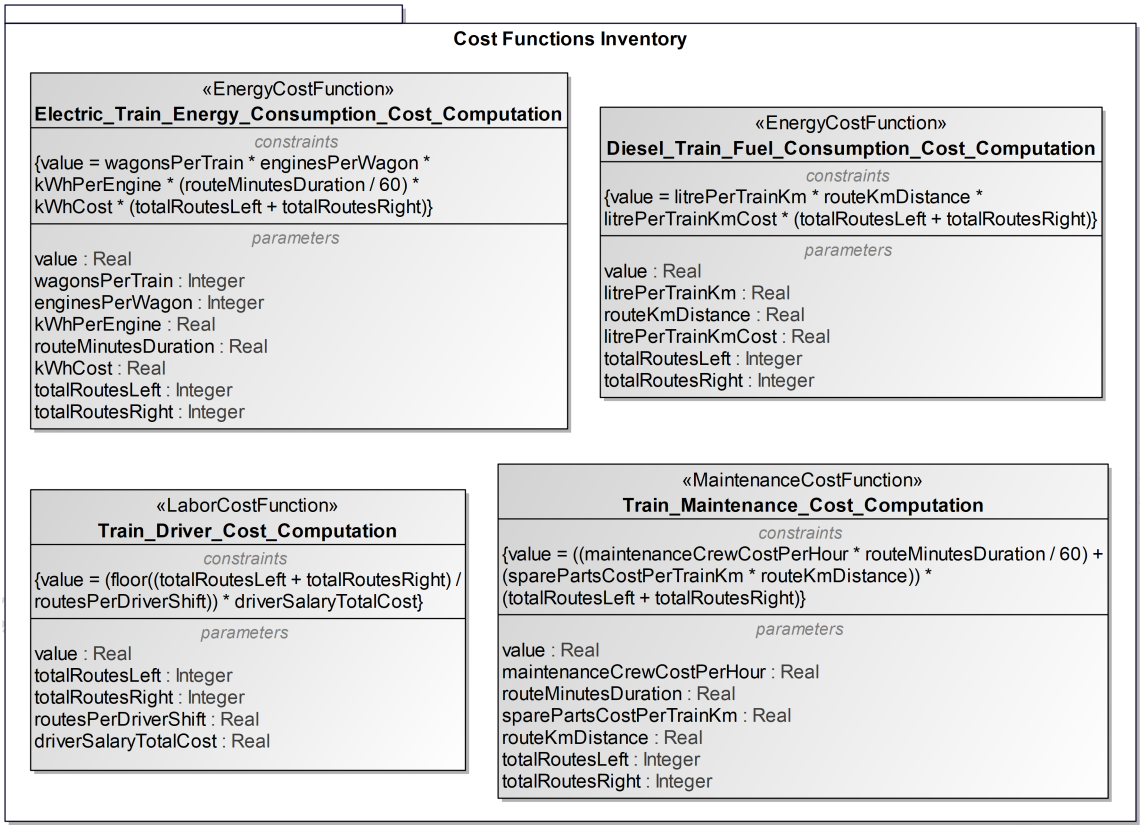


FIGURE 4.14: Cost functions inventory.

with (ii) the duration of a complete route (in hours), and (iii) electric train spare parts costs per train-Km, multiplied with (iv) the route's distance (in km); this summation is multiplied with (v) the total number of performed routes. Note that these functions are contained in the Cost profile functions inventory depicted in Figure 4.14.

In order to compute each cost value, parametric diagrams provided by the SysML, are employed; Figure 4.15 depicts such a parametric diagram holding the equation for the computation of the energy cost value of an electric train.

The equation takes input properties whose values are used to compute it; these properties may belong to different entities. In particular, the energy consumption cost function takes input from (i) the Athens_Metro_Electric_Train entity, (ii) the Sim_Routes_L1 entity, and (iii) the Routes_L1_Energy_Cost entity. The Athens_Metro_Electric_Train provides the numbers of the train wagons, their electrical engines, and the energy consumption of these engines in kilowatt-hours. The Sim_Routes_L1 entity supplies the duration of a complete route and the total numbers of routes, performed in both directions, during peak hours. Finally, the Routes_L1_Energy_Cost entity provides the cost of a kilowatt-hour; note that this is a cost-specific input property. The equation is solved within the parametric diagram (using OpenModelica as the math solver) and the output energy cost value is extracted for further cost assessment. Here, the *value* property represents the extracted output value of this equation. Using a parametric diagram within the modeling environment, the engineer does not have to go through the complexity of writing additional code for the computations. She simply provides the corresponding input values and defines the functions that will be used; after the computations are complete, the final required cost values are easily extracted and may

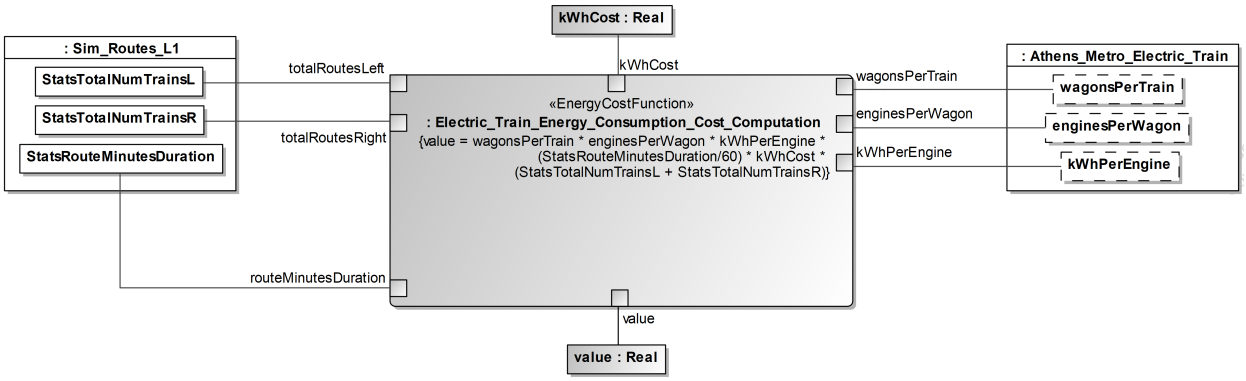


FIGURE 4.15: Electric trains energy consumption computation.

be used for the cost analysis of the system.

(iii) Compute composite costs

Since the aforementioned functions refine a respective cost entity, the computed output value is stored within these cost entities. An OpExCost entity obtains its value from the sum of the values of EnergyCost, LaborCost and MaintenanceCost entities via automatic summations. Therefore, at this stage of the analysis, the engineer has all the components and the method needed to automatically compute the OpExCost value. Indicatively, in Figure 4.13, the Routes_L1_Opex_Cost obtains a "39387.35" Euro cost value (assuming a "6"-minute train inter-arrival interval); this is the result of the summation of the Routes_L1_Energy_Cost, Routes_L1_Driver_Cost and Routes_L1_Maintenance_Cost entities' values during the defined peak hours, when the Athens Metro was studied.

(iv) Verify costing requirements

In the aforementioned LoS analysis, LoS and LoS-specific requirements were used to evaluate the system; desired LoS was set at a "C" level, while the simulation (assuming a "6"-minute route frequency) returned LoS "D". Thus, the aforementioned LoS_Comfort requirement was not satisfied by the Line_1 entity. Here, the same principles are followed for the evaluation of the Athens Metro from a cost perspective. The Opex_Cost_Restriction requirement is either satisfied or not, leading to the system's cost evaluation. The computation formula is computed within a parametric diagram, where input/output properties are used; specifically, (i) the cost value, provided by the Routes_L1_Opex_Cost entity, (ii) the initial OpEx cost value, provided by the requirement data entity, and (iii) the pre-defined percentage OpEx threshold, held within the same entity, are used as input. When the formula is computed, i.e. the computed Routes_L1_OpEx_Cost value is compared against the threshold and a specific result is extracted, i.e. "true" or "false", leading to the satisfaction of the requirement. The problematic –non-verified– requirement is annotated with a red-colored frame; this way, the engineer is informed about the defective element(s). In our case, the extracted output value returned "false", meaning that the Routes_L1_OpEx_Cost value does not surpass the threshold, thus, the related requirement (OpEx_Cost_Restriction) is satisfied.

Results

Table 4.3 contains alternative Metro design solutions w.r.t. passenger comfort LoS and operational cost increase, corresponding to different train frequencies for Line 1. In particular, system engineers run simulations to calculate comfort LoS for "7", "6", "5", and "3"-minutes intervals

TABLE 4.3: Evaluating the trade-off between LoS and cost in RTS.

Athens Metro Line 1		Routes (L + R)	Peak hours OpEx (E)	Peak hours OpEx increase (%)	Daily OpEx (E)	Daily OpEx increase (%)	Avg LoS	Accepted solution
Line 1 trains frequency (minutes)	7	56	35575.68	–	160494.03	–	D	✗
	6	62	38827.15	9.1	164213.1	9.14	D	✗
	5	78	49551.84	39.2	174243.98	15.8	C	✓
	3	128	81315.8	128.5	205632.99	36.7	B	✗

between arriving trains of Line 1, during rush hour. All of these alternatives provide valid system (i.e. appropriate) outputs, according to Wymore [293]. However, engineers are interested only in the acceptable ones, i.e. those that satisfy both comfort LoS and cost restrictions. That is, comfort LoS should be set to level "C" for Line_1 with an increase of the operational cost up to 40% during peak hours (see Figure 4.12).

Having yielded suitable solutions for the Metro peak business hours (8-11 a.m.), operational cost data may also be extrapolated on a daily level. Metro operators have a dual OpEx requirement, concerning the peak hours (i.e. 40% is the maximum for relative OpEx increase) and the daily operation (i.e. 25% is the maximum for relative OpEx increase). Thus, daily OpEx increase is also depicted in Table 4.3.

In the context of RTS, the primary design goal is to improve comfort LoS for passengers, while the cost restriction is a maximum relative OpEx increase over which this improvement is not financially viable. The acceptability of a design solution on both fronts is explicitly stated in the last column of Table 4.3 (i.e. "Accepted solution" column).

We observe that both the "7"- and "6"-minutes frequencies satisfy the cost constraints, although LoS remains unacceptable ("D"). The rest of the configurations (i.e. those under "6" minutes) achieve better LoS ("C" or even "B"). In particular, the "5"-minutes configuration is the optimal solution that balances acceptable LoS and cost; note that cost is acceptable for both peak and daily hours. Time intervals under "5" minutes (i.e. "3" minutes) lead LoS to an even better level (i.e. "B"), however the relative OpEx cost increase it demands violates the constraints set for both the peak hours and the daily operation.

In this case, our approach assisted the Metro Operator to easily identify an acceptable solution with the proper balance between performance and cost. This assessment takes place in a single graphical modeling environment, where both performance and cost estimations are integrated within the SysML model and presented to the user. Quantifying the trade-offs is key.

It is certainly not a trivial task to increase patient comfort LoS in a large-scale RTS. It costs money and it is often not a first-level priority for the operators. In the case of Athens metro, our exploration was successful since we reached an acceptable LoS configuration, within the cost limitations that the operator had set. Revenue and profit exploration was not part of the study discussed with the operator.

4.6 Discussion

Using the Athens Metro RTS as a case study, we demonstrated that our framework can be applied in a real-world system and that it yields useful results for improving aspects of the system. In fact, it assisted Athens Metro operators to easily assess and identify an acceptable solution (i.e., satisfying certain constraints) that strikes the proper balance between performance and cost, with the

passengers' comfort LoS being the key performance indicator and operational cost (OpEx) being the primary cost driver. This assessment takes place in a single graphical modeling environment, where both performance and cost estimations are integrated into the SysML model and presented to the operator/user; this helps in the decision-making process.

Specifically, we apply our framework to RTS, serving (a) LoS parameters definition, based on system requirements, (b) designation of the required LoS during system's operation, and (c) estimation of the expected LoS, and verification against the required LoS, using system model's details and simulation results. This is achieved with a general framework for requirement specification, hiding detailed, specific, low-level requirements. In this way, the approach promotes the process of collaborative system development (and co-design), beyond the limited boundaries of a system engineering team. The engineer is facilitated to define LoS requirements and relate them to specific components, e.g. stations or routes, in different levels of detail. This enables the automatic simulation of the RTS operation and identifies the components where the desired LoS are not achieved. The latter triggers an iterative process, since service operators may classify the system under study and further investigate ways to improve LoS. To this end, RTS-related LoS concepts are integrated into the corresponding SysML model, while automated estimation of LoS is provided, via simulation as an intermediate step. Simulation is performed using DEVS.

In addition to performance, cost is a crucial system design parameter. In this work, we provide the designer with the capability of analyzing cost aspects of a system using SysML. In particular, the proposed framework facilitates system operators to explore the most cost-efficient operational solution dealing with the volume of the served commuter populations. These mechanisms assist the designer to perform a cost analysis of systems dynamically, at different levels of granularity, across domains.

The exhaustive coverage of all end-users and stakeholders considerations, is out of the scope of this work, since it is not a comprehensive analysis of all the QoS metrics (quality, etc), associated costs (price, value etc) or the operations of a whole RTS. Nevertheless, we envision that, following our framework, further generalizing the domain of application with the introduction of additional concepts and activities related to QoS and LoS in SysML could be pursued. Hence, the framework, currently applied only on RTS, could be extended to include different types of public transportation systems, e.g. buses and trams, allowing the study and analysis of the quality of public transportation services from a wider perspective.

To the best of our knowledge, we are the first to implement the concept of LoS in MBSD, leading to specific benefits: integrated definition of LoS in a specific domain; high-level, yet specific and usable indications of expected quality, as perceived by end users, during operation; automation in diverse system model utilization with simulation and LoS calculation; traceability, leading to automated identification of specific model elements that degrade the overall LoS; standards-based integration of different software tools; encapsulation of complexity under the hood and offering of usable abstractions to manage and study LoS.

Chapter 5

Applying proposed framework in Cyber-Physical Human Systems

> "User-centered design means understanding what your users need, how they think, and how they behave — and incorporating that understanding into every aspect of your process."
>
> Jesse James Garrett

In this chapter, we present the application of the proposed MBSD framework (described in Section 3) in the CPHS domain to integrate the concerns of human participants into the design of such systems.

In Section 5.1, we first describe the problem in CPHS design. To tackle this problem, we focus on the engagement of human participants in design by transforming their human concerns into critical system design requirements called criticalities (see Section 5.1.1). We also discuss the need to perform cost analysis in CPHS by balancing different service and budget constraints imposed by the human participants themselves (see Section 5.1.2). Furthermore, we focus on providing additional design capabilities to a CPHS designer by integrating decision support capabilities into CPHS models to explore alternative system solutions that meet the human concerns (see Section 5.1.3).

Next, in Section 5.2 we apply MBSD to CPHS, and in Section 5.2.2 we introduce criticalities in CPHS design. In particular, we analyze three distinct views, i.e. human, system, and criticalities, to model human behavior, system specification, and critical aspects of the system, respectively, and we describe the design process that traverses these views.

In Section 5.3, we apply our QoS meta-model (presented in Section 3.4.1) to CPHS. Based on this meta-model, we create a CPHS SysML profile to support the human-centric perspective in CPHS design.

In Section 5.4, we also apply our cost meta-model (introduced in Section 3.5.1) to CPHS to enable their design and evaluation from an acquisition cost perspective.

As a case study, in Section 5.5, we present the design of a REMS considering the service and budget constraints of patients in terms of the equipment installed in their homes, as well as the decisions that a REMS designer can make in configuring a REMS that meets the needs of the patients (see Sections 5.5.1-5.5.5). We also present the results extracted from the analysis of the REMS case study (see Sections 5.5.6-5.5.8).

Finally, in Section 5.6, we discuss observations emerged from this analysis and we comment on the way the proposed framework helped to address the REMS design problem.

5.1 Problem statement: Integrating human concerns in CPHS design

CPHS identify the necessity of human participants and their special role in the operation of these systems. Therefore, CPHS development approaches address the important aspects of human interference and, of course, safety [86]. In parallel, the huge and increasing number of small communication devices –including sensor components– connected to the Internet has led to a growth of applications based on interconnected peers interacting with the environment, i.e. IoT [45]. Nevertheless, these devices are usually owned and used by humans and operate for their benefit.

These parallel domains –CPHS and IoT– appear to be converging, as identified in [35], leading to numerous and increasingly important applications that require human involvement/participation while impacting human life. Medical applications are a domain where this evolution can be decisive for the effectiveness of monitoring and remediation activities.

However, human participants cannot be addressed as simple system components. System components are designed to meet specific functional requirements and provide a system solution. Such components are selected or built to meet the functional and physical requirements and to be integrated into the developed product. This practice cannot be applied to human participants who exhibit different behaviors and usually set their own requirements. Not to mention the fact that they are a crucial factor for the acceptance and success of the system.

These concerns have been identified, and the role of the human participant has been studied [120, 301, 206, 182, 208, 240, 46, 263, 78, 255, 86]. However, human participants have been considered as objects of study, overlooking their potential role as active participants in the design process.

Although CPHS can drastically affect the lives of human users (see the healthcare domain), their impact on users is usually only taken into account at the end of the design process (if at all) [208]. Moreover, even at this stage they are usually studied in isolation from other cyber or physical elements. Keeping the human factor out of the loop during the system design can lead to poor quality and performance of the provided services, as perceived by the user, and high costs for the operation of the system [25, 28].

In addition to integrating the human factor into system design, dealing with CPHS complexity is another key challenge in designing high-quality systems. This is exactly where model-based design comes into play. However, previous approaches focus on isolated stages of the design process (e.g., simulation or verification) without employing a well-defined, integrated framework that guides the steps from concept to implementation [118]. Such a framework can be implemented with SysML.

Despite the expressiveness of SysML, it does not inherently support the incorporation of human aspects and considerations into system design. In particular, there are no standardized elements that can cover the human perspective such as the perceived quality of a service provided by the designed system [72]. Only a few approaches attempt to use it to design CPHS considering the human participant [130]. In these cases, the latter is usually symbolized as an external actor connected to system components [285]; however, this notation is not representative of the integral role of humans *within and as part of the system*.

5.1.1 Transforming human concerns into design requirements

The driving forces in this work are the human participants, their concerns, and the activities they perform. To this end, the human participants of the CPHS will be involved in the early stages of the system development process, i.e. during the system design activity. This involvement is expected to promote the identification of human concerns, while the model-based nature of the approach facilitates the definition of human concerns, their correlation to system components, and

the designation of critical aspects of the system. This framework facilitates human participants to dive into system details via specific views and modeling constructs, depicting key performance indicators to help them understand what is feasible and how the system can be improved. Well-known facilities of model-based development [42, 113, 127, 145], such as system model validation, automated requirements verification, and traceability are also used. Overall, the approach aims to understand the human participant's requirements and satisfy them with an appropriate system design, through a process that continuously develops and maintains the bond between the human participant and the system. Therefore, it is expected that humans will develop deep awareness of the CPHS and their willingness and motivation to participate will increase when it is rolled out.

In [146], the concept of criticality was introduced to depict human concerns as critical design requirements and restrict system functionality. The authors extracted the primary concerns that human users have when operating medical CPHS and transformed them into criticalities, classifying them in primary criticality categories. Then, existing medical CPHS were evaluated based on their accommodation of various concerns and criticalities. In this work, we further explore the transformation of human concerns into system requirements by greatly extending and enriching the concept of criticality introduced in [146]. For this purpose, our QoS profile comes into play and is used in the CPHS domain as described in the following sections.

5.1.2 Cost analysis in CPHS

In the context of CPHS, provided personalized services are crucial for human users. However, dealing with the capital and/or operational expenses of a "perfect" system may be prohibitive. Moreover, users' needs may vary, resulting in different configurations of the system corresponding to different acquisition budgets. A CPHS configuration is considered suitable only if it meets the user's performance or other requirements. However, users may be tempted to purchase more expensive solutions than they really need. The designer should therefore help users choose a suitable CPHS configuration, taking also into account their budgetary constraints. Note though that finding the right balance between cost and performance is not trivial, even in simple scenarios. Therefore, proper cost analysis and evaluation is necessary during system design.

5.1.3 Decision-making in CPHS

In the context of CPHS, we focus on integrating decision-making capabilities into the design of such systems to enable system designers to explore alternative solutions that meet the needs and criticalities imposed by human users. We follow the decision-making process from Section 3 where the system model, in this case the CPHS model, is transformed into a decision model whose results are automatically incorporated back into the system model. This process runs iteratively on any typical SysML model and it is therefore appropriate for CPHS. It facilitates the designer's exploration of alternative design solutions and minimizes the manual effort required to achieve them.

5.2 Applying MBSD in CPHS

5.2.1 Basic concepts

In order to apply MBSD to the CPHS domain and build domain-specific models, SysML is adapted to accommodate domain-specific aspects; this is done via the profiling mechanism. In particular, based on our proposed framework (described in Section 3), we construct a CPHS SysML profile that includes QoS entities, cost entities, and structural entities, all related to the domain under

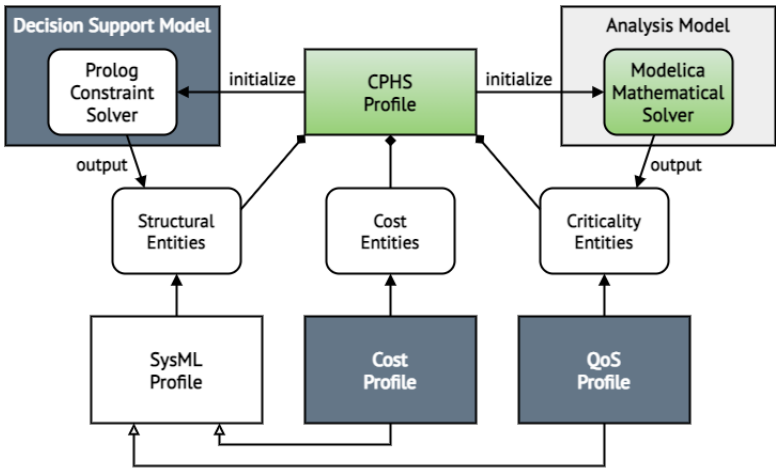


FIGURE 5.1: Applying profiles to CPHS.

study. The way the CPHS profile was built to facilitate MBSD in the CPHS domain is shown in Figure 5.1. First, we applied the general SysML profile (white-colored) to CPHS, which contains basic stereotypes such as blocks to describe structural CPHS entities (such as devices, actuators, etc.). Second, we applied our QoS profile (grey-colored), which contains QoS-related concepts such as graded quantitative requirements to define and manage criticalities. Third, our cost profile (grey-colored) is applied to the CPHS profile to define and manage cost entities, which in turn should satisfy cost criticalities and cost-related human concerns. In Section 3.1.1, we pointed out that a CPHS can be analyzed using customized functions or equations that encode the specific needs of the human user(s). Therefore, an analysis model in the form of a simple mathematical solver (green-colored) is sufficient to compute criticality verification formulas and verify criticalities; note that an OpenModelica solver is used (see Section 3.2.2). Finally, to support the designer's decision-making process, a logical solver (white-colored) is used to evaluate different system design solutions that satisfy the criticalities.

5.2.2 Introducing criticalities in CPHS design

We adopt a human-centric perspective in CPHS design. Therefore, the basic concepts involve human participants, their intended activities and their concerns. The system is designed to provide operations to accommodate human activities and deal with the concerns. In this context and following the principle of the separation of concerns, the human view and the system view are proposed to model human behavior and critical aspects on one hand and system specification on the other hand. Modeling elements of the two views are connected when required; the critical aspects of a CPHS as a whole are examined in a separate, criticalities view. The proposed CPHS design process is defined in terms of a BPMN diagram and explained, while the required extensions in SysML are defined in the CPHS profile.

Basic concepts

[255] depicts the human participant or person as the primary stakeholder (core component) of a CPHS, that is placed at the center of a Human Service Capability Description (HSCD) model; this participant is characterized by specific capabilities. These capabilities facilitate a CPHS designer to understand the tasks or activities that people perform within the scope of the CPHS and how they interact with other people, components, and the system itself. The generic capability of the

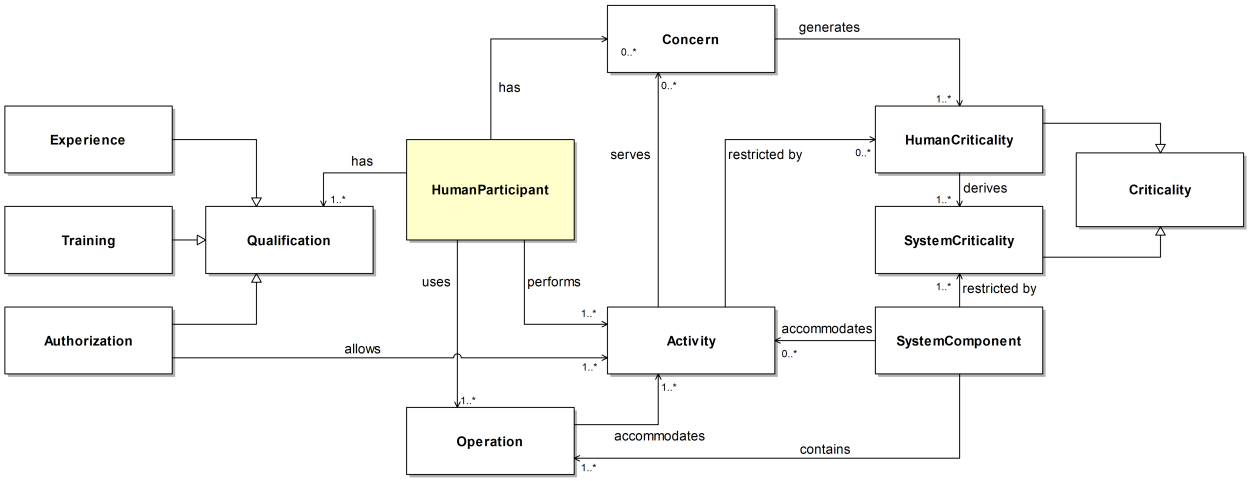


FIGURE 5.2: Basic concepts of CPHS.

human participant is her qualification which describes the aptitude of the participant. It comprises three additional capabilities, i.e. experience, training and authorization. The first one refers to the knowledge or even previous experience that the participants may have, regarding the system and associated activities. Training refers to how accustomed the participants are to the activities they perform. Authorization refers to what activities participants are allowed to perform. The combinations of these concepts categorizes participants. This categorization facilitates the system designer to effectively identify and integrate participants into the design process.

However, although in approaches like [255] the participant is treated as the primary component of a CPHS, they do not focus on describing and integrating the concerns that the participant may have. Addressing these concerns is crucial for the integration of any CPHS into the lives of human participants, since it enables the designer to develop a personalized system that covers their needs. This in turn affects their willingness to actively participate in the system. In our work, we use and adapt basic human concepts of the HSCD model (presented in [255]), such as qualification, experience, training, authorization, and operation (see next paragraph), and propose necessary extensions to it, as shown in Figure 5.2.

Primarily, we have added the concept of the concern; human participants may have certain concerns regarding the operation of the system and its effects on activities they perform. A human concern may correspond to a certain system component with specific properties. These properties correspond to additional constraints and requirements that should be enforced on system design and affect functionality and operation. However, there is a logical leap between the high-level concerns of the participant (who is typically not a system design expert) and the system-level requirements that need to be met (via an expert designer) so that the system would be functional. Hence, an open issue is to manage how human concerns are translated into requirements, in a way that is (i) useful to both the system designer and the human participant, (ii) clearly defined in the design model, and (iii) effectively verifiable. This translation should eventually align the view of the human participant with the view of the system. To achieve this, we employ the concept of criticalities that are generated by human concerns, and correspond directly to critical system design requirements [200]. Specifically, the criticality is a concept semantically correlated with the severity of the effect to the human participant, should a component or an operation, provided by the system, fail or suffer performance degradation [31]. We have identified two specializations of the criticality, namely human criticality and system criticality. Human criticalities are generated directly by the concerns of human participants and restrict the activities that the participants

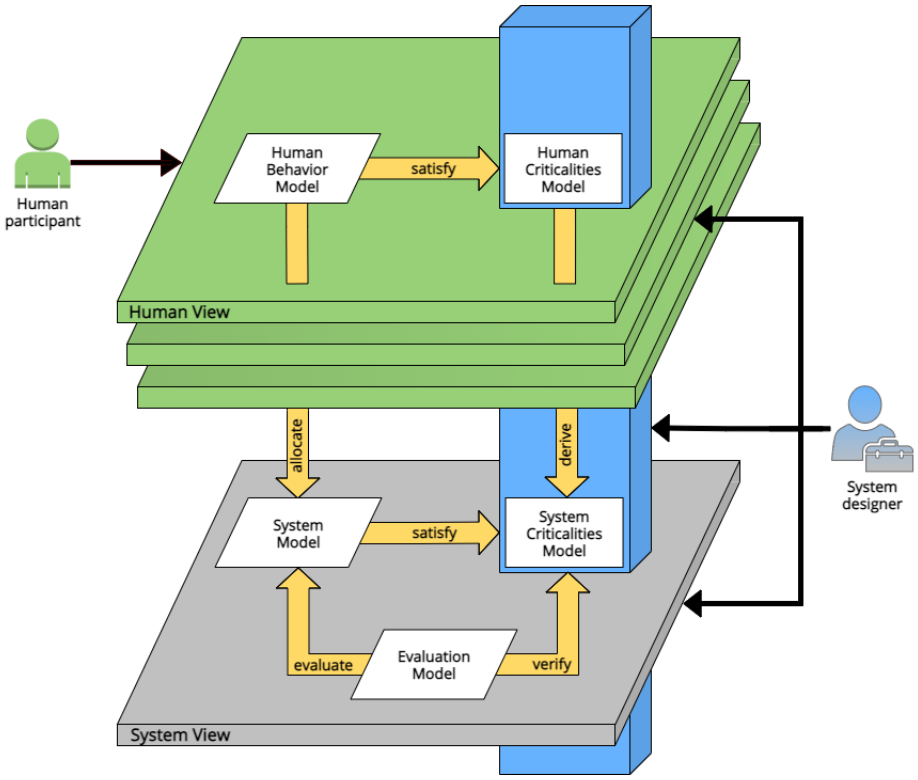


FIGURE 5.3: Design model of CPHS.

perform. System criticalities are derived from human ones and restrict components of the system that in turn accommodate these activities; system components contain specific operations that also accommodate human activities.

Design model

In our proposed approach, three key views are defined for the design of CPHS: the human view, the system view and the criticalities view. These views are represented by corresponding models [72], that comprise the overall model of a CPHS, integrating the full set of information of the system design; all views/models are depicted in Figure 5.3 and are analyzed in the following.

Human View (green-colored). We acknowledge that a CPHS is governed by the behavior of its human users [255] and we enable their active participation not only in the operation of the system but also in its design. The basic premise is that a human user that takes part in the CPHS design is also willing to actively use and operate the system. To represent the view of the human CPHS participant, we introduce a *Human Behavior Model*, as depicted in Figure 5.3. This model contains entities related to the behavior of the human participant, like activities she performs within the context of the CPHS.

Criticalities View (blue-colored). Human participants typically have concerns about the system they are wiling to use. These concerns correspond to critical design requirements or criticalities (as described in previous Section 5.2.2). Specifically, concerns generate human criticalities. Both concerns and criticalities are contained within a *Human Criticalities Model*; note that the activities of the human participant, contained within the behavior model, should satisfy the human criticalities

model. Regarding the latter model, system criticalities are derived from human ones and are contained within the *System Criticalities Model*. Both criticalities models comprise the Criticalities View.

System View (grey-colored). The human behavior model also affects the *System Model*, which represents the actual system. This is because human participants perform activities that employ system components which comprise the system structure; this structure is defined by the designer and is contained within the system model. The system model needs in turn to satisfy system criticalities. In addition, an *Evaluation Model* completes the system view since it can be used to verify system criticalities and evaluate the system model. If the evaluation succeeds, this means that the system view fits to the view of the human participant; if not, the system designer should consult with the human participant and revisit either the system structure and operation and/or the criticalities imposed on them, until a satisfactory solution is reached.

Design process

Based on the defined views (analyzed in Section 5.2.2), two critical roles are identified; system designer and human participant. The role of the system designer –as an expert– is to define and model all entities that comprise the CPHS, using a single design/modeling environment, and traversing all views. The role of the human CPHS participant is to express needs/concerns and facilitate the designer to achieve the most suitable CPHS solution for the participant. Note that the human participant does not dive into the technical details of the process but rather "drives" its application by supplying appropriate feedback (e.g., concerns) to the system designer. The cooperation between these two roles is an integral aspect of system design and is made explicit in the following, where it applies.

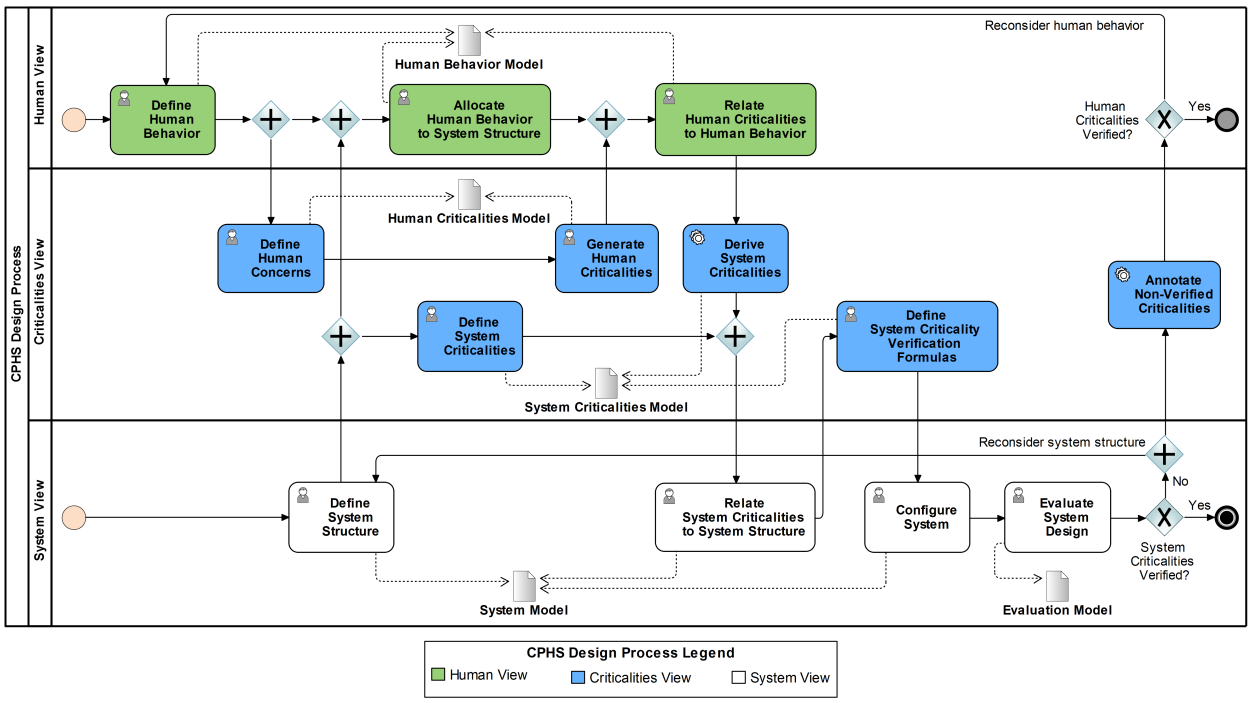


FIGURE 5.4: Design process of CPHS.

The CPHS design process is presented in Figure 5.4 and is divided into three lanes. Each lane corresponds to a defined view and contains design actions; some actions are executed by the designer –in coordination with the human participant–, while others are executed in an automated fashion within the design environment.

The process is initiated in the human view (upper lane in the Figure 5.4) by the designer who *defines human behavior*, i.e. the activities that CPHS participants perform within the constraints of the system. In turn, the designer communicates these activities to the human participants in a simplified manner to achieve mutual understanding of the system.

Having communicated the involved activities, the CPHS designer *defines system structure*, i.e. basic system components that comprise the CPHS structure. This action is executed within the system view (bottom lane in Figure 5.4).

Having this structure at hand, the designer can *allocate human behavior to system structure*. This way, the CPHS components support the execution of the performed human activities.

Following the human behavior definition, the designer motivates human participants to express concerns/needs and conditions regarding CPHS usage and operation. The participants verbally communicate such concerns to the designer. In turn, the system designer *defines human concerns*, inserting them in a raw form (textually) within the design environment. Concerns are defined in the criticalities view (middle lane in Figure 5.4) in a custom descriptive way. The SysML language is used, offering the Requirements diagram, suitable for depicting these concerns.

At this point, concerns are expressed in a simplified manner but are not yet of direct use to the designer or the human participant. To be useful, they need to be graded and quantified, becoming suitable to system design. To do that, concerns *generate human criticalities*. This generation is executed within the criticalities view by the designer who describes specific criticalities (in cooperation with the participants). It is worth mentioning that each criticality is characterized by a "levels" quantitative property whose value indicates the desired level that covers the needs of the participants; the –desired– criticality levels for human criticalities are solely provided by the participants.

Following the generation of human criticalities, the designer *relates human criticalities to human behavior*. Specifically, each defined activity should satisfy corresponding human criticalities; thus, corresponding "satisfy" relationships between performed human activities and criticalities are set by the designer. Note that human concerns are general and refer to the CPHS as a whole; activities are related to human criticalities which are more specific and restrict functionality. Therefore, activities satisfy specific human criticalities but are not directly connected to high-level free-form concerns.

The human criticalities, defined by the designer in cooperation with human participants, are still a product of the human view and criticalities view, but are not yet directly "tied" to the actual system structure. To achieve the latter, human criticalities should be aligned with the system view. To this end, they are translated into corresponding system criticalities that directly restrict system structure. Specifically, the designer *derives system criticalities* and *relates system criticalities to system structure*. Specifically, the designer initiates an automated process that (i) creates system criticalities within the system model and (ii) forms connections between criticalities and system components, given that the aforementioned allocations between human activities and system components have been performed. This process is analyzed in the following Section 5.3. Note that the designer can manually *define system criticalities* that are not necessarily derived by or connected to corresponding human criticalities. These separate system criticalities can enrich the system model and cover various system design requirements.

Having system criticalities at hand, the designer *defines verification formulas* for each criticality within the criticalities view. These formulas hold simple if-then-else expressions or mathematical equations, used to verify system criticalities. Specifically, each formula facilitates the evaluation

of the computed output, i.e. the real criticality level, against the one desired by the participants. While the insertion of the formula(s) to be used is done by the designer, computations are performed automatically by the modeling environment (see Section 5.3).

Within the system view, the designer is able to perform two actions that conclude the view: (i) *configure the system*, and (ii) *evaluate system design*. The first action, i.e. configuring the CPHS, refers to the population of the properties of the system components with specific values. This way, the components become instances of the system. Values can be inserted by the designer and can be used to compute formulas. The second action is to evaluate the CPHS, based on the aforementioned formula computations. Note that the proposed design process can be repeated in several iterations, facilitating the designer to re-do some steps. Specifically, in case system criticalities are verified, the process within the system view (and the design process) terminates since the system instance (together with its configuration) is acceptable; in turn, the system can be implemented in the real world in order to be used and operated by the human participants. However, should the system evaluation fail, the process branches into parallel paths. The designer can reconsider system structure and explore other design alternatives –in coordination with the human participant–, until a satisfactory solution is reached. In parallel, the design environment can automatically *annotate problematic (non-verifiable) criticalities*. Based on this annotation, related human criticalities can be tested for verification, leading to actions that the designer and human participant can take, i.e. reconsider human behavior, and then follow the aforementioned workflow. In case the human criticalities are verified and the concerns of the participant are satisfied, then the process of the human view ends.

5.3 Applying QoS meta-model

Design entities

As described in Section 5.2.2, the CPHS world can be viewed from multiple perspectives, i.e. the human, system and criticalities views. To construct a comprehensive profile [72] for a CPHS, we need to represent the concepts used in those views. This representation is based on standard modeling languages [197], like SysML or BPMN, as follows.

The proposed CPHS profile is depicted in Figure 5.5. Green-colored entities serve the human view, blue-colored entities illustrate proposed extensions belonging to the criticalities view, and grey-colored entities comprise the system view; the white-colored ones are standard entities that are provided by the modeling languages.

It is worth mentioning that these languages may not provide standard notation for representing all the concepts of the CPHS views. For example, BPMN itself does not provide the means to annotate business models with concerns or criticalities [213]. In addition, the current version of the SysML specification [199, 261] does not *directly* address human participants or criticalities off-the-shelf. However, SysML provides generic entities that can serve as their basis (such as the white-colored entities of Figure 5.5). We employ and extend these languages to represent CPHS design concepts.

The human view is served by BPMN, a graphical language that is proposed by the OMG [198, 231]; we utilize it as a standardized way to model the human participants along with their activities in the form of a sequence from start to end [180].

The *Human Participant* entity is used to model the roles of the humans participating in the process. The participant has specific concerns regarding the system functionality and operation, and performs activities within the constraints of the system; these activities are represented by the *Task* entity. In turn, task entities *serve* concerns, *satisfy* respective human criticalities and are *allocated to* physical CPHS entities. It is worth mentioning that the human participant is a specialization of a

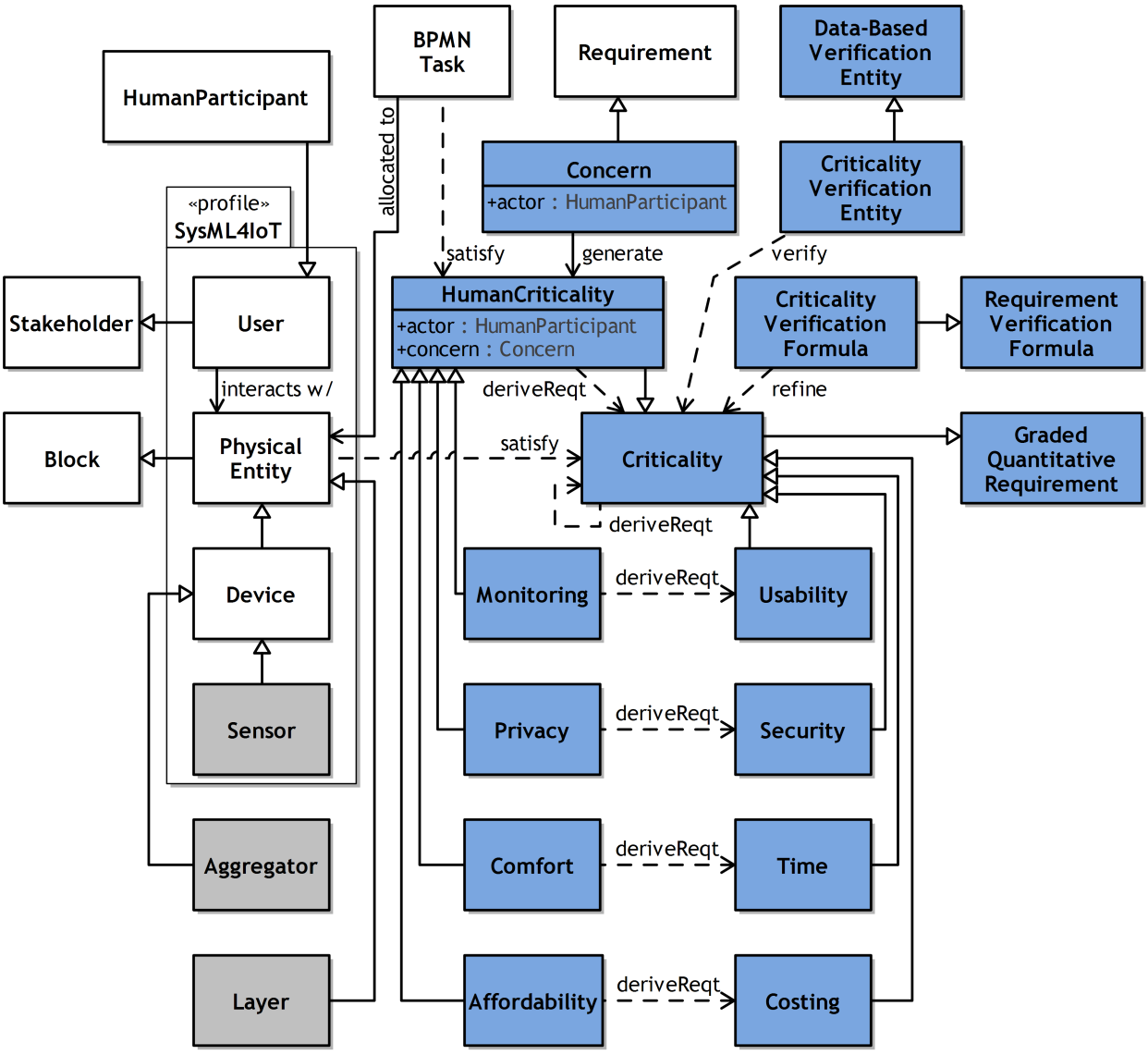


FIGURE 5.5: CPHS SysML profile.

User element which in turn is a specialization of the SysML *Stakeholder* [72]. Note that the user is a basic entity of the SysML4IoT [44] profile. This profile is aligned with the ISO/IEC/IEEE 15288 standard [295] and provides specific stereotypes to design IoT-based systems, like CPHS. Based on the SysML4IoT, we extend the user that interacts with elements of the system view.

The system view is served by SysML [72], an OMG modeling language for system design. In Figure 5.5, we employ the *Physical Entity* that is contained within the SysML4IoT profile; this element serves as the identifiable part of the physical environment that is of interest to the stakeholders of the system. Note that the physical entity is a specialization of the SysML *Block*, i.e. SysML's basic modular unit for representing entities (components) of a system [72]. Physical entities can be a hardware *Device*, an *Actuator* (e.g., on/off switch), or a *Sensor*. In our CPHS profile, the proposed system view comprises (i) the sensor, actively used by a human participant, (ii) the *Aggregator* which is a specialization of a device and is used to collect all the data transmitted from the sensor(s), and (iii) the *Layer* that acts as a physical entity representing the environment where the sensor(s) and aggregator device(s) exist and are communicating with each other.

The criticalities view is analyzed in the CPHS profile. The *Criticality* entity is a specialization of the more generic *Graded Quantitative Requirement*, which in turn is a specialization of the basic SysML *Requirement* [72]. Note that the graded requirement is an intermediate entity that supplies the levels property as a quantitative indicator (thus, the characterization "graded") that the corresponding criticality inherits. In addition, the criticality entity represents critical system requirements and is the basis for human ones. Specifically, we distinguish the human and system criticalities, having the criticality as the entity to serve system criticalities, while the human ones are depicted via a separate *Human Criticality* entity; the latter is a specialization of the criticality. This way, we can also depict the derivation of system from human criticalites; we connect the human criticality and the criticality entity (that represents system ones) with a *derive* relationship. There is another derive connection that begins from and points to the criticality entity itself, indicating that it may comprise other system criticalities. Regarding human criticality, it is generated by a *Concern* element that is a specialization of the SysML requirement.

Finally, we present the *Verification* and the *Criticality Verification Formula* entities that serve the verification process of the criticalities, leading to the overall evaluation of the system. The formula is the stereotype of SysML's *Constraint Block* [72] and represents the functions or expressions that refine the criticalities. These expressions are computed in a standardized fashion, i.e. via the graphical SysML *Parametric Diagrams* [72, 211]; the Verification element is a specialization of such a diagram. Parametric diagrams can convey input properties and values from system components, compute the expression, and extract the computed value. This value is the actual level of the associated criticality that can be in turn compared with the participant's desired criticality level; this results to the verification (or not) of the criticality, and consequently, its satisfaction by the components of the system.

Criticality types

Inspired by the bibliography [291, 225], we identify four criticality types for each view (human and system). We examine these particular types because (i) they apply to a variety of CPHS, independently of the specific technical properties of each CPHS implementation, (ii) they can encapsulate (group) more fine-grained criticalities within them, and (iii) are sufficient and necessary for a comprehensive CPHS design.

These criticality types are depicted in our CPHS profile (see Figure 5.5). The profile contains the relations between human and system criticalities and their specializations. Specifically, the green-colored entities represent human criticalities types, understandable by human participants,

while the blue-colored ones are system criticalities; note that system criticalities are derived by corresponding human ones.

Next, we describe the depicted criticalities in detail. The *Affordability* type refers to the budgetary constraints of the human participant with respect to the acquisition of CPHS components (or even the entire CPHS). This human criticality type derives Costing system criticalities, which constrain the actual price of the system components (or the overall costs of the system) [291]. The *Monitoring* type reflects the need of human participants for continuous and instantaneous operations within the system (e.g., display data without delay). This need is translated to *Time* system criticalities that relate to the immediate operation (e.g., real-time transmision of data over a network) of a CPHS [225]. *Privacy* criticalities refer to the protection of the personal data of human participants; when, how and to what extend their data is communicated to others is crucial for them during system operation [122]. Privacy-type human criticalities derive *Security* system criticalities [154], which mandate the provision of secure data communication and storage. *Comfort* human criticalities refer to the need for system devices that provide optimal comfort to the human participant e.g., in terms of ease-of-use. Such criticalities derive *Usability* type system design requirements [137].

Implementation

As mentioned in Section 5.2.2, some CPHS design actions (e.g., the derivation of system criticalities from human ones), can be automated using additional software modules. We have implemented this kind of automation by developing plugins (in the form of Java software modules), which accompany our defined CPHS Profile. These plugins are a specialization of the generic framework plugins described in Section 3.6. They are applied to the CPHS model in order to enrich it with additional functionality (which is performed automatically) and are integrated directly into the CSM modeling tool. In our work, we have implemented plugins to provide the following functionality.

(i) Generation of human criticalities. This plugin enables the generation of human criticalities from associated concerns of the human participant. With respect to the implementation, a menu within the design environment provides the means for the generation of human criticalities from concerns. The designer is facilitated and guided to insert criticality data (i.e. name, type, description and crititcality levels). Then, the corresponding human criticality is automatically generated within the system model.

(ii) Derivation of system criticalities. The automatic derivation of system criticalities from human ones is enabled via this plugin. A menu is provided by the modeling environment, prompting the designer to describe (i.e. insert a name and textual description) system criticalities. Following this description, system criticalities are automatically created as a result, derived from respective human ones. The desired levels of the human criticalities are translated by the designer into appropriate system criticality levels.

All system criticalities are in turn connected to appropriate CPHS components. These connections are formed in an automated fashion given that the aforementioned allocations between human activities and system components have been performed.

In summary, the derivation of system criticalities is a guided process which handles the complexity of forming the associated connections and it does so without errors or probability of forgetting a connection. Therefore, it takes care of keeping the model consistent and complete.

(iii) Configuration of system. Via the modeling environment, the system designer can choose an appropriate (pre-constructed) configuration for the system. Such configurations facilitate the decisions of the designer by giving her the freedom of option, abstracting away the complexity of property value generation. When the designer chooses a certain configuration among others, specific properties of the component(s) are populated with pre-defined values. Our plugin supports the exhibition of the available configurations in the form of a menu. After one configuration is selected, it automatically incorporates the corresponding property values. In a similar manner, we integrate Prolog rules, which are linked to system components, into the plugin and enrich the configurations. Specifically, going one step further from "available" options to "best" options, we execute these rules "under the hood" and derive the most suitable configuration that satisfies the majority of the associated criticalities. This configuration is encoded into the configurations menu and can help the designer choose the property values that correspond to the "best" system design.

(iv) Evaluation of system design. The computation of a criticality formula is performed within SysML parametric diagrams. As their name implies, specific parameters (and properties) are used to compute the formula. In a SysML design model, such a diagram cannot be created within the criticalities (i.e. SysML requirement entities), nor can the computed output be stored in them. For this purpose, the designer initiates the automatic creation of the verification criticality data entity, via a plugin action provided in the design environment. When the verification data element is created, it is connected by the environment to a respective system component entity, a criticality and a formula entity; specifically, the verification data entity (i) evaluates the component by computing its level, (ii) verifies the system criticality by comparing the computed level against the desired one, and (iii) holds the result of the formula, i.e. the computed criticality level. Note that the latter is extracted from a parametric diagram automatically incorporated to the verification data element. The formula is solved within the parametric using OpenModelica as the (math) solver; this solver is integrated into the modeling tool (Cameo Systems Modeler), and is incorporated within SysML parametric diagrams. Note that the properties may belong to different entities.

When the obtained criticality level does not verify the desired one, the frame of the "defective" criticality as well as its connection with the respective system component(s) becomes red-colored, notifying the designer that there is a problem. In addition, the modeling environment exhibits appropriate information to the system designer, as well as adjustments she can make. The latter may include suggested actions or solutions (like reconsidering the structure of the system). This process can occur several times –dynamically– in the system's design process, adjusting to changing operational circumstances. Thus, by assessing the modeling environment's automated feedback, the designer can further improve system design.

5.4 Applying cost meta-model

Similar to the RTS case study, we apply our custom Cost profile to CPHS, to enable the design and evaluation of such systems, from a cost perspective. The cost functions library is also used in this case, enabling computations related to the cost perspective, which is readily available to the designer during design time, along with the profile. To automate such computations, we employ parametric diagrams. The end results (i.e. the computed cost entity values) are in turn used to populate the parameters of cost profile elements; related requirements (e.g., conformance to budgetary constraints) are then verified by comparing the computed and desired values.

The profile is general and extendable enough to be applied in any system, leveraging the expressiveness and flexibility of SysML. This claim is further supported by applying the proposed SysML cost-related extensions, to the CPHS domain.

In the context of a CPHS, the challenge is to explore acceptable design solutions accommodating human (end-user) concerns regarding system operation, as well as CapEx restrictions. The SysML Cost profile enables a personalized cost analysis; it provides the CPHS designer with the necessary tools to assess costs at different levels, i.e. from the overall system CapEx cost to the individual acquisition expenses of single components, in an automated fashion.

The outline of the cost analysis is summarized as:

1. Cost entities are generated within the system model using the generic cost profile.
2. Following the definition of the cost entities, the designer can add and use functions, that are readily available in the cost inventory, to compute costs. In order to compute each cost value, SysML parametric diagrams, are employed. The equation takes input properties whose values are used to compute it; these properties may belong to different entities. The equation is solved within the parametric diagram (using OpenModelica [76] as the math solver) and the output cost value is extracted for further cost assessment. Note that using a parametric diagram within the modeling environment, engineers do not have to go through the complexity of writing additional code for the computations. The corresponding input values are simply provided while the engineers define the functions that will be used; after the computations are complete, the final required cost values are easily extracted and may be used for the cost analysis of the CPHS.
3. Composite costs are computed. Since the aforementioned functions refine a respective cost entity, the computed output value is stored within these cost entities. A composite cost entity may obtain its value from the sum of the values of other cost entities via automatic summations.
4. Costing requirements are verified. Defined cost requirements are either satisfied or not, leading to the system's cost evaluation. Specifically, a computed cost value is compared against the specified threshold and a result is extracted, i.e. "true" or "false", leading to the satisfaction of the requirement. The problematic –non-verified– requirement is annotated with a red-colored frame; this way, the engineer is informed about the defective element(s).

5.5 Case study: Criticalities in REMS Home system

In order to demonstrate the feasibility and applicability of the proposed approach, we employ a real-world case study, i.e. the REMS that belongs in the healthcare domain. In the following Sections, we begin with the description of a REMS, focusing on a specific subsystem, i.e. the home of the patient. In order to effectively design this subsystem, the three key views of our approach (i.e. human, criticalities, and system) are applied to it; we start with the human (here, patient) view where human behavior within the REMS is defined, patient concerns are expressed and patient criticalities are generated (see Section 5.5.1). We then explore the derivation of REMS Home system criticalities from human ones; patient and system criticalities compose the criticalities view of the REMS Home (Section 5.5.2). Following this, in Section 5.5.5, the REMS Home system view is analyzed, where defined system criticalities are verified (or not), leading to the evaluation of the REMS Home.

As described in Section 2.8.2, REMS is a remote healthcare monitoring system [14] for elderly individuals.

Here, we focus on the REMS Home subsystem, where the elderly patient resides and may operate appropriate medical equipment in the context of the REMS CPHS. Note that this subsystem is essentially the front-end of the entire REMS, directly visible and closest to the human participant. In an example Home scenario, an elderly individual has a successful cardio-surgery. After a few days she is sent to her home where she can live independently and recover, although daily monitoring of her physiological signs by health caregiver(s), is mandatory. For this purpose, the caregiver has proposed the formation of a REMS for the patient's Home via the acquisition of the following components.

- An Electrocardiogram (ECG) sensor that can measure the heart rate of the patient and detect heart failure, arrhythmia and strokes [1].
- A small central communication unit, i.e. a data aggregator that is used to collect the sensor-generated data and directly transmit it to a remote healthcare facility for monitoring and assessment by the caregiver(s).

In the context of the REMS, we map the generic roles identified in Section 5.2.2 for CPHS, as follows: the system designer is the REMS designer that models the most suitable REMS solution for a patient, while the patient is the user and primary REMS participant who expresses her needs/concerns to help the designer achieve this solution.

5.5.1 Defining human behavior

In this Section, we apply the "define human behavior", "define human concerns", "generate human criticalities", and "relate human criticalities to human behavior" actions of the CPHS design process (see Figure 5.4) that are performed within the human view and the criticalities view.

Figure 5.6 depicts how a REMS patient is actively participating in the design process. Human and criticalities views are presented next to each other. The patient can easily observe both together, seeing the tasks that she performs (left side - human view), and actually affect the views, declaring her needs and criticalities that are met by these tasks (right side - criticalities view).

Via BPMN, the designer defines the primary REMS activities that an elderly patient performs within her own home, which is the primary focus of the designer w.r.t. the patient view. In turn, the designer describes these activities to the patient. As an example of core REMS home activities, the patient wears and activates a wearable medical device, measures her vital signs and views them to check whether there is a problem with her medical condition. Note that the latter viewing activity is also performed by remote medical personnel; however here the focus is on the activities of the patients within their own home. In Figure 5.6, such activities are displayed as green-colored elements; indicatively, we focus on the aforementioned REMS activities, performed by the patient, i.e., the "activate device", the "measure vital signs" and "view vital signs". In turn, the patient expresses her critical concerns regarding REMS operation to the designer. An example would be: "I want to be able to charge my heart sensor device once in a while". The designer documents these concerns in a simplified form to the system model.

In Figure 5.6, distinct concerns, which REMS patients have actually expressed (see Section 5.5.8), are illustrated (right diagram named "Criticalities view") as blue-colored entities. As indicative examples, we focus on two of them.

The first concern is related to the activation, use and deactivation of medical devices. These activities are constrained by the battery lifetime of the devices; this is a typical constraint in all systems with wearable electronic devices. Thus, the patient expresses the following concern: "I want the device to have a long battery lifetime. This way, I feel comfortable to use it without needing to charge it most of the time." The designer documents this need, creating the corresponding "Lasting_battery" concern element.

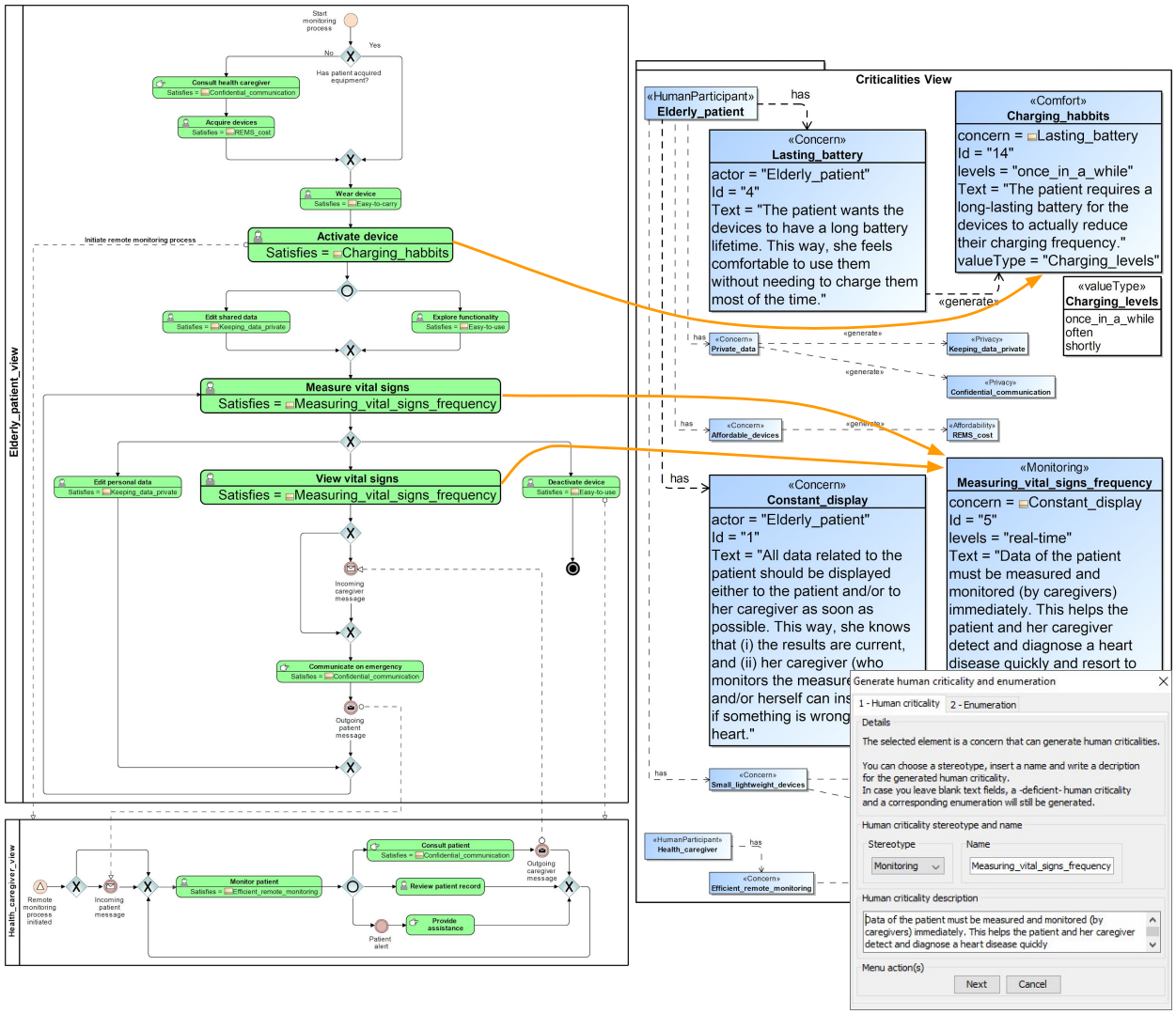


FIGURE 5.6: Mapping of patient activities to patient concerns and criticalities.

The second concern is related to the measurement of the patient's vital signs (e.g., heart rate) using a medical device. The patient expresses her concern from her point of view: "My medical data should be displayed to me and/or my caregiver as soon as possible. This way, I know that (i) the results are current, and (ii) my caregiver (who monitors the measurement) and/or I can instantly see if something is wrong with my heart." This expression is inserted as text into a Concern element, which is created by the designer and is titled "Constant_display".

In a similar manner, concerns regarding: the privacy of the medical data of the patient when they are sent to and seen by the caregivers, the total cost of acquiring the REMS home equipment, as well as whether the devices provide comfort (e.g., they are small, light-weight, etc.), are also created capturing additional needs of the patient(s).

Based on the proposed design process (see Section 5.2.2), these concerns generate criticalities. The designer inserts the type of criticality (e.g., comfort, privacy, monitoring, or affordability) and a textual description (in cooperation with the patient). The –desired– criticality levels are solely provided by the patient. For example, the generated privacy human criticality may have "high", "low" or "no privacy" levels; the patient expresses its high importance for her by choosing "high" as the desired level.

The plugin menu provided by the design environment to generate human criticalities (described in the implementation Section 5.3) is depicted in Figure 5.6 (down right).

Considering our two REMS indicative examples, the "Lasting_battery" concern refers to an increased mobility, allowing the elderly patient to conveniently use the REMS at her home without keeping devices plugged in a power outlet most of the time. This guides the designer to generate a corresponding comfort criticality, named "Charging_habits". The translation takes this particular form because the designer –cooperating with the patient and examining her feedback– elicits that the patient requires a long-lasting battery to actually reduce the charging frequency; battery and frequency are inversely proportional. The expressed levels for this criticality are the following: "once_in_a_while", "often", and "shortly". In this case, to match the patient's exact requirement, the patient selects the value "once_in_a_while" for the respective level property.

The "Constant_display" concern guides the designer to generate a monitoring human criticality, named "Measuring_vital_signs_frequency". This criticality falls into this particular type (monitoring), since it reflects the importance of the monitoring conditions of the patient's vital signs, while they are being measured. When this occurs as soon as possible, it helps the patient and her caregiver detect and diagnose a heart disease (e.g., a cardio-vascular anomaly) quickly and resort to proper medical countermeasures. The patient expresses and selects the "real-time" value for the criticality's level property, indicating that she needs REMS to operate immediately; other patient-defined level values are "quickly" (several seconds delay), and "non-important" (i.e. delayed).

These criticalities should be satisfied by defined patient activities. For example, the "Measuring_vital_signs_frequency" and the "Lasting_battery" human criticalities should be fulfilled by the "Measure vital signs" and the "Activate device" patient activities, accordingly. The formation of such relations is critical; if the execution of an activity within the REMS context fails one or more patient concerns, the system is considered unusable and should be redesigned accordingly. In Figure 5.6, "satisfy" relationships are depicted as a mapping between the human view and the criticalities view (orange lines).

5.5.2 Exploring criticalities definition and derivation

In this Section, the next two steps of our design process, i.e. "derive system criticalities" and "define system criticality verification formulas" are described. Note that these are solely actions of the system designer executed within the criticalities view.

At first, the designer defines an abstract REMS home structure. In Figure 5.7 three system components, defined in an abstract form (i.e. as simple named system components without set properties), are depicted. These components are modeling actual REMS devices or layers. Specifically, the Shimmer3-ECG is a wearable medical sensor device, the ODROID-XU4 is a medical data aggregator device, and the Home of the elderly patient acts as the encompassing layer. Each component only holds specific operations that represent its respective functions; for example, the ECG device is powered on/off, monitors a patient's heart function, streams/pushes the generated data to the aggregator and alerts in case a measurement surpasses a normal value threshold.

These components support the execution of patient activities. For example, the patient activates a medical device that should measure her heart rate. To support this activity, a medical ECG sensor device –performing this measurement– should be defined as a component of the REMS structure. The designer connects activities and corresponding components with an "allocation" relationship. In Figure 5.7, the "activate device" activity is allocated to the "power on/off" function of the Shimmer device, the "measure vital signs" primary activity is allocated to the device (as a whole), and the "view vital signs" is connected to both the data aggregator and the home of the patient; these allocations are represented as red-colored lines from activities to components in the

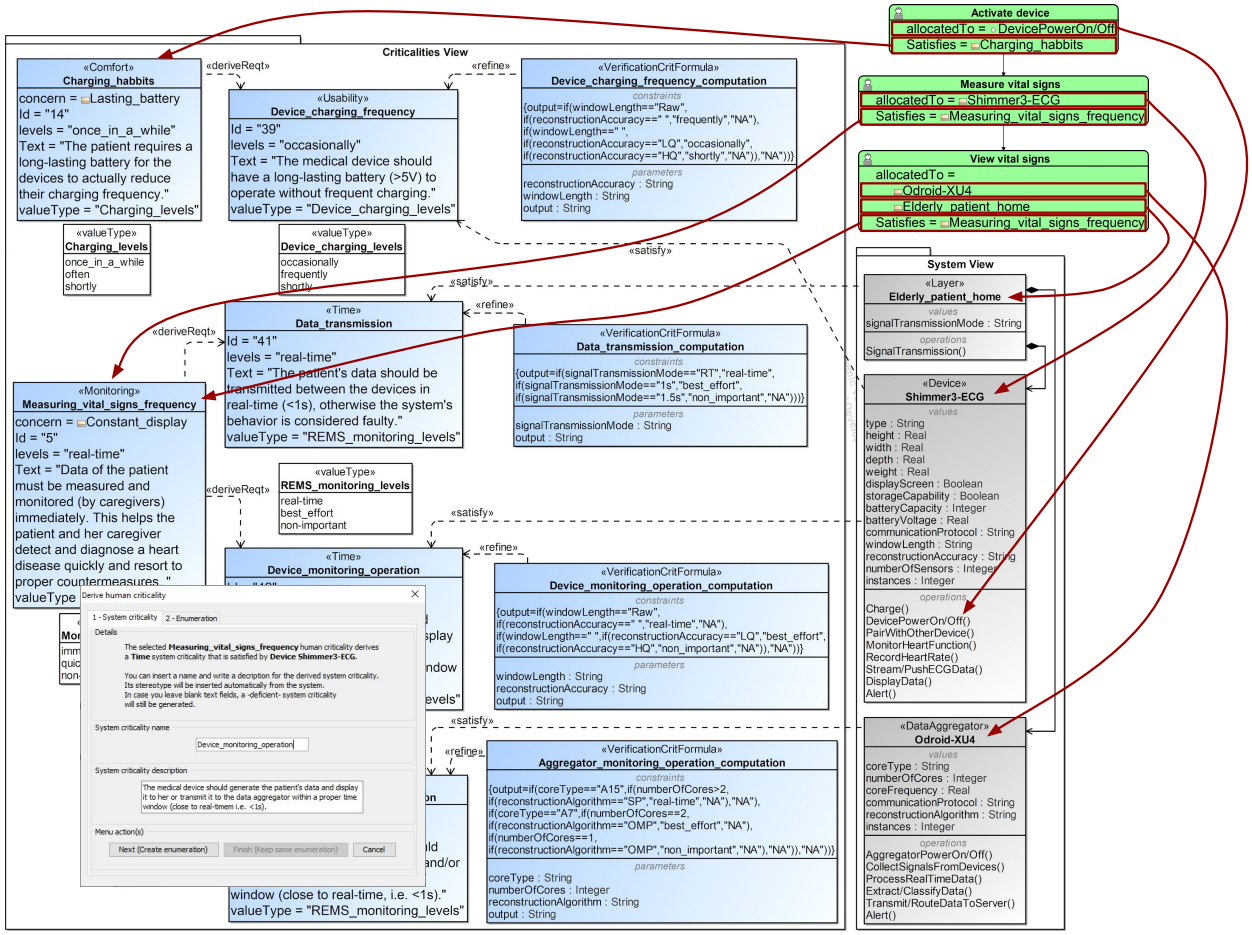


FIGURE 5.7: Connections between views, and derivation of system criticalities from human ones.

figure. Note that the aforementioned activities are also connected to the respective human criticalities (see Section 5.5.1) via "satisfy" relationships; red-colored arrows are used to also depict these connections. At this point, the REMS designer can derive system criticalities and connect them to REMS home components. Consider, in our use case, the "Measuring_vital_signs_frequency" and the "Charging_habbits" human criticalities (analyzed in the previous section), expressed by the patients.

The first one is a "monitoring"-type criticality that complies with the need of REMS patients for real-time operation; thus, they have provided "real-time" as the value of the criticality's level. This criticality is related to two primary patient activities that in turn are allocated to all three REMS home system components. Three system criticalities are derived, each connected to a respective system component. Specifically, "Time" system criticalities that restrict the whole system, i.e. the home of the patient, and/or its components, i.e. medical sensor device and data aggregator, are created. Note that in this specific case, the level property of the newly created system criticalities has the same value as the human one, i.e. "real-time". In other cases, following the criticalities transformation, the criticality levels are also translated to respective classes, that are more suitable (e.g., system-wise) to express the levels of the system criticality.

The second one is the "comfort"-type "charging frequency" human criticality; it represents the need of patients to conveniently use a medical device without charging it frequently and is thus related to the usability of the system. A "once_in_a_while" level value has been expressed by the

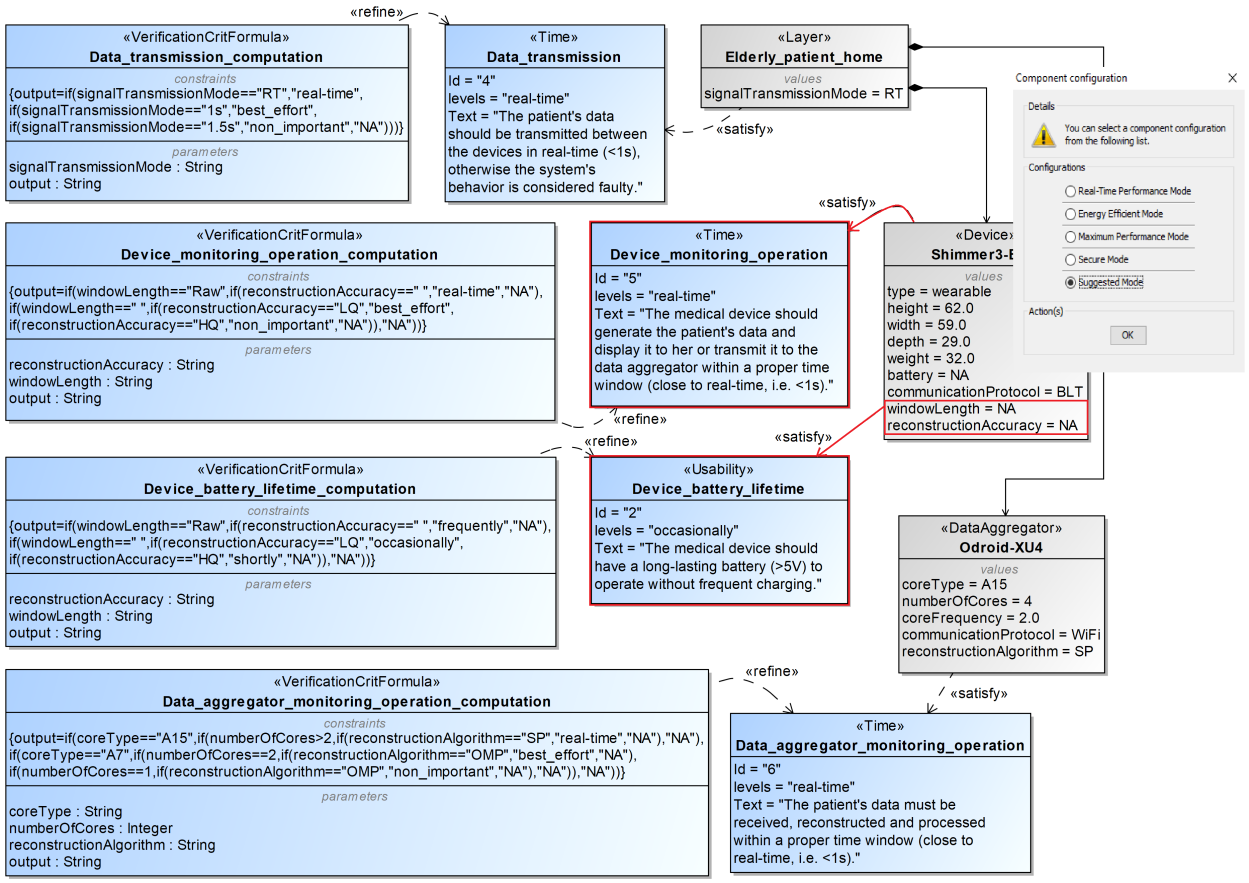


FIGURE 5.8: REMS design - Suggesting system configurations.

patients for this criticality. In turn, this criticality derives a "Charging_frequency" system criticality for the device. Note that the system criticality has the "occasionally" level value, translated from the "once_in_a_while" level of its human criticality.

Based on the CPHS design process, each derived system criticality should be refined via formulas that can be computed; this leads to the verification of the criticality via the computed result. This result is the real criticality level that the system actually provides. For example, as shown in Figure 5.7, the "Device_charging_frequency" is refined by a respective "VerificationCritFormula" that specifies the output of a logical expression, regarding the times a device must be charged to be operative. The formula holds a defined expression that is a simple if-else statement, e.g., if the device has low quality (LQ) reconstruction accuracy, then the output level is "occasionally".

5.5.3 Automated decision-making

We follow the indicative scenario of our case study. At this point, the REMS designer comes into play, creating a complete REMS home model based on the aforementioned components, taking into account system requirements identified by the patient [195]. The designer aims to achieve a satisfactory REMS design; the external decision-making system facilitates the designer to reach efficient solutions that can lead REMS patients to receive high-quality healthcare services.

In Figure 5.8, an excerpt of the REMS home model is depicted. Grey-colored entities depict three basic components of the REMS home: the *Shimmer3-ECG*, i.e. the medical sensor device that measures and records the vital signs of the patient, the *Odroid-XU4*, i.e. the data aggregator that

collects all the sensor-generated medical data, and the *Elderly_patient_home*, i.e., the layer where the elderly patient resides and operates the devices; the layer contains the other components. All components hold specific properties that characterize them, e.g., the Shimmer device has a battery capacity, communication protocol for connection to other devices, window length for the generated data, reconstruction accuracy for the transmission of the data, and size properties.

Each system component is connected to blue-colored requirements via *satisfy* relationships. In our example, indicative requirements regarding the monitoring operation of the whole REMS (e.g., medical data transmission) and the battery lifetime of the device are illustrated. Each requirement has its own description and holds the *levels* property where a desired level is specified. For example, the "Device_monitoring operation" requirement holds a "real-time" level value, indicating that the medical data of the patient should be managed immediately, while the "Device_battery_lifetime" requirement has the "occasionally" level value, indicating the need of the patient to charge the medical device once in a while.

The aforementioned requirements are connected to blue-colored requirement verification constraints via *refine* relationships. For example, the "Device_monitoring_operation_computation" constraint refines the corresponding requirement and holds a specific "if-then-else" formula and parameters; these parameters refer to properties of the system component (here, Shimmer device) that should satisfy the requirement. Indicatively, when the *windowLength* and *reconstructionAccuracy* properties of the device obtain the "raw" and blank (" ") values accordingly, then the device monitoring level is "real-time".

Having a complete system model at hand, the REMS designer may configure the system; via the modeling environment the decision-making plugin is activated. This mode (selected in Figure 5.8) is encoded into the configurations menu and can help the designer choose the property values that correspond to the most satisfactory system design. These values may refer to a full solution, i.e. all defined requirements are satisfied, or a partial solution, i.e. some requirements may be satisfied.

In the following, based on the transformations of SysML system model entities to Prolog entities, we present listing 1 that is an indicative Prolog knowledge base that connects components, properties and requirements in a REMS home context. For demonstration purposes, we focus on the medical sensor device and two indicative requirements that are associated with this device, namely the battery lifetime and the monitoring operation requirements.

The fact that the device and its requirements are associated, is encoded by a component-requirement Prolog rule that has two parts: (i) the declaration of the device along with two specific properties that are important for satisfying the requirements, i.e. the reconstruction accuracy of medical data, and the window length of this data, and (ii) the declaration of the actual requirements that are associated to the device (e.g., device_Battery_Lifetime). The latter contain the same properties as the device, and the name of their level.

The association between the device and requirements is actually a satisfaction relationship; this is declared within a requirement satisfaction Prolog predicate, namely the "deviceSatisfyLevel". The device that should satisfy the requirements is declared, along with all its properties. In addition, the values of the required levels, in order for the requirements to be satisfied, are also specified. In our example, the "occasionally" level value should be achieved w.r.t. the battery lifetime requirement; w.r.t. the monitoring operation requirement, the "real-time" level value should be achieved.

In order to check whether these level values can actually be reached (or not), two more Prolog entities are employed. Firstly, the value-range Prolog rules describe how these values are achieved through device properties. In our example, they encode the names of the device properties and the requirements' levels, while setting the range of permissible values, e.g., the "ReconstructionAccuracy" can take either the "LQ" or blank value, while the "DeviceBatteryLifetimeLevel" may be

"occasionally" or "frequently", accordingly. Secondly, following these rules, different instantiations of the device, i.e. property values that achieve different requirement levels, are specified via exact-value Prolog predicates. Indicatively, w.r.t. battery lifetime, when the reconstruction accuracy is "LQ" (i.e. low quality), the device can be charged "occasionally", while the device's monitoring operation level is "best effort".

Finally, the aforementioned predicates and rules are combined into a single rule. The REMS designer wants to configure the medical sensor device for the REMS home; the general configuration rule encodes that the device should satisfy the levels that are declared in the requirement satisfaction predicate (i.e. "occasionally" and "real-time"), subject to the battery lifetime and monitoring operation requirements.

In the context of the REMS home model, the plugin queries automatically the Prolog engine to retrieve the configuration (property values) that achieves the desired requirement levels. In the case presented in Figure 5.8, a partial solution has been found, as explained in the following. The property values of the layer and data aggregator are returned from the Prolog constraint solver and are incorporated into the REMS model. Based on these, the monitoring requirements of the layer and the data aggregator are satisfied. However, the monitoring and battery lifetime requirements of the device cannot be satisfied by the chosen configuration; a solution that covers both requirements was not feasible, i.e. non-applicable values ("NA") were returned to the component. Thus, these requirements are annotated as red-colored, indicating that there is a problem. In addition, the design environment shows appropriate information to the designer, and adjustments she can make, e.g., suggested actions, like specifying another level for the requirements.

LISTING 5.1: REMS model to Prolog predicates and rules.

```
/*i. Exact-value Prolog predicates*/
device_Battery_Lifetime('LQ', '␣', 'occasionally').
device_Battery_Lifetime('␣', 'Raw', 'frequently').
device_Monitoring_Operation('␣', 'Raw', 'real-time').
device_Monitoring_Operation('LQ', '␣', 'best_effort').

/*ii. Value range Prolog rules*/
device_Battery_Lifetime(ReconstructionAccuracy, WindowLength, Device_Battery_Lifetime_Level) :-
WindowLength\=='Raw', WindowLength=='␣', ReconstructionAccuracy=='LQ', Device_Battery_Lifetime_Level=='occasionally'.
device_Battery_Lifetime(ReconstructionAccuracy, WindowLength, Device_Battery_Lifetime_Level) :-
WindowLength=='Raw', ReconstructionAccuracy=='␣', Device_Battery_Lifetime_Level=='frequently'.
device_Monitoring_Operation(ReconstructionAccuracy, WindowLength, Device_Monitoring_Operation_Level) :-
WindowLength=='Raw', ReconstructionAccuracy=='', Device_Monitoring_Operation_Level=='real-time'.
device_Monitoring_Operation(ReconstructionAccuracy, WindowLength, Device_Monitoring_Operation_Level) :-
WindowLength\=='Raw', WindowLength=='', ReconstructionAccuracy=='LQ', Device_Monitoring_Operation_Level=='best_effort'.

/*iii. Component-requirements Prolog rules*/
device(WindowLength, ReconstructionAccuracy) :-
device_Battery_Lifetime(WindowLength, ReconstructionAccuracy, Device_Battery_Lifetime_Level),
device_Monitoring_Operation(WindowLength, ReconstructionAccuracy, Device_Monitoring_Operation_Level).

/*iv. Requirement satisfaction Prolog predicates*/
deviceSatisfyLevel(device(Battery, CommunicationProtocol, Depth, Height, ReconstructionAccuracy, Type,
Weight, Width, WindowLength), 'occasionally', 'real-time').

/*v. System configuration Prolog rule*/
configure([device(Battery, CommunicationProtocol, Depth, Height, ReconstructionAccuracy, Type, Weight,
Width, WindowLength)]) :-
deviceSatisfyLevel(device(Battery, CommunicationProtocol, Depth, Height, ReconstructionAccuracy, Type,
Weight, Width, WindowLength),
device_Battery_Lifetime(ReconstructionAccuracy, WindowLength, Device_Battery_Lifetime_Level),
device_Monitoring_Operation(ReconstructionAccuracy, WindowLength, Device_Monitoring_Operation_Level).
```

5.5.4 REMS home cost computation

In the context of REMS, each patient's healthcare needs may vary, resulting into different configurations of the system corresponding to different acquisition budgets. The designer, working for the medical facility, should assist a patient to choose an appropriate REMS Home configuration taking also into account budgetary constraints. Needless to say that a configuration is considered suitable only if it fits patient health monitoring requirements. However, patients may be tempted to acquire more expensive solutions than they really need. The designer should assist them to avoid such a predicament or at least make a conscious decision on their REMS Home configuration.

In order to do that, the system designer creates the REMS Home SysML model. The SysML Cost profile enables a personalized cost analysis; it provides the REMS designer with the necessary tools to assess costs at different levels, i.e. from the overall REMS CapEx cost to the individual acquisition expenses of medical and data communication devices, in an automated fashion.

The designer constructs the REMS Home model starting with its structure (see Figure 5.9). Three basic components are defined, along with specific properties that describe them: (i) Shimmer3-ECG [30], i.e. the medical device, comprising an ECG sensor; size, battery, communication protocol or signal reconstruction accuracy are some of the properties of the sensor device that characterize it. (ii) Odroid-XU4 [115], i.e. the data aggregator; properties such as type of CPU core or number of these CPU cores assist the designer to fully describe this entity. (iii) Elderly_Patient_Home, i.e. the abstract representation of patient Home REMS, containing the sensor device(s) and the data aggregator(s) which are connected and communicate with each other.

To depict the needs and concerns of a patient, the designer defines requirements attached to REMS components. The patient's views are depicted as abstract requirements, described by text and a quantitative levels property, whose value indicates the desired level that covers the needs of the patient. This way, it is easier for patients to describe their design requirements. However, this information is too abstract to result in analytical system configurations. Thus, each requirement is refined by the designer to describe requirement levels, by formulas which are computed based on properties of the REMS components. These formulas are described by the designer and are applied to different patient configurations. They are stored in SysML VerificationReqFormula entities, refining requirements, as shown in Figure 5.9. VerificationReqData entities are defined to store the actual computed value of each requirement's level, corresponding to a specific configuration. They can be compared to the desired levels to verify that each requirement is satisfied [195].

Indicatively, the Device_Battery_Lifetime requirement is associated with the Shimmer sensor in Figure 5.9, holding the levels property with an "occasionally" value, meaning that occasional charging is required to refill the battery of the Shimmer device. The VerificationReqFormula entity refining this requirement provides the formula for computing the Device_Battery_Lifetime requirement level.

The Equipment_Acquisition is a cost-related patient requirement (it is termed costing, as shown in Figure 5.9), indicating the level of expenses required by an elderly patient for purchasing a REMS Home configuration. Thus, it should be associated to the Elderly_Patient_Home entity. However, this is a cost requirement. Thus, since there is a need to assess the costs of the overall solution, the designer creates a composite CapExCost entity (see Elderly_Patient_Home_CapEx_Cost in Figure 5.9) that represents the total capital expenses of the REMS Home, and connects it (via estimation relationship) to the top component of the structural hierarchy, i.e. the Elderly_Patient_Home. Note that this is a composite CapExCost entity, which shall be computed in a latter stage. It is created at this stage to associate the Equipment_Acquisition requirement with it.

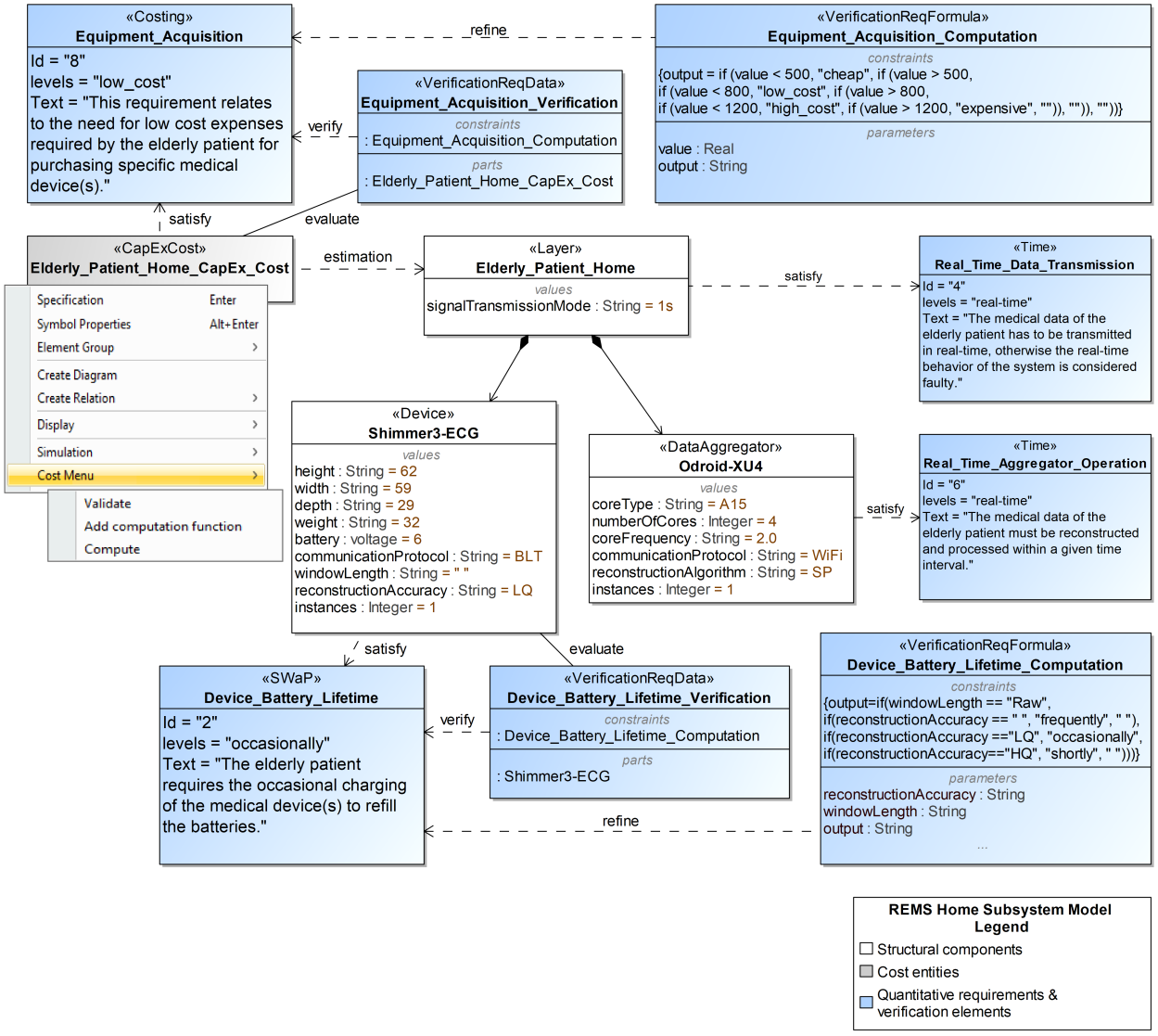


FIGURE 5.9: REMS Home model.

Note that in our modeling tool, i.e. Cameo Systems Modeler (CSM) [32], the CapExCost entity is accompanied by a Cost Menu that contains specific buttons; these buttons correspond to steps that the designer can follow in order to perform a complete and accurate cost analysis. Specifically, the designer can (i) automatically generate cost entities for the child structural components, (ii) add computation functions to extract the values of individual cost entities, and (iii) automatically compute the values of aggregated and composite cost entities. The Equipment_Acquisition requirement, depicted in Figure 5.9 is of a "low_cost" level. Thus, based on the formula, defined in the Equipment_Acquisition_Computation VerificationReqFormula, the acquisition cost of the equipment for the patient should not exceed "800" Euro. Is this possible taking into account the other requirements set for the patient? The operation of the system should be performed in "real-time" to meet the patient's healthcare needs, while the patient wished to have to charge the sensor "occasionally". How are cost requirements related to other requirements? All of them are estimated based on the configuration of the system, e.g. the properties of the Shimmer3-ECG and Odroid-XU4 components. Our SysML Cost profile should assist the designer to associate CapExCost cost entities to these components, specify functions for the cost estimation of each of them, and automatically compute the values of Elderly_Patient_Home_CapEx_Cost already defined by the designer. Note that this can be based on automated –recommended– pre-configured setups.

(i) Generate cost entities

The CapExCost entity, connected to the Elderly_Patient_Home layer, is composed of aggregated costs, i.e. the AcquisitionCost and IntegrationCost entities, connected to the same component. In addition, each child of the layer, i.e. the device and the data aggregator, should be connected to respective AcquisitionCost and IntegrationCost entities. Note that the AcquisitionCost is chosen to demonstrate the cost of purchasing these components, while the IntegrationCost represents the cost of integrating them, according to the SysML Cost profile (see Section 3.14). The profile facilitates the automatic generation of sub-cost entities from a composite one, in this case the Elderly_Patient_Home_CapEx_Cost (see Figure 5.10). They are automatically created, initialized and connected to the components of the system; even if the designer neglects the insertion of a cost entity, our approach integrates it automatically to the model. In the REMS Home case, due to the minimal number of components, the contribution of the IntegrationCost entity to the overall CapEx is negligible. Although cost entities are defined automatically, the system designer should add their properties used for their computation, as explained in the next section.

It is worth mentioning that all REMS-related data were provided by the Hamad Medical Corporation and the College of Engineering at Doha University, in Qatar, as part of the EMBIoT research project [11]. Due to the sensitive nature of this medical data, it remains proprietary; we were allowed to disclose only high-level (anonymized) insights and information. In addition, according to the hospital's policies, devices' cost data remain constant on a yearly basis. Note that, irrespective of change frequency, the system designer can easily incorporate any data-related alterations (e.g., the price of a REMS device may be increased or decreased) within the model. Computations encapsulating supply-chain backlogs, back orders, etc., or even promotional discounts, can be inserted in the model in the form of additional functions by the system designer according to his/her modeling objectives.

(ii) Add cost computation functions

The cost entities connected to the device and the data aggregator are considered as individual, i.e. their cost values are computed one by one by custom cost computation functions; in our case, only the AcquisitionCost entities are computed, while the values of the Integration costs are set to zero.

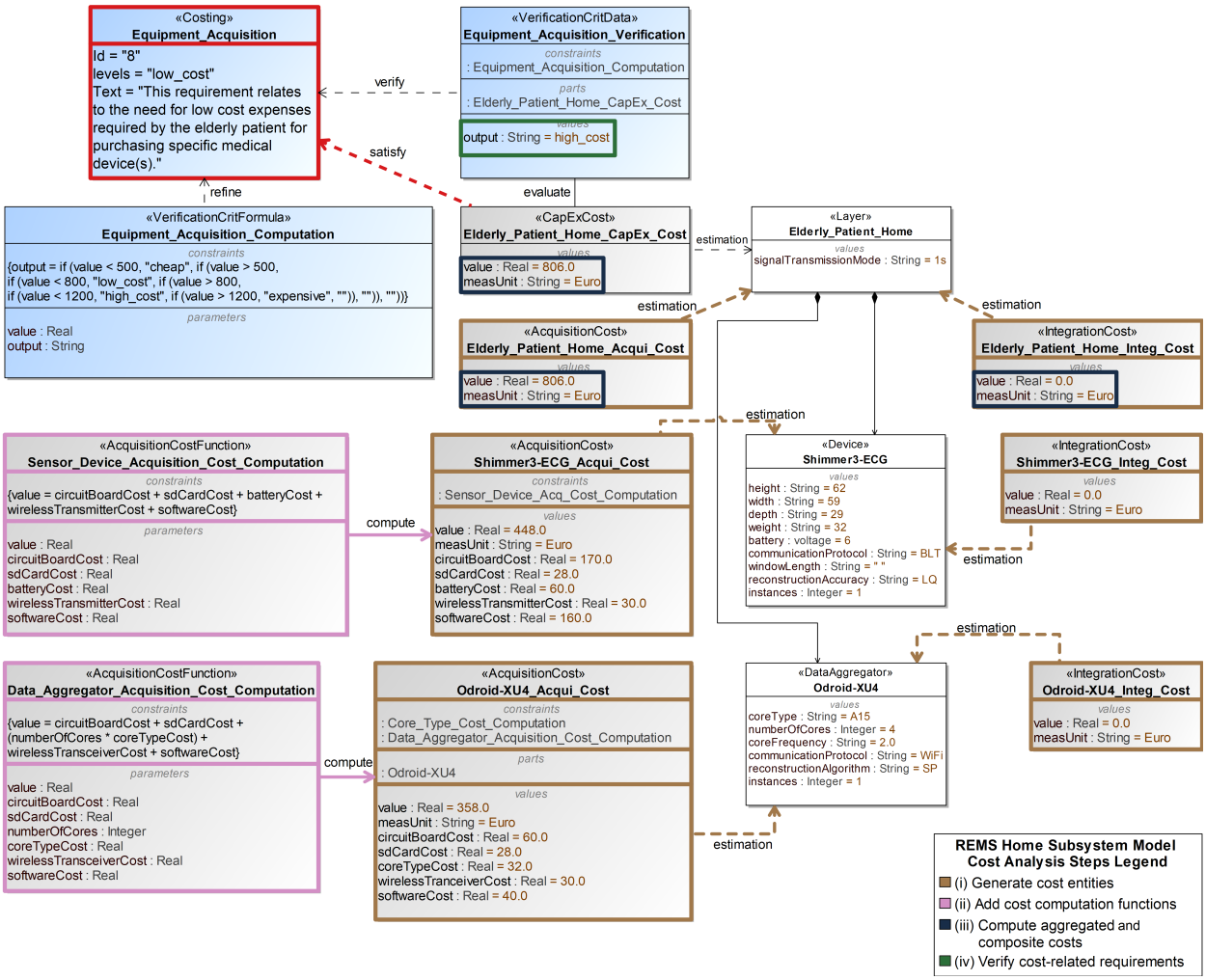


FIGURE 5.10: CapEx cost computation in the REMS Home model.

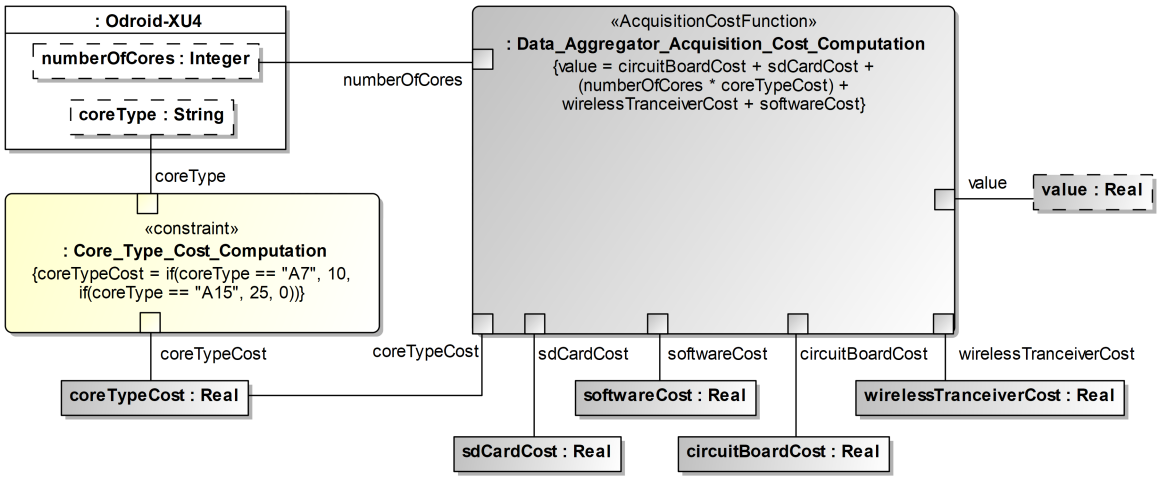


FIGURE 5.11: Odroid-XU4 data aggregator acquisition cost computation.

When our SysML Cost profile is applied to the REMS Home system model, functions from the inventory are readily available for computation; these functions are solved within the modeling tool in SysML parametric diagrams, contained in respective individual cost entities, using OpenModelica [205] as the (math) solver. Indicatively, the parametric diagram for Odroid-XU4 aquisition cost is illustrated in Figure 5.11. Note that input properties from the cost entity and system components may be used for the computation of the function, while the output is the resulting cost value, stored within the cost entity. It is worth mentioning that our approach supports dependencies between functions; for example, in Figure 5.11, there is a secondary function within the same parametric diagram (see yellow-colored block) which produces the value of a property that is in turn used as input to the main cost computation function. Here, the cost of different core types is first computed and then used to compute the cost value of the multi-core Odroid data aggregator.

(iii) Compute aggregated and composite costs

Following the addition of the required cost computation functions, the latter are automatically computed, while respective cost values are extracted and stored to associated cost entities. In turn, based on these values, the values of aggregated costs can be computed via simple summations. The AcquisitionCost entity, connected to the Home layer, is aggregated; therefore, it obtains its value only from the summation of the Acquisition-type costs of the device and the data aggregator. The same applies to the respective Integration costs. The CapExCost entity of Home layer is derived by the summation of the values of the aggregated cost entities.

(iv) Verify cost requirements

In order to verify the Equipment_Acquisition costing requirement depicted in Figure 5.10, the value of the CapExCost entity is employed; the corresponding VerificationReqFormula entity receives this value as input, initiating the verification process, while the actual value of the cost-related level is computed within it (via parametric execution). In the case of REMS Home model presented in Figure 5.10, CapExCost value is "806" Euro, resulting in a "high_cost" level of the Equipment_Acquisition costing requirement. Thus, the requirement may not by met, thus, it is automatically framed with red color.

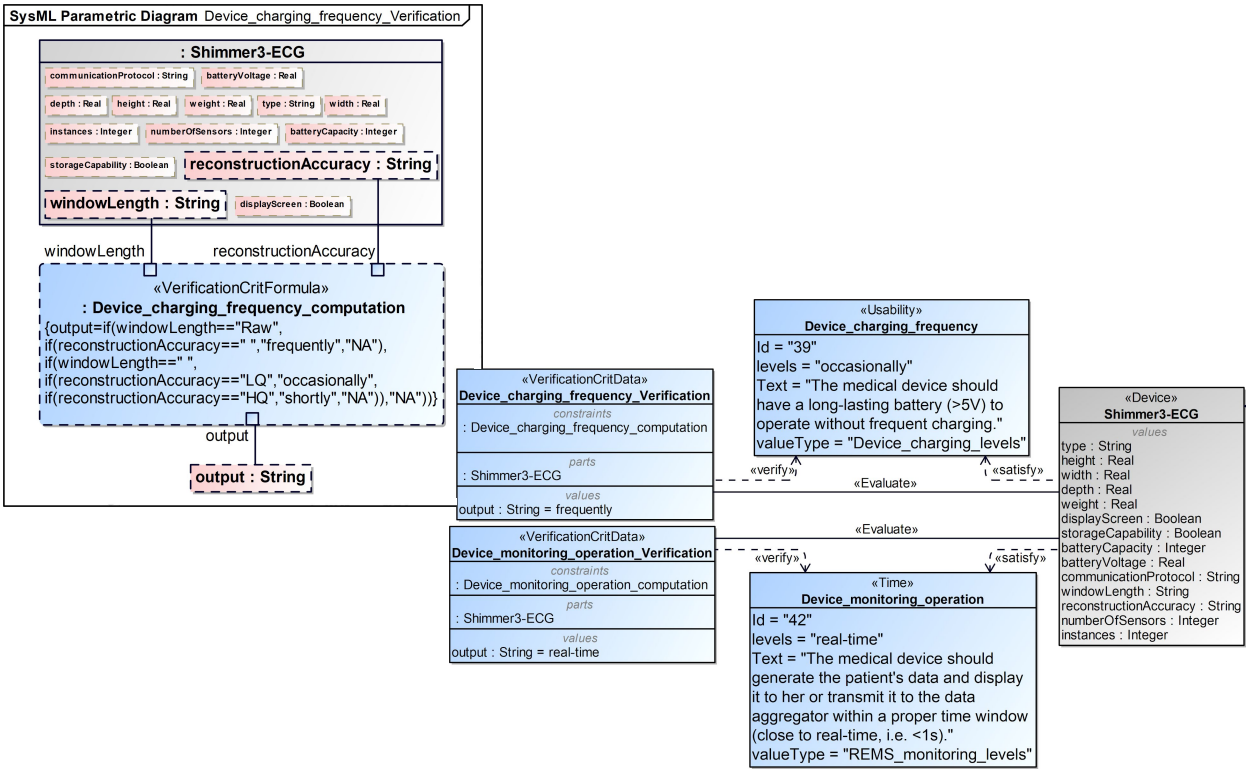


FIGURE 5.12: Computation of "Device_charging_frequency" criticality level using parametric diagram.

It is worth pointing out that when there is a verification problem, our approach enables the modeling environment to automatically exhibit appropriate information to the designer, as well as adjustments that he/she can make. The latter may include suggested actions or solutions (like re-configuring the system) to reduce costs. This process can occur several times –dynamically– in the design process of the system, adjusting to changing operational conditions [195].

5.5.5 Criticalities verification

In Figure 5.12, an excerpt of the REMS system view is depicted. Indicatively, the Shimmer device is illustrated holding specific properties. This device should satisfy the "device charging frequency" and the "device monitoring operation" system criticalities. Each of these criticalities is connected to auxiliary verification data entities; in our case, the "device charging frequency" system criticality is verified by the "device charging verification" entity and the "device monitoring" entity is related to the "device monitoring verification" entity.

On the left of the charging verification data element a parametric diagram is also shown. Input and output properties from related system (and other) entities are used to compute the Verification criticality formula. Note that the properties may belong to different entities. For example, the inputs are taken from the Shimmer3-ECG's properties, i.e. the reconstruction accuracy or the window length. The output property is part of the verification data element and holds the extracted output value of the equation.

Having the capability of system verification, the system designer can choose to configure the system by populating the properties of the system components with appropriate values. These

values are used as input to a parametric diagram to compute formulas. For example, the Shimmer ECG device holds the "reconstruction accuracy" property that can take the value of "LQ", i.e. low quality. The "window length" is another property with "Raw" as its value, i.e. data are transmitted as soon as they are measured.

After system configuration, our approach facilitates the designer to evaluate the REMS and check whether a criticality is satisfied by the corresponding model components. She can examine and verify the criticalities either individually or altogether. She can then assess the modeling environment's automated feedback, and further improve the system design. Specifically, individual REMS elements that fail to satisfy the required criticality degrade the criticality's overall level. Such elements are indicated in the modeling environment so that the patient can detect a problematic criticality and the REMS designer can focus on improving it. Automated suggestions are also provided to further facilitate this activity.

In Figure 5.13, the evaluation of REMS model is depicted, where a criticality is defective, since it cannot be verified and satisfied by the current system. This criticality is connected to other elements that are represented in the system view, the criticalities view or human view, thus, the patient and the designer can easily be informed about the problem in either view.

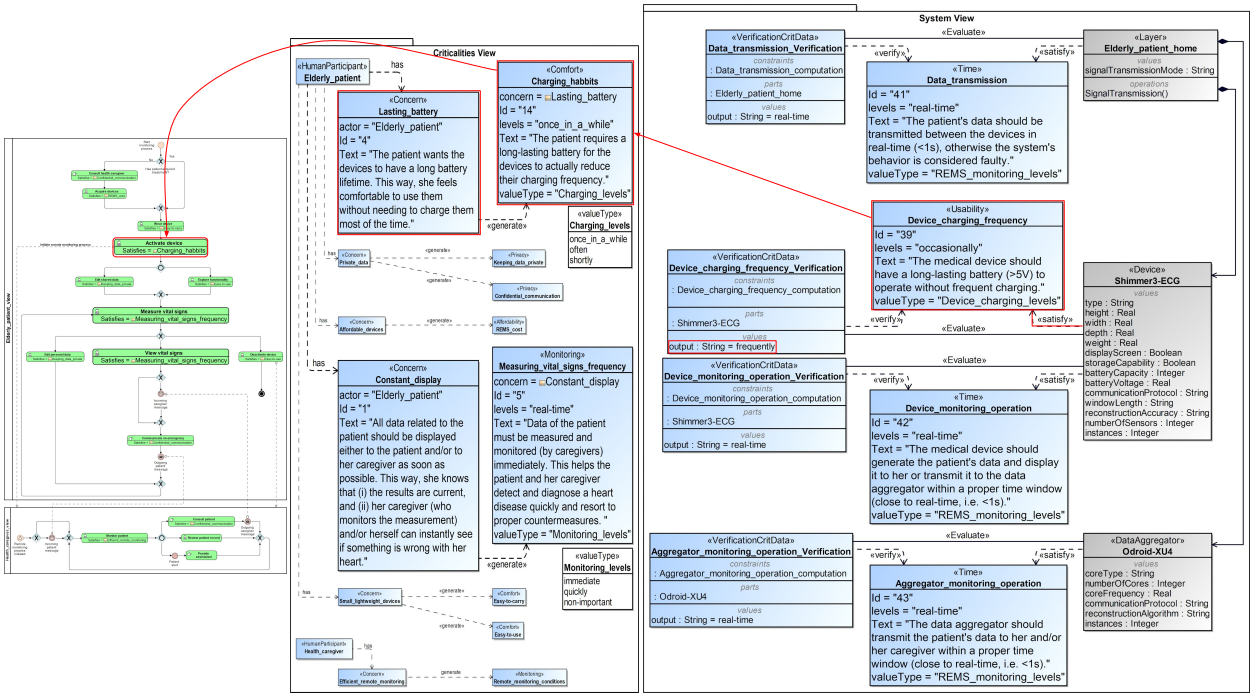


FIGURE 5.13: Evaluating criticalities across views.

Specifically, the charging frequency criticality was defined with an "occasionally" desired level. After the system configuration and evaluation, the obtained/computed criticality level, here "frequently", does not verify the desired one, thus, the frame of the "defective" entity becomes red-colored. In the system view, the designer is notified visually that there is a problem; the computed output and the "satisfy" connection between criticality and component also become red-colored. In the criticalities view, the respective human criticality, as well as its concern alert both the patient and the designer about the problem. Finally, the activity that satisfies the human criticality becomes red-colored and informs the patient.

Note that, in case there is a problem, the environment shows relevant information to the designer, as well as adjustments that can be made. The latter can be suggested actions, like choosing

TABLE 5.1: Evaluating the trade-off between performance and cost levels in REMS home.

CapEx cost (€)	Cost limit (€)	Cost level	QoS level			Accepted solution
			Real-time operation		Battery lifetime	
			Device	Data aggregator	Device	
631	500-800	low-cost	best effort	best effort	occasionally	×
631	500-800	low-cost	best effort	best effort	frequently	×
756	500-800	low-cost	best effort	real-time	frequently	×
806	800-1200	high-cost	real-time	real-time	shortly	×

another configuration, regarding the improvement of the device's charging frequency.

In addition, the monitoring operation criticality of the device is real-time. The parametrically computed output returned the same value, based on the device's specific properties. Thus, this criticality is verified against the desired one.

5.5.6 Results on configuring REMS home equipment

Table 5.1 contains alternative REMS home solutions w.r.t. the levels of requirements which are imposed by a REMS designer, corresponding to the REMS home configuration, presented in Figure 5.9. The *Real-time operation* and the *Battery lifetime* requirements are considered as performance parameters, according to Wymore [293]. Thus, the trade-off between performance and cost for the REMS home configuration (see Figure 5.9) is depicted in the table. All of the alternatives provide valid system (i.e. appropriate) outputs, according to [293]. However, the REMS designer is interested in the acceptable ones, i.e. those that satisfy both performance and cost restrictions. That is, the operation of the system in "real-time" should be supported with a "low-cost" solution, where the battery of the device (e.g., sensor) should be charged "occasionally" for an increased lifetime.

In the context of REMS, the primary design goal is to meet the concerns of a patient (like real-time operation), while the cost restriction is the budget of the patient for the acquisition of the REMS home equipment (devices). The acceptability of a design solution on both fronts is explicitly stated in the last column of Table 5.1 (i.e. "Accepted solution" column).

As shown in this table, there is not an acceptable solution, meaning that there is not a REMS design output under the technology that is employed in the EMBIoT project, that can satisfy all three patient requirements. Compatible alternative matching for them is depicted in the table, though non-satisfied patient requirements are set in Figure 5.9.

As derived from the analysis results, the primary CapEx driver of the REMS is the real-time operation requirement, since the battery behavior of the device does not have an impact on cost (see first two rows of Table 5.1). In case the elderly patient requires a "low-cost" CapEx level, he/she should be willing to compromise on the real-time operation of the medical units (which should measure, process, and transmit medical data as quickly as feasible) or the battery lifetime of the device (i.e. its charging frequency which should be at least once every other day). Specifically, according to the table, there are cases where the budgetary constraints of the patient are covered (i.e. CapEx cost reaches "631" Euro or "756" Euro that correspond to "low-cost" levels), while from an operational quality perspective, the device does not operate in "real-time" (i.e. it makes a "best effort"; second and third row of table). In case all requirements are "real-time", the capital expenses necessarily reach a "high-cost" level (over "800" Euro) that may not be desired by the patient, since it surpasses his/her budgetary constraints (fourth row of Table 5.1)). Moreover, the

latter solution requires very frequent charging of the device since its batteries are drained quickly. Therefore in these scenarios, acceptable solutions can only exist upon the patient's compromise. If "real-time" operation is a prerequisite for the condition of the patient, the cost is high and batteries should be charged shortly. To meet "low-cost" and "occasionally" battery charging, the condition of the patient should allow "best-effort" data transmission and monitoring. This is true in most of the cases in the EMBIoT case study.

In the case of REMS, performance and cost constraints set by the patient are not satisfied. Patients have to choose a "lower" QoS solution to obtain a "low-cost" solution. Real-time operation is in fact the key indicator in this case. To obtain real-time operation, the patient must pay for a "high-cost" solution. This case is indicative of the fact that finding the right balance between cost and performance is not trivial, even in a simple REMS scenario with two devices. Patients may be inclined to pay more to address their primary concern, such as real-time operation. However, they have to decide whether they really need it or not, based on their condition and their doctor's advice.

5.5.7 Results on stress-testing a fall-detection system

We demonstrate the feasibility and applicability of the proposed model-based framework in the design of a fall detection IoMT-based system, where elderly patients reside and may operate appropriate medical equipment in the comfort of their home; we focus on the fall detection case, where the position and stance (sitting, lying, etc.) of elderly individuals should be recorded and monitored via specific medical sensor device(s) [151]. Should a patient fall, the system alerts remote healthcare providers to act.

In an example Home scenario, a post-surgery elderly patient with hart failure can live independently at his residence and recover, although frequent monitoring of his position and stance (sitting, standing, falling) is mandatory. For this purpose, an IoMT-based system was designed and implemented by Hamad Medical Corporation and the College of Engineering at Doha University as part of EMBIoT project [11], in Qatar. The system installed at patients' homes consists of the following components: a) a fall detection sensor, suitable for recording the position and acceleration of the patient's body and b) a data aggregator that is used to collect sensor-generated data and transmit it to a remote healthcare facility (e.g., a hospital) for monitoring and assessment. After acquiring these devices, the patient can start the monitoring process and receive remote healthcare services.

The application of the proposed framework had two purposes: (a) to enable the researchers in the College of Engineering at Doha University to provide an implementation of the system that matches patient concerns and test its performance accordingly and (b) to assist the IoMT system designers employed in Hamad Medical Corporation, to configure an acceptable solution, properly initializing EMBIoT equipment for patients, taking into account their specific concerns.

The overall process is depicted in Figure 5.14. Note that each step as well as the frame of the elements that are associated to this step have the same distinct color, while steps are shown in the legend of the figure.

Define system structure & human criticalities. The Hamad Medical Corporation in collaboration with the College of Engineering, Doha University, decided to employ the *Shimmer3-IMU* [30] as the medical sensor device that the patient wears for the detection of occurrence and severity of sudden falls. The *Odroid-XU4* [115] was the data aggregator chosen to collect all sensor-generated data and transmit it to the remote healthcare facility. Both devices were employed in similar applications, however they should be tested and properly configured to satisfy fall detection requirements.

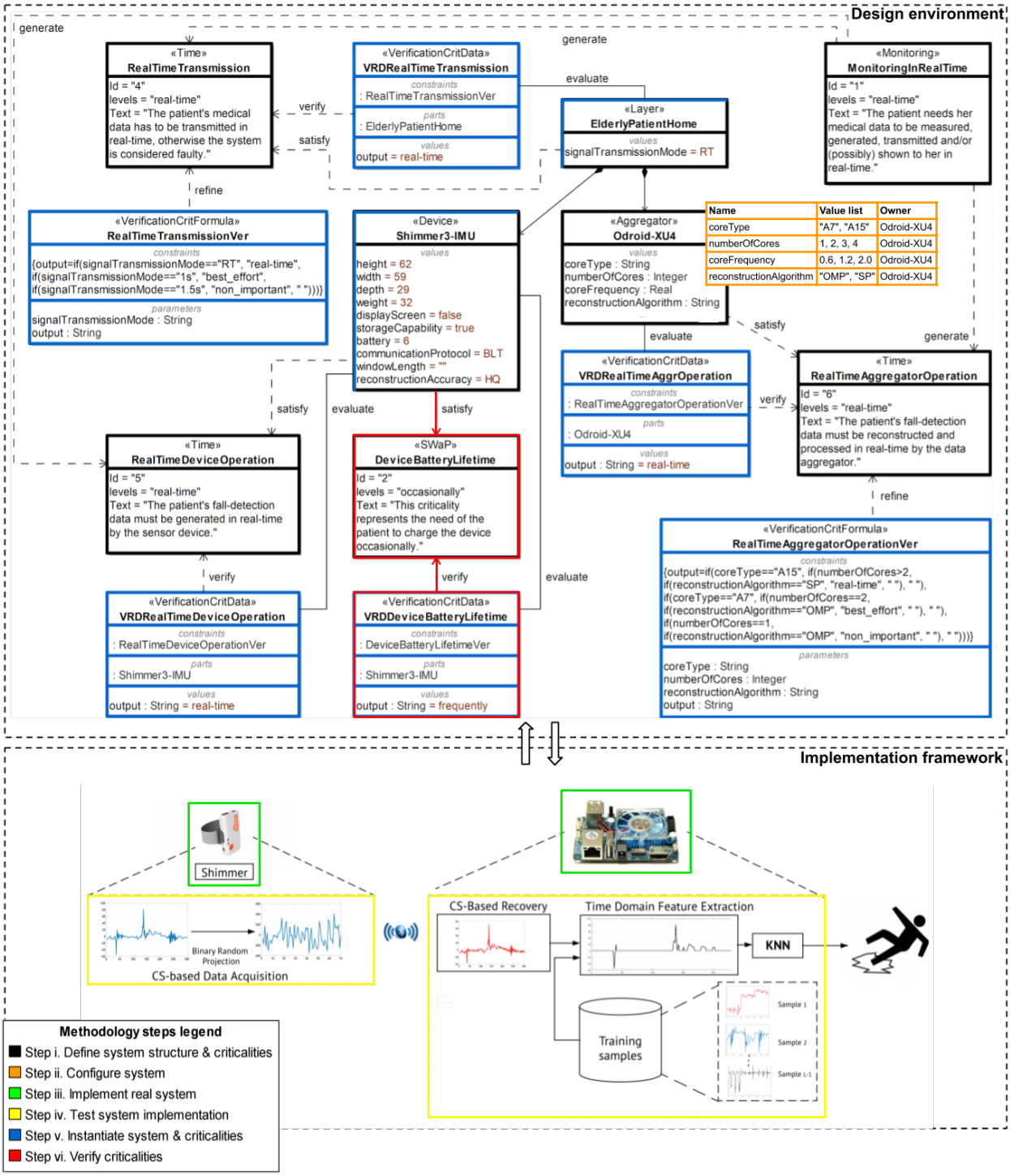


FIGURE 5.14: Model-based framework applied to Home fall-detection scenario.

A two-layer model represents the patient home, *ElderlyPatientHome* composed of the device(s) and the aggregator, forming a structural hierarchy (see black shapes in Figure 3). Each components holds specific properties, describing it. They are defined in the next step.

System components are associated to human criticalities, representing patient concerns, often in conflict to each other. Their are described in a textual form and graded in levels, to be easily described by patients. Indicatively, in Figure 5.14, two conflicting patient concerns are explored. The first one, *MonitoringInRealTime*, is of *monitoring* type and depicts the need to realize possible fall with no delay. It is grated in three levels: "real time", "best effort" and "non important". Depending on the patient condition, a small delay (e.g. up to one minute) could be tolerable, thus "best effort" may be selected. This concern related to IoMT system as a whole, thus is associated to the Home layer. The second criticality is *SWaP* (i.e. Size, Weight and Power) of *comfort* type that restricts the size and energy consumption of the wearable device, in this case *Shimmer3*. Comfortable wearables lead to a greater willingness to adopt and use a remote monitoring system. *Monitoring* concerns automatically generate (indicated as arrows in Figure 5.14) *Time* restrictions associated to all system components. In practice, they indicate the time frame all the system components should operate in. Regarding Time criticalities, the Elderly Patient Home is connected to the *Real Time Transmission* criticality that describes the requirement of all components to communicate in real-time. The Odroid Aggregator is associated with the *Real Time Aggregator Operation* describing that real-time collection and processing of sensor data is required in order to decide whether the patient has fallen. The Shimmer Device satisfies the *Real Time Device Operation* criticality that points out the need for real-time generation and transmission of fall detection data. All these requirements are described in a qualitative fashion using graded levels, by the patient with the assistance of IoMT system designer. They are further analyzed in a quantitative fashion by the system designer with the assistance of the system constructor (e.g. Doha University researchers) in Step V (blue-colored shapes is Figure 5.14).

Configure the Home IoMT System. Fall detection system designer identifies system component properties, relevant to system design (e.g. those that their values reflect on the setup of the system) and constructs values lists in collaboration with the system constructor. For example the Odroid data aggregator property value list is depicted in Figure 5.14, as an orange-color-framed rectangle (Step II of the framework). According to this list, the *core type* property of the Odroid aggregator may obtain the "A7" (i.e. low-powered, medium-performance CPU core) or the "A15" (i.e. high-power, maximum-performance CPU core).

These values are restricted by the device characteristics, as for example for *core type* property, while they may also be restricted by the system constructor implementation decisions, as for example for *reconstruction algorithm* property. As described in [54], the reconstruction algorithm of the compressed signal received by the sensor plays an important role on the aggregator performance when deciding when a fall occurred or not. In this case two well-known greedy algorithms, the orthogonal matching pursuit (OMP) [272] and subspace pursuit (SP) [49] were selected for reconstructing the compressed acceleration data. Greedy algorithms are widely considered for real-world applications as they exhibit a simple implementation architecture, a fast convergence to the solution and a sub-optimal recovery.

Implement & test real system. Following its design, the actual system is composed, as depicted in Figure 5.14 with a green-colored frame. The system constructor is now aware of all the properties of the system that affect design decisions. They are defined in Step II. Thus, their may test the behavior of actual system in a controlled environment, to explore how these properties affect each other and the graded levels of related criticalities, with the collaboration of the system designer.

Different sapling rates may be used in the sensor to identify the position of the body, while data is compressed to be transmitted to Odroid-X14 aggregator, where the signal is recovered and based on training samples, the aggregator decides whether a fall may have occurred. Implementation details are provided in [54]. One of the primary performance metric explored while testing the operation of Odroid-X14 aggregator was execution time needed to make a decision on whether a fall occurred. The execution time metric directly related to the estimation of real time operation criticality level for the aggregator and is related to all the properties of the aggregator, indentified in Step II (see the orange-colored frame is Figure 3). As agreed between Hamad Medical Corporation (system designer) and Doha University (system constructor), execution time less than 0.02 secs ensures real time operation (real time level of the *RealTimeAggregatorOperation* criticality). The core type, number of cores, core frequency and data reconstruction algorithm affect execution time. Thus, different combinations of their values should be explored during testing. Testing results are presented in Figures 5.15 and 5.16, indicating the execution time using "A7" cores and "A15" cores, respectively. The execution time has been estimated for different combinations, and based on that, the three *RealTimeAggregatorOperation* criticality levels, namely "real-time" (execution time less than 0.02 sec), "best=effort" (execution time less than 0.2 sec) and "non-important" are extracted. One should note that the "A7" core may not perform in real time. The "A15" may perform in real time using more that two cores in any frequency when SP reconstruction algorithm is used, while using OMP this may be performed using more than 1.2 GHz frequency.

Testing the actual system against patient criticality requirements prior its installation, eliminates system configurations that do not guaranty the efficient operation of the system. This enables the system constructor to provide solution that may be more acceptable by patients, since the system is configured and tested taking patient concerns into account as well.

Instantiate system model & criticality verification data. In this step, the blue-color-framed blocks, shown in Figure 5.14, representing instantiated system elements, obtain specific values after system testing. For example, the Shimmer device holds distinct values regarding size or data generation and transmission properties (e.g., $reconstruction\ accuracy = "High\ Quality(HQ)"$) (see Figure 5.14).

According to our framework, each defined human criticality, representing patient concerns, is associated with a respective verification criticality formula, describing criticalities in a quantitative fashion, and a verification data entity, containing data extracted by the actual system testing to verify the criticalities. For example, the *Real Time Aggregator Operation* criticality is linked to the *Real Time Aggregator Operation Verification* formula and is verified using data stored in *VRD Real Time Aggregator Operation* verification data associated with Odroid-XU4 aggregator (see Figure 3). These are also represented as blue-color-framed blocks in Figure 5.14.

The *Real Time Aggregator Operation Verification* formula is constructed by the system designer to calculate *Real Time Aggregator Operation* criticality level. The formulas provide a quantitative description of the human criticality, associating Odroid-XU4 properties based on the results of system testing provided by system constructor. *Real Time Aggregator Operation Verification* formula is based on the data presented in Figures 5.15 and 5.16. The criticality computation results in "real-time" level value, based on the values of Odroid-XU4 properties, which are retrieved by the actual system, and is stored in *VRD Real Time Aggregator Operation* verification data entity.

Verify patient criticalities. In Figure 5.14, the *Device Battery Lifetime* human criticality, representing comfort patient concern, whose desired level is set to "occasionally", is not verified by the proposed configuration. Thus, it is annotated by red color. This indicates the need of the patient to charge the device once in a while. The related verification data element holds the computed

"frequently" level. When these values are compared, the criticality is not verified. In addition, the verification data and the connection between the criticality and the component are also annotated, depicting that the instantiated component cannot satisfy its criticality. The monitoring patient concern, represented as *Monitoring Real Time* human criticality related to the whole system is satisfied, thus not annotated.

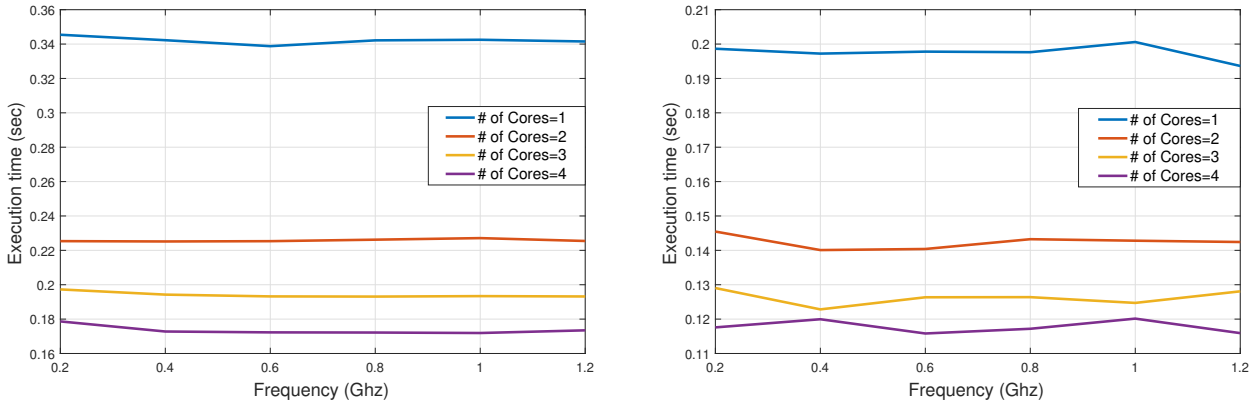
It is worth mentioning that in case there is an inconsistency (annotated criticality), the modeling environment alerts the system designer, exhibiting appropriate information. The patient may be informed that not all concerns can be satisfied by the available system configurations and decide with the assistance of the system designer which of them should be relaxed. In this case, real time monitoring needs frequently battery charging.

Although patient's concerns may be recorded as part of the system requirements, aka criticalities, (steps I and II of the proposed framework), the tricky part is to find a way to associate them with the characteristics (structural and behavioral) of the system implementation (steps III and IV). The criticality verification formulas defined by the system designer (step V) relay on the combination of specific system properties and the system behavior under different conditions. The latter may only be obtained as the result of complex system testing. Thus, the system designer may only obtain this information in collaboration with the system constructor.

In most cases, system constructors study their system behavior (for example system performance, security or energy consumption) under different condition and provide metrics important to them, not directly linked to patient concerns. The system designer should be able to relate these metrics to specific patient concerns depicted in an abstract manner, as criticality levels. To do this, system designers construct criticality verification formulas. However, their are restricted to the metrics available by the system constructors. If the patient concerns are incorporated in the system requirements, even in a abstract, graded fashion, the system constructor may take them into account, while testing the system, and, thus, provide the system designer all the necessary metrics.

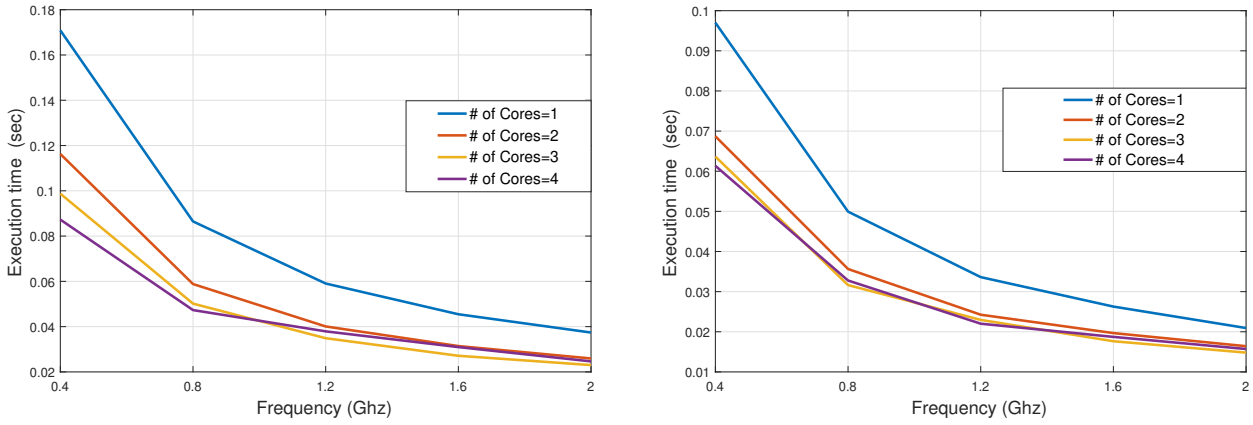
Take the *MonitoringRealTime* criticality for example. The patient wants to be monitored in real-time, no delays, however small are tolerable. The question for the system designer is under which conditions and employed which configurations this is possible. Thus, the designer records restrictions on all the devices operating in the patient's home. For example, the *RealTimeAggregatorOperation* criticality imposed to Odroid data aggregator component, indicating that it should detect possible fall detection real-time. However, this requirement may mean little to the system constructor. How fast is real-time? Thus, time limitations should be explored, while testing the system. Although the designer may realize that the *RealTimeAggregatorOperation* criticality verification may relay on the combination of the aggregator properties, the output value may not be determined unless the real system is tested combining these properties to determine the execution time. Having this in mind, the Qatar University team tested the Odroid-XU4 compoment, using several combinations of processors, number of cores, their respective frequencies and reconstruction algorithms. The corresponding results are presented in Figures 5.15, 5.16, assessed by averaging the experiment over "100" trails.

The following observation can be deduced from the obtained results. (i) The "A15" outperform the "A7" cores regardless of the frequency and number of cores used. (ii) For the "A7" cores, the core frequency does not play an important role in the acceleration of the computation process. In fact, the number of cores govern entirely the performance in terms of execution time. (iii) The "A15" cores exhibit a faster performance. Moreover, both "SP" and "OMP" algorithm respond well to the increase of frequency and the number of cores. In general, for a particular CPU (core) frequency, when the number of cores is doubled, the execution time is halved. For instance, using "0.4" GHz and doubling the number of "A15" cores ("1" $\rightarrow$ "2" $\rightarrow$ "4") reduces the execution time ("0.17" sec $\rightarrow$ "0.119" sec $\rightarrow$ "0.088" sec) and ("0.97" sec $\rightarrow$ "0.68" sec $\rightarrow$ "0.061" sec) for



(A) OMP (B) SP

FIGURE 5.15: Execution time using the "A7" cores based on number of cores and operating frequency.



(A) OMP (B) SP

FIGURE 5.16: Execution time using the "A15" cores based on number of cores and operating frequency.

"OMP" and "SP", respectively. (iv) Regardless of the number, type and operating frequency of cores, the processing time is always below the real-time window of "2" seconds. Therefore, more analysis can be further applied to the data without compromising the real-time critical metrics. Finally, it may be deducted that the optimal configuration for the data aggregator to operate in real-time, contains the "A15" core type, "4" cores and "2" GHz frequency, using the "SP" as a suitable algorithm for collected data reconstruction and compression.

Not all of this information is depicted by system designer, when constructing the corresponding condition of the *RealTimeAggregatorOperation* verification formula. Designers utilize the information necessary to validate specific configuration of the system, made available to patients (see for example that not all possible frequencies are made available when instantiating the system).

5.5.8 Framework evaluation by REMS participants

In previous sections, we applied the proposed design framework in the REMS home case study for a single-patient use case, and we described all the steps of the design process. In this section, we summarize useful cumulative insights derived from several REMS patients that participated in the system design by going through the design process.

Specifically, in the context of a research project, we cooperated with a hospital in our area that wanted to apply a system like the REMS and monitor the medical condition of elderly patients while they are at their own home. In cooperation with the hospital, we found elderly patients/volunteers that were informed about systems like REMS and were willing to use and operate such a system. 31 human individuals were potential REMS buyers and users that required a medical system for the monitoring of their health condition. 58% of the individuals were male, while the rest were female. In addition, their age ranged from 60 to 80 years; therefore they belong to the target group of potential patients.

We were assigned the role of REMS designer and went through the proposed comprehensive design process. Our experience shows that the steps of this process can be done in a straightforward fashion, starting from the definition of the human behavior and ending to the evaluation of the system design; this is achieved while the designer continues to work in a single design/modeling environment.

Each patient cooperated with the designer (i.e., us), providing concerns and criticalities that should be fulfilled by system design. In cooperation with the designer, the patients followed the whole design process, and in the end, they filled out a small survey regarding their experience.

Patients were able to provide their raw concerns and criticalities. The concerns were categorized into four distinct identified criticalities types that cover the spectrum of CPHS like REMS, i.e."monitoring", "privacy", "comfort", and "affordability". For each criticality class, the patients expressed its importance by choosing a desired level. This is presented in Figure 5.17.

Nearly 66% declared that the monitoring of their medical data must be done in "real-time". We assume that this choice corresponds to the need for immediate action as soon as an anomaly w.r.t. vital signs, is detected. 60% of the patients needed a "strong" privacy/security of their personal medical data, while 70% wanted to charge their devices "once in a while". The first choice (i.e. "strong" privacy) corresponds to the sensitive nature of medical data, which is actually clear to the patients. The second one relates to the requirement of portability and comfort; patients want to charge the device occasionally without the need to keep their mind on the battery while doing everyday tasks. The interesting fact is that 60% of the patients required an overall "low-cost" REMS, while the rest could afford a "high-cost" or even "expensive" REMS, considering that a more costly system provides higher quality devices and services. Note that there was not any patient-expressed concern that could not be categorized in a criticality type; the classification is fully inclusive.

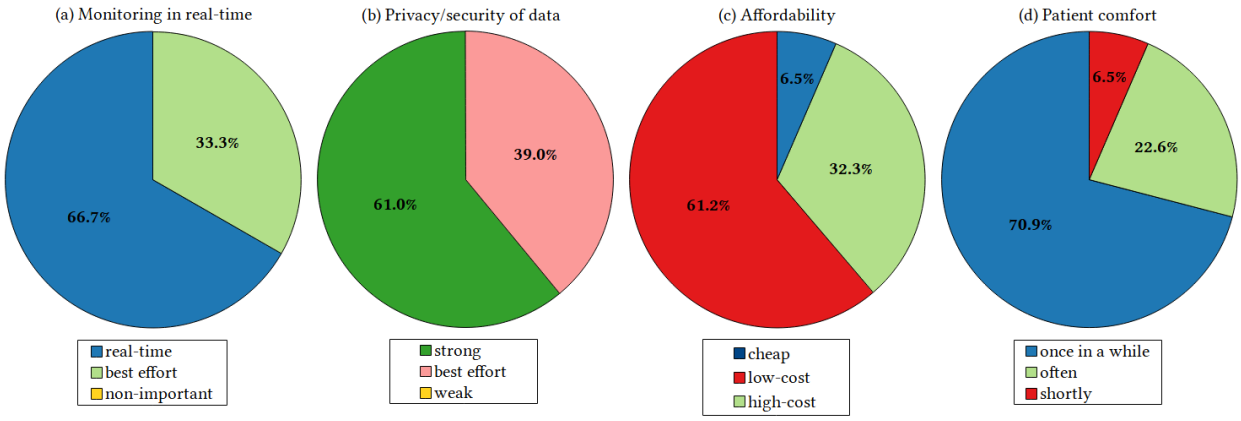


FIGURE 5.17: Preferred levels per criticality type

After concerns and criticalities are provided as input, the designer proceeds with the rest of the framework steps (human/system criticality derivation, system configuration) until reaching the point of criticality evaluation. At this stage, designers and patients observe that the system is configured in a way that covers some criticalities of the patients, while not fulfilling others; not all criticalities may be addressable at the same time. For example, a patient needs the monitoring of her data in real-time. In parallel, she wants to charge the devices that monitor her health condition only once in a while. The REMS is configured in such a way that achieves the first, however, a real-time operation quickly drains the battery of the devices. This demonstrates that certain criticalities may even be in direct conflict with each other e.g., due to physical constraints (real-time needs more battery). In those cases, it falls on the patient to choose which criticality to "sacrifice" (by relaxing the associated desired levels) in order to achieve a feasible/practical REMS design solution.

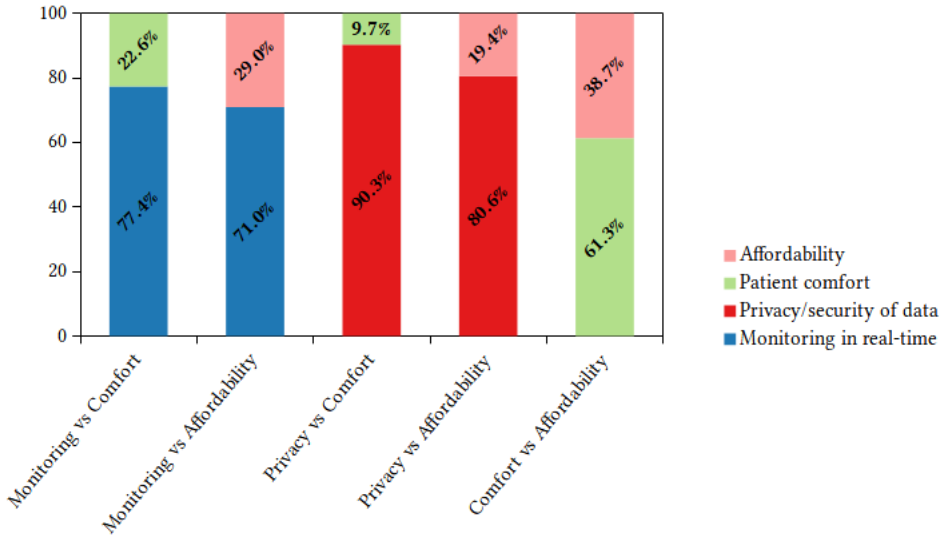


FIGURE 5.18: Conflicts between criticality types.

By examining different pairwise combinations, we isolated cases where one criticality (of a certain type) was not satisfied, while another (of a different type) was addressed by the system. Via this process, we identified the following conflicting pairs: monitoring and comfort, monitoring and affordability, privacy and comfort, privacy and affordability, and comfort and affordability. Note that there is no conflict between monitoring and privacy; this is due to the attributes of the system components which allow real-time operation using strong encryption and security.

Having identified these conflicts pertaining to the system operation, we asked the patients to decide what is more critical for them in each case. Based on Figure 5.18, an important observation is that patients were willing to give up comfort and affordability for real-time monitoring of their data (77% of the patients chose real-time monitoring over comfort; a similar trend applies to 71% of the patients who chose real-time monitoring over affordability) and high privacy/security (90% considered privacy more important than comfort, and 80% chose privacy over affordability). In addition, between comfort and affordability, 60% needed an economical solution rather than the convenience of small-size and/or light-weight devices. Thus, a secure, high-performance system that remains at an economical level for purchase is the REMS solution that most of the patients wanted. Therefore, REMS models were designed for the patients along these axes.

Finally, after the design process yielded the required results, producing REMS models that fulfill the patients' needs, we provided the patients with a short survey regarding the following: (i) putting in order of importance the four identified criticality types, and (ii) providing general feedback regarding their experience.

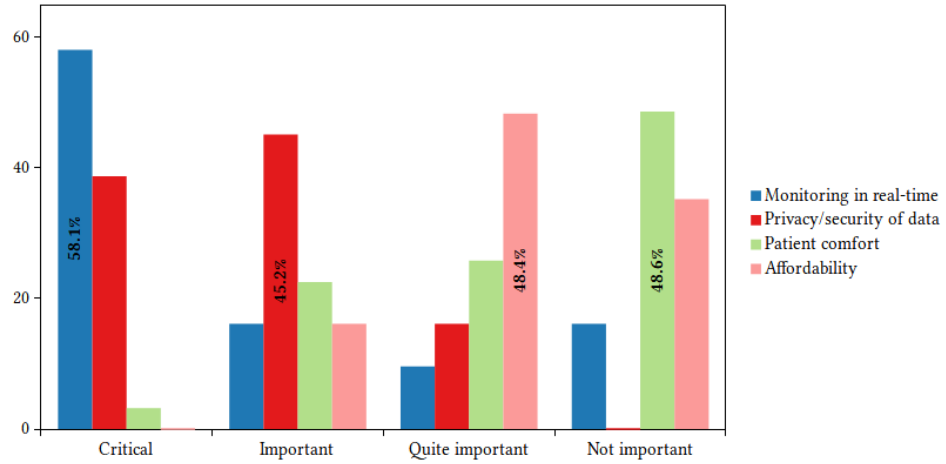


FIGURE 5.19: Order of importance of criticality types.

In Figure 5.19 we present the percentages of patients that classified the different criticality types based on their importance. In order to derive cumulative insights on this raw classification, we first assign a grade to each order of importance to treat them numerically:

(i) $order\_grade(Critical) = 4$,

(ii) $order\_grade(Important) = 3$,

(iii) $order\_grade(Quite\ important) = 2$,

(iv) $order\_grade(Not\ important) = 1$.

We then compute the grade of a type as a simple weighted average using percentages as weights:
$type\_grade(type) = \sum_{order} percentage(order, type) * order\_grade(order)$. This results in table 5.2.

TABLE 5.2: Calculated grades of criticality types.

Criticality type	Grade
Monitoring in real-time	3.18
Privacy/security of data	3.17
Patient comfort	1.81
Affordability	1.80

We observe that real-time monitoring is the most important criticality type. This is justifiable since monitoring is related to the core functionality of the REMS. Thus, the major need of the patients is their health to be monitored continuously (i.e. in real-time). In addition, patients place the privacy of their data a bit lower than real-time monitoring but much higher than the cost of the REMS or the comfort of the REMS devices. Comfort is comparable to affordability, albeit achieving a higher overall preference.

Regarding the general experience feedback, 80% of the patients declared that their cooperation with the designer, as well as their participation in the system personalization and design, was very important (see Figure 5.20); it affected their decision to actually use and operate a personalized REMS. In parallel, the remaining 20% would use a REMS regardless of their participation in such a design process. Thus, we observe that a critical take-away is that personalization –not only post-purchase but during the actual design— supersedes the need for buying generic off-the-shelf systems. This was justified in the field as follows: since the patients' view (i.e. concerns, needs, etc.) was taken into consideration during system design, patients could actually choose a feasible and suitable REMS solution for them. In turn, they could acquaint themselves much better with such systems, efficiently using and operating them.

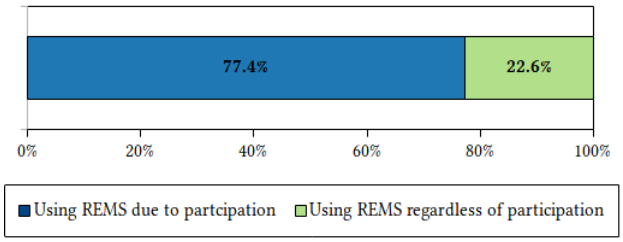


FIGURE 5.20: Impact of participation in REMS design.

5.6 Discussion

In the context of the REMS case study, we employed three aspects of our framework. First, we applied our QoS profile containing QoS-related concepts such as graded quantitative requirements to define and manage human and system criticalities. Second, we applied our Cost profile to define and manage cost criticalities and concerns of the human user. Finally, to assist the decision-making process of the designer, a logical solver was integrated in the system model and was seamlessly used to evaluate different system design solutions.

With this application, we identified the following critical contributions. First, the framework provides an efficient mechanism to specify and manage hierarchies of inter-dependent criticalities. Second, it enables the generation of human criticalities from human concerns. Consequently, it facilitates the transformation of human criticalities into system criticalities in a semi-automated

fashion; the designer follows a guided process to define criticalities, assisted by automated mechanisms. Third, the Cost profile acts as an auxiliary profile, ascertaining that the design solution can also fulfill the budgetary requirements of the user; from the user's perspective cost is yet another criticality that needs to be addressed. Fourth, the entire design process took place in a single environment, within which the logical solver (assisting the decision-making process) was integrated; this facilitated the designer to directly make decisions and explore design solutions without dealing with the underlying complexity.

In summary, applying the framework (and its profiles) in the CPHS context we identified multiple views (human, system, criticalities) which assisted the involvement of the user in the design process; as shown in the evaluation (see Section 5.5.8) users were in favor of this framework.

Chapter 6

Conclusions

> "Everything that has a beginning has an ending. Make your peace with that and all will be well."
>
> Jack Kornfield, Buddha's Little Instruction Book

In Section 6.1, we state our concrete research contributions. In Section 6.2, we provide an overview of relevant future work.

6.1 Review of research contributions

We classify our contributions in three categories, described in the following: domain-specific and application-oriented (Section 6.1.1), general and MBSD-oriented (Section 6.1.2), and standardization (SysML)-oriented (Section 6.1.3).

6.1.1 On the application of the framework on specific domains

With respect to the specific application domains (RTS in Section 4 and CPHS in Section 5), the following contributions were identified.

In the context of RTS, the target beneficiaries were the system operators. We employed a modeling environment where QoS and cost analyses are integrated into a single, central RTS model. This integration was an original contribution in the context of RTS, since nothing similar had been used in the past by operators. We facilitated operators to make informed operational decisions QoS- and cost-wise; the operators can assess directly the effects of such decisions on the RTS. It is also shown that the cost meta-model is extremely useful in the particular domain.

In the context of CPHS we developed a comprehensive multi-view framework to (i) identify the concerns of human participants as criticalities, which are in turn transformed into system design requirements that affect the system's structure and functionality, and (ii) provide automated decision-making capabilities to ensure that the final design solution conforms to human concerns. At the end of the process, human participants can be informed about the verification of their concerns without having any low-level knowledge of the design process itself, as well as its constraints. This is achieved via a view-based interaction; the view is specifically designed for them. Corresponding CPHS profiles were employed and evaluated in two distinct applications in the REMS case study. First, we configured REMS home equipment helping the participants understand trade-offs between the concerns that they themselves rated. This was achieved via the introduction of the criticality concept which represented QoS. Second, we stress-tested a fall detection REMS scenario, where we helped the designer by ensuring that the design was complete w.r.t. human concerns; i.e. no concerns were accidentally omitted during system testing.

6.1.2 On the contribution of framework in MBSD

Generalizing our contributions beyond specific domains, we observe that by using the same QoS/-cost profiles, concepts, and toolsets, we can apply our framework to different systems with distinct challenges and objectives; specializations can employ additional domain-specific profiles that are variations of the same principal concepts. In summary, the framework is:

- *generic*, since it proved to be applicable to various SoS (even in cases where the QoS requirement hierarchy was complex e.g., CPHS) by employing cross-domain concepts such as graded quantitative requirements.
- *valid*, since it was verified with real data and requirements in multiple domains; each stakeholder (system operators in RTS, human participants in CPHS) achieved their objectives and made required decisions w.r.t. the system.
- *complete*, since it integrates the entire MBSD process, starting from structure and requirement definition, going through analysis and requirement verification, and ending to decision-making that is incorporated back into design. The latter "closes the loop" and improves the system as a whole.

Delving more in the details of the contributions of the framework, we extended Friedenthal's framework by integrating QoS and cost meta-models into the system model. We introduced a generic way to transform system models into decision-making models to facilitate decision-making in SysML. We further integrated analysis tools, which are based either on simulations or mathematical solvers, into the system model. It is worth noting that the concept of the analysis model, as well as the way we manage it, will be adopted by the new SysML 2 standard. Finally, we identified the need to feed the results of the analysis and decision-making model back into the system model for further use (such as system evaluation). We further note that the proposed framework can be seamlessly integrated with SYSMOD since all related concepts are compatible and complementary to SYSMOD.

6.1.3 On current standardization efforts

Our work contributes to current standardization efforts. We identify two contributions on this front: (i) compatibility with UAF, and (ii) ideas for SysML 2.

Compatibility with UAF

Our QoS-oriented system design relates to the architecture structure that corresponds to one of UAF's grid cells (see Figure 2.8) named resource structure (in Resources row). However, to enable this design to serve QoS, the latter should be represented by resource constraints (in the same row, Constraints column) that are directly associated to the resource structure. We employ specific profiles to describe the resource structure and we propose meta-models to describe QoS/cost constraints that fit the resource constraints. On a domain-specific basis, we differentiate these constraints (in the Constraints column) as follows. In the RTS, we design based on operational constraints. In the CPHS, we design based on personnel constraints, such as human drivers, which are described via criticalities. In essence, we propose basic, generic meta-models that describe QoS and cost in the form of resource constraints and we enrich them via specializations/extensions to describe operational constraints and human drivers, depending on the domain. In light of this, our framework is fully compatible with UAF, thus system designers that already use UAF can employ our meta-model(s) for QoS/cost management, adjusting the Resources row (resource structure) and introducing QoS/cost specializations on the Constraints column.

Ideas for SysML 2 new standard

Finally, our research improves and benefits from the wider SysML landscape. As a standard, SysML transited through various versions (from 1.3 to 1.6) during the course of our work. Initially, we faced language limitations when endeavoring to describe requirements or expressions without quantitative parameters. During version updates, some aspects were improved. For example, the fact that we were able to represent requirements and integrate and compute functions through a mathematical solver (SysML parametrics) in later versions was useful for our work. Moreover, one of our principal SysML-oriented contributions was keeping the results of the analysis model within a distinct (i.e. not the same as the associated requirement) entity so that they can be easily managed; for example, changes can be recorded. This in turn eases the integration of the analysis results back into the initial model. In principle, maintaining such distinct entities that are correlated with each requirement makes sense modeling-wise. While SysML has adopted the concept of an analysis model that the designer employs and specializes to analyze a system, it does not currently support the integration of analysis results back into the system model in a consistent and clear fashion (even conceptually). Encapsulating them in the requirement itself would be one direction to explore; the latter could be a part of ongoing work on SysML 2.

We make the software (plugins) and models implementing our research and framework publicly available as open source [139]. Doing that, we hope to help other designers use the generic profiles and accompanying plugins in order to design and analyze their own systems and pursue their own research directions in QoS-aware MBSD using SysML.

6.2 Future work

Having the concrete framework of this dissertation at hand, we can extend and build upon it to enable the following research directions.

Exploring SysML 2 standard. One direction of future work is the investigation of (i) how the analysis model(s) can be integrated efficiently in new SysML versions (like SysML 2), and (ii) how process automation (including analysis, evaluation and decision-making) can be better supported in such versions natively. Analysis results could be fed back to the model in entities that record them. Moreover, one relevant line of future work would be employing new expressions that render parametric diagrams obsolete, since expressions will be directly executable in SysML.

Green computing and ethical concerns. Moreover, in this thesis we have solidified the notion that QoS is a prominent dimension of system design. In a related context, IEEE has revised the system design standard IEEE 7000-2021 Standard Model Process for Addressing Ethical Concerns [111, 282] to take into account ethical issues and concerns such as privacy, morality and environmental impact. Drawing inspiration from this domain, and acknowledging the need for environmentally-friendly ("green") systems, we would like to manage the concern for the environmental footprint of such systems and their provided services. In practice, we plan to extend our framework to manage such concerns in a manner similar to QoS management. Specifically, the designer, using the aforementioned IEEE standard, can develop a SysML meta-model containing non-functional requirements, similar to QoS. These requirements will be specialized for the description and representation of environmental footprint in the context of green computing. The rest of the non-domain-specific framework can be used as-is.

QoS-aware Cloud engineering. Another aspect of our future work entails the application of QoS-awareness in cloud services. Specifically, the cost profile can be applied in different domains, such as cloud services, enabling providers and customers to assess provided solutions and infrastructures from a cost perspective as well. In conjunction with the green computing direction, we will study such systems as a combination of: (i) technology, (ii) financial aspects, and (iii) environmental footprint since all three dimensions affect and depend on each other. Following our framework, such dimensions can be integrated in a single system model. To this end, we will expand our analysis to more fine-grained cost categories, and provide domain-specific cost computation functions. We further plan to extend our SysML cost profile to support revenue and profit analysis as well. Corresponding properties and functions may be supported and automatically computed in a similar way as cost ones. Furthermore, the integration of external tools, facilitating financial analysis, with SysML modeling frameworks shall also be explored.

Bibliography

[1] Juul Achten and Asker E Jeukendrup. "Heart rate monitoring". In: *Sports medicine* 33.7 (2003), pp. 517–538.

[2] J. I. Aizpurua et al. "A Model-Based Hybrid Approach for Circuit Breaker Prognostics Encompassing Dynamic Reliability and Uncertainty". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.9 (Sept. 2018), pp. 1637–1648. ISSN: 2168-2216.

[3] Joan E van Aken. "Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules". In: *Journal of management studies* 41.2 (2004), pp. 219–246.

[4] Noura Alhakbani et al. "An efficient event matching system for semantic smart data in the Internet of Things (IoT) environment". In: *Future Generation Computer Systems* 95 (2019), pp. 163–174.

[5] Sinan Si Alhir. *Guide to Applying the UML*. Springer Science & Business Media, 2006.

[6] Mohamed Hafez Fahmy Aly, Hassan Hemeda, and Mohamed Abdelnaby El-sayed. "Computer applications in railway operation". In: *Alexandria Engineering Journal* 55.2 (2016), pp. 1573–1580.

[7] David Ameller et al. "Dealing with Non-Functional Requirements in Model-Driven Development: A Survey". In: *IEEE Transactions on Software Engineering* (2019).

[8] Carsten Amelunxen and Andy Schürr. "Formalising model transformation rules for UML/MOF 2". In: *IET software* 2.3 (2008), pp. 204–222.

[9] ATTIKO METRO, S.A. *Attiko Metro*. http://www.ametro.gr/page/default.asp?id=4&la=2. 2016.

[10] Kyle Kellogg Azevedo. "Modeling sustainability in complex urban transportation systems". PhD thesis. Georgia Institute of Technology, 2010.

[11] H. Baali et al. "Empowering Technology Enabled Care Using IoT and Smart Devices: A Review". In: *IEEE Sensors Journal* 18.5 (2018), pp. 1790–1809.

[12] Omar Badreddin et al. "fSysML: Foundational Executable SysML for Cyber-Physical System Modeling." In: *GEMOC@ MoDELS*. 2016, pp. 38–51.

[13] Radhakisan Baheti and Helen Gill. "Cyber-physical systems". In: *The impact of control technology* 12.1 (2011), pp. 161–166.

[14] Mirza Mansoor Baig and Hamid Gholamhosseini. "Smart health monitoring systems: an overview of design and modeling". In: *Journal of medical systems* 37.2 (2013), pp. 1–14.

[15] Manas Bajaj et al. "Slim: collaborative model-based systems engineering workspace for next-generation complex systems". In: *Aerospace Conference, 2011 IEEE*. IEEE. 2011, pp. 1–15.

[16] Loyd Baker et al. "Foundational concepts for model driven system design". In: *INCOSE Model Driven System Design Interest Group* 16 (2000), pp. 15–16.

[17] Laurent Balmelli et al. “Model-driven systems development”. In: *IBM Systems journal* 45.3 (2006), pp. 569–585.

[18] Richard L Baskerville, Mala Kaul, and Veda C Storey. “Genres of Inquiry in Design-Science Research”. In: *Mis Quarterly* 39.3 (2015), pp. 541–564.

[19] Razieh Behjati et al. “Model-based automated and guided configuration of embedded software systems”. In: *European Conference on Modelling Foundations and Applications*. Springer. 2012, pp. 226–243.

[20] John Boardman and Brian Sauser. “System of Systems-the meaning of of”. In: *2006 IEEE/SMC International Conference on System of Systems Engineering*. IEEE. 2006, 6–pp.

[21] Heiko Bock. *The definitive guide to NetBeans Platform*. Apress, 2009.

[22] Barry Boehm et al. “Cost estimation with COCOMO II”. In: *ed: Upper Saddle River, NJ: Prentice-Hall* (2000).

[23] H. Brander et al. “Quality of service in broadband communications”. In: *International Conference on Integrated Broadband Services and Networks, 1990.* 1990, pp. 166–171.

[24] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.

[25] Philipp Brauner et al. “On studying human factors in complex cyber-physical systems”. In: *Mensch und Computer 2016–Workshopband* (2016).

[26] Roger J Brooks and Andrew M Tobias. “Choosing the best model: Level of detail, complexity, and model performance”. In: *Mathematical and computer modelling* 24.4 (1996), pp. 1–14.

[27] Daniel Browne et al. “Enabling systems modeling language authoring in a collaborative web-based decision support tool”. In: *Procedia Computer Science* 16 (2013), pp. 373–382.

[28] Manfred Broy and Albrecht Schmidt. “Challenges in engineering cyber-physical systems”. In: *Computer* 47.2 (2014), pp. 70–72.

[29] Magdalena Bujnowska-Fedak and Urszula Grata-Borkowska. “Use of telemedicine-based care for the aging and elderly: promises and pitfalls”. In: *Smart Homecare Technology and TeleHealth* (2015), p. 91.

[30] Adrian Burns et al. “SHIMMER™–A wireless sensor platform for noninvasive biomedical research”. In: *IEEE Sensors Journal* 10.9 (2010), pp. 1527–1534.

[31] Alan Burns and Robert Davis. “Mixed criticality systems-A review”. In: *Department of Computer Science, University of York, Tech. Rep* (2013), pp. 1–69.

[32] NoMagic, Inc. *Cameo Systems Modeler*. https://www.nomagic.com/products/cameo-systems-modeler. 2021.

[33] Felician Campean and Unal Yildirim. “Enhanced sequence diagram for function modelling of complex systems”. In: *Procedia Cirp* 60 (2017), pp. 273–278.

[34] Ionut Cardei, Mihai Fonoage, and Ravi Shankar. “Model based requirements specification and validation for component architectures”. In: *2008 2nd Annual IEEE Systems Conference*. IEEE. 2008, pp. 1–8.

[35] Nelson H. Carreras Guzman et al. “Conceptualizing the key features of cyber-physical systems in a multi-layered representation for safety and security analysis”. In: *Systems Engineering* 23.2 (2020), pp. 189–210.

[36] Christos G. Cassandras. “Discrete-Event Systems”. In: *Handbook of Networked and Embedded Control Systems*. 2005.

[37] Olivier Casse. *SysML in Action with Cameo Systems Modeler*. Elsevier, 2017.

[38] Moo Hyun Cha and Duhwan Mun. "Discrete event simulation of Maglev transport considering traffic waves". In: *Journal of Computational Design and Engineering* 1.4 (2014), pp. 233–242.

[39] Yu-Hern Chang, Chung-Hsing Yeh, and Ching-Cheng Shen. "A multiobjective model for passenger train services planning: application to Taiwan's high-speed rail line". In: *Transportation Research Part B: Methodological* 34.2 (2000), pp. 91–106.

[40] Kee-Kuo Chen, Huang Madeleine, et al. "Journal of the Eastern Asia Society for Transportation Studies." In: (2001).

[41] Ming Cheng et al. "A review of flexible force sensors for human health monitoring". In: *Journal of Advanced Research* (2020).

[42] Jean-Yves Choley et al. "Topology-based Safety Analysis for Safety Critical CPS". In: *Procedia Computer Science* 95 (2016). Complex Adaptive Systems Los Angeles, CA November 2-4, 2016, pp. 32–39. ISSN: 1877-0509.

[43] Diane J Cook et al. "CASAS: A smart home in a box". In: *Computer* 46.7 (2013), pp. 62–69.

[44] Bruno Costa, Paulo F Pires, and Flávia C Delicato. "Modeling IoT Applications with SysML 4IoT". In: *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2016, pp. 157–164.

[45] Bruno Costa et al. "Design and analysis of IoT applications: a model-driven approach". In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE. 2016, pp. 392–399.

[46] Filip Cuckov, Grant Rudd, and Liam Daly. "Framework for model-based design and verification of human-in-the-loop cyber-physical systems". In: *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2017, pp. 401–402.

[47] Matthew Dabkowski et al. "Network science enabled cost estimation in support of mbse". In: *Procedia Computer Science* 16 (2013), pp. 89–97.

[48] Judith Dahmann et al. "SysML executable systems of system architecture definition: A working example". In: *2017 Annual IEEE International Systems Conference (SysCon)*. IEEE. 2017, pp. 1–6.

[49] Wei Dai and Olgica Milenkovic. "Subspace pursuit for compressive sensing signal reconstruction". In: *Information Theory, IEEE Transactions on* 55.5 (2009), pp. 2230–2249.

[50] Mourad Debbabi et al. *Verification and validation in systems engineering: assessing UML/SysML design models*. Springer Science & Business Media, 2010.

[51] Lenny Delligatti. *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[52] Peng Deng et al. "A model-based synthesis flow for automotive CPS". In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems, ICCPS 2015, Seattle, WA, USA, April 14-16, 2015*. Ed. by Alexandre M. Bayen and Michael S. Branicky. ACM, 2015, pp. 198–207.

[53] Tharam Dillon et al. "Cyber-physical systems: Providing Quality of Service (QoS) in a heterogeneous systems-of-systems environment". In: *5th Digital Ecosystems and Technologies Conference*. IEEE. 2011, pp. 330–335.

[54] Hamza Djelouat et al. “Real-time ECG monitoring using compressive sensing on a heterogeneous multicore edge-device”. In: *Microprocessors and Microsystems* 72 (2020), p. 102839. ISSN: 0141-9331.

[55] Dov Dori. “SysML: Use Case, Block, and State Machine Diagrams”. In: *Model-Based Systems Engineering with OPM and SysML*. Springer, 2016, pp. 29–35.

[56] Charalampos Doukas and Ilias Maglogiannis. “Bringing IoT and cloud computing towards pervasive healthcare”. In: *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE. 2012, pp. 922–926.

[57] Oliver C Eichmann, Sylvia Melzer, and Ralf God. “Model-based development of a system of systems using unified architecture framework (UAF): A case study”. In: *2019 IEEE International Systems Conference (SysCon)*. IEEE. 2019, pp. 1–8.

[58] Christos Emmanouilidis et al. “Enabling the human in the loop: Linked data and knowledge in industrial cyber-physical systems”. In: *Annual Reviews in Control* 47 (2019), pp. 249–265.

[59] BS EN. “13816: 2002: Transportation”. In: *Logistics and Services. Public Passenger Transport. Service Quality Definition, Targeting and Measurement* (2002).

[60] Jose Luis Espinosa-Aranda and Ricardo Garcıa-Ródenas. “A discrete event-based simulation model for real-time traffic management in railways”. In: *Journal of Intelligent Transportation Systems* 16.2 (2012), pp. 94–107.

[61] Huascar Espinoza et al. “Challenges in combining SysML and MARTE for model-based design of embedded systems”. In: *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2009, pp. 98–113.

[62] Jeff A Estefan et al. “Survey of model-based systems engineering (MBSE) methodologies”. In: *Incose MBSE Focus Group* 25.8 (2007), pp. 1–12.

[63] Bahar Farahani et al. “Towards collaborative intelligent IoT eHealth: From device to fog, and cloud”. In: *Microprocessors and Microsystems* 72 (2020), p. 102938.

[64] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. 1st. Addison-Wesley Professional, 2012. ISBN: 0321888944.

[65] Evangelia Filiopoulou et al. “Integrating cost analysis in the cloud: A SoS approach”. In: *2015 11th International Conference on Innovations in Information Technology (IIT)*. IEEE. 2015, pp. 278–283.

[66] George S Fishman. *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.

[67] Florida Department of Transportation. *2013 QUALITY/LEVEL OF SERVICE HANDBOOK*. http://teachamerica.com/tih/PDF/2013QLOSHandbook.pdf. 2013.

[68] H Fouad et al. “Analyzing patient health information based on IoT sensor with AI for improving patient assistance in the future direction”. In: *Measurement* (2020), p. 107757.

[69] Daniel Fredes et al. “Automatic transformation applied to a software process using MDA”. In: *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*. 2014, pp. 1–4.

[70] Sanford A Friedenthal and Cris Kobryn. “4.1. 2 Extending UML to Support a Systems Modeling Language”. In: *INCOSE International Symposium*. Vol. 14. 1. Wiley Online Library. 2004, pp. 686–706.

[71] Sanford Friedenthal, Regina Griego, and Mark Sampson. "INCOSE model based systems engineering (MBSE) initiative". In: *INCOSE 2007 symposium*. Vol. 11. 2007.

[72] "Preface". In: *A Practical Guide to SysML (Third Edition)*. Ed. by Sanford Friedenthal, Alan Moore, and Rick Steiner. Third Edition. The MK/OMG Press. Boston: Morgan Kaufmann, 2015, pp. xvii–xx. ISBN: 978-0-12-800202-5.

[73] Sanford Friedenthal and Christopher Oster. *Architecting Spacecraft with SysML: A Model-Based Systems Engineering Approach*. CreateSpace Independent Publishing Platform, 2017.

[74] Peter Fritzson. "Modelica—A cyber-physical modeling language and the OpenModelica environment". In: *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE. 2011, pp. 1648–1653.

[75] Peter Fritzson and Vadim Engelson. "Modelica—A unified object-oriented language for system modeling and simulation". In: *European Conference on Object-Oriented Programming*. Springer. 1998, pp. 67–90.

[76] Peter Fritzson et al. "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 4 (2020), pp. 241–295.

[77] JJ Fruin. "Pedestrian planning and design." In: (1987), p. 206.

[78] Z. Fu et al. "Modeling and Integrating Human Interaction Assumptions in Medical Cyber-Physical System Design". In: *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*. 2017, pp. 373–378.

[79] Marisol Garcıa-Valls et al. "Pragmatic cyber physical systems design based on parametric models". In: *Journal of Systems and Software* 144 (2018), pp. 559–572.

[80] Julien Gardan and Nada Matta. "Enhancing knowledge management into systems engineering through new models in SysML". In: *Procedia CIRP* 60 (2017), pp. 169–174.

[81] S. Garredu et al. "A survey of Model-Driven approaches applied to DEVS - a comparative study of metamodels and transformations". In: *Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2014 International Conference on*. 2014, pp. 179–187.

[82] Domenico Gattuso and Antonio Restuccia. "A tool for railway transport cost evaluation". In: *Procedia-Social and Behavioral Sciences* 111 (2014), pp. 549–558.

[83] Jürgen Gausemeier et al. *SYSTEMS ENGINEERING in industrial practice*. University of Paderborn, 2015.

[84] Guido L Geerts. "A design science research methodology and its application to accounting information systems research". In: *International Journal of Accounting Information Systems* 12.2 (2011), pp. 142–151.

[85] Philipp Geyer et al. "A Systems Engineering Methodology for Designing and Planning the Built Environment—Results from the Urban Research Laboratory Nuremberg and Their Integration in Education". In: *Systems* 2.2 (2014), pp. 137–158.

[86] Miriam Gil et al. "Designing human-in-the-loop autonomous Cyber-Physical Systems". In: *International Journal of Human-Computer Studies* 130 (2019), pp. 21–39.

[87] Martin Glinz. "On non-functional requirements". In: *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. IEEE. 2007, pp. 21–26.

[88] Carlos Alberto Gonzalez Perez et al. *A SysML-Based Methodology for Model Testing of Cyber-Physical Systems*. Tech. rep. University of Luxembourg, 2018.

[89] Robert B Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[90] Pablo Grube, Felipe Núñez, and Aldo Cipriano. "An event-driven simulator for multi-line metro systems and its application to Santiago de Chile metropolitan rail network". In: *Simulation Modelling Practice and Theory* 19.1 (2011), pp. 393–405.

[91] Matthias Güdemann et al. "SysML in digital engineering". In: *Proceedings of the First International Workshop on Digital Engineering*. 2010, pp. 1–8.

[92] Reinhard Haberfellner et al. *Systems engineering*. Springer, 2019.

[93] Kyle Hampson. "Technical evaluation of the systems modeling language (SysML)". In: *Procedia Computer Science* 44 (2015), pp. 403–412.

[94] Li Haoyu et al. "An IoMT cloud-based real time sleep apnea detection scheme by using the SpO2 estimation supported by heart rate variability". In: *Future Generation Computer Systems* 98 (2019), pp. 69–77.

[95] Cecilia Haskins et al. "Systems engineering handbook". In: *INCOSE*. Vol. 9. 2006, pp. 13–16.

[96] Matthew Hause et al. "The SysML modelling language". In: *Fifteenth European Systems Engineering Conference*. Vol. 9. 2006, pp. 1–12.

[97] Matthew Hause, Graham Bleakley, and Aurelijus Morkevicius. "Technology update on the unified architecture framework (uaf)". In: *INSIGHT* 20.2 (2017), pp. 71–78.

[98] Andrea Herrmann and Barbara Paech. "MOQARE: misuse-oriented quality requirements engineering". In: *Requirements Engineering* 13.1 (2008), pp. 73–86.

[99] Erik Herzog, Asmus Pandikow, and AB Syntell. "SysML–an assessment". In: *Syntell AB, SE* 100 (2005), p. 55.

[100] Alan R Hevner et al. "Design science in information systems research". In: *Management Information Systems Quarterly* 28.1 (2008), p. 6.

[101] Hannes Hick, Matthias Bajzek, and Clemens Faustmann. "Definition of a system model for model-based development". In: *SN Applied Sciences* 1.9 (2019), pp. 1–15.

[102] Rich Hilliard. "Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems". In: *IEEE* 12.16-20 (2000), p. 2000.

[103] Oliver Hoehne. "Rail Systems Viewed from a System-of-Systems Perspective". In: *INSIGHT* 19.3 (2016), pp. 36–38.

[104] Hans-Peter Hoffmann. "Deploying model-based systems engineering with IBM® rational® solutions for systems and software engineering". In: *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*. IEEE. 2012, pp. 1–8.

[105] Jon Holt and Simon Perry. *SysML for systems engineering*. Vol. 7. IET, 2008.

[106] Andreas Horni, Kai Nagel, and Kay W Axhausen. *The multi-agent transport simulation MATSim*. Ubiquity Press London, 2016.

[107] Imre Horváth et al. "Comparison of three methodological approaches of design research". In: *DS 42: Proceedings of ICED 2007, the 16th International Conference on Engineering Design, Paris, France, 28.-31.07. 2007*. 2007, pp. 361–362.

[108] Fei Hu et al. "Robust Cyber–Physical Systems: Concept, models, and implementation". In: *Future Generation Computer Systems* 56 (2016), pp. 449–475. ISSN: 0167-739X.

[109] Yilin Huang, Mamadou D Seck, and Alexander Verbraeck. "A DEVS library for rail operations simulation". In: *Proceedings of the 2011 Emerging M&S Applications in Industry and Academia Symposium*. Society for Computer Simulation International. 2011, pp. 76–83.

[110] Mahmoud Hussein, Shuai Li, and Ansgar Radermacher. "Model-driven Development of Adaptive IoT Systems." In: *MODELS (Satellite Events)*. 2017, pp. 17–23.

[111] *IEEE 7000-2021 - IEEE Standard Model Process for Addressing Ethical Concerns during System Design*. 2021. URL: https://standards.ieee.org/standard/7000-2021.html.

[112] Juhani Iivari and John R Venable. "Action research and design science research-Seemingly similar but decisively dissimilar". In: (2009).

[113] N. Iriondo, E. Estévez, and M. Marcos. "Automatic Generation of the Supervisor Code for Industrial Switched-Mode Systems". In: *IEEE Transactions on Industrial Informatics* 9.4 (2013), pp. 1868–1878. ISSN: 1551-3203.

[114] ISO ISO. "IEC 25000 Software and system engineering–Software product Quality Requirements and Evaluation (SQuaRE)–Guide to SQuaRE". In: *International Organization for Standarization* (2005).

[115] Jovan Ivković et al. "ODROID-XU4 as a desktop PC and microcontroller development boards alternative". In: *Proc. 6th Int. Conf.(TIO)*. 2016.

[116] Matthias Jarke et al. "The brave new world of design requirements". In: *Information Systems* 36.7 (2011), pp. 992–1008.

[117] Yosr Jarraya, Mourad Debbabi, and Jamal Bentahar. "On the meaning of SysML activity diagrams". In: *2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. IEEE. 2009, pp. 95–105.

[118] Jeff C Jensen, Danica H Chang, and Edward A Lee. "A model-based design methodology for cyber-physical systems". In: *2011 7th international wireless communications and mobile computing conference*. IEEE. 2011, pp. 1666–1671.

[119] M Jirgl, Z Bradac, and P Fiedler. "Human-in-the-Loop Issue in Context of the Cyber-Physical Systems". In: *IFAC-PapersOnLine* 51.6 (2018), pp. 225–230.

[120] M. Jirgl, Z. Bradac, and P. Fiedler. "Human-in-the-Loop Issue in Context of the Cyber-Physical Systems". In: *IFAC-PapersOnLine* 51.6 (2018). 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018, pp. 225–230. ISSN: 2405-8963.

[121] Gulraiz J Joyia et al. "Internet of Medical Things (IOMT): applications, benefits and future challenges in healthcare domain". In: *J Commun* 12.4 (2017), pp. 240–7.

[122] Christos Kalloniatis, Evangelia Kavakli, and Stefanos Gritzalis. "Addressing privacy requirements in system design: the PriS method". In: *Requirements Engineering* 13.3 (2008), pp. 241–255.

[123] GD Kapos et al. *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. 2013.

[124] George-Dimitrios Kapos et al. "An Integrated Framework for Automated Simulation of SysML Models Using DEVS". In: *Simulation* 90.6 (June 2014), pp. 717–744. ISSN: 0037-5497.

[125] George-Dimitrios Kapos et al. "An integrated framework for automated simulation of SysML models using DEVS". In: *Simulation* 90.6 (2014), pp. 717–744.

[126] George-Dimitrios Kapos et al. "Model-based system engineering using SysML: Deriving executable simulation models with QVT". In: *2014 IEEE International Systems Conference Proceedings*. IEEE. 2014, pp. 531–538.

[127] George-Dimitrios Kapos et al. "A declarative approach for transforming SysML models to executable simulation models". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2019).

[128] Robert Karban, Nerijus Jankevičius, and Maged Elaasar. "Esem: Automated systems analysis using executable sysml modeling patterns". In: *INCOSE International Symposium*. Vol. 26. 1. Wiley Online Library. 2016, pp. 1–24.

[129] R Karban et al. "MBSE Initiative–SE2 Challenge Team, Cookbook for MBSE with SysML". In: *SE2 Challenge Team* (2011).

[130] Waldemar Karwowski and Tareq Z Ahram. "Interactive management of human factors knowledge for human systems integration using systems modeling language". In: *Information Systems Management* 26.3 (2009), pp. 262–274.

[131] Shwetambara Kekade et al. "The usefulness and actual use of wearable devices among the elderly population". In: *Computer methods and programs in biomedicine* 153 (2018), pp. 137–159.

[132] Aleksandr A Kerzhner and Christiaan JJ Paredis. "A SysML-based language for modeling system-level architecture selection decisions". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 45011. American Society of Mechanical Engineers. 2012, pp. 1263–1276.

[133] T. G. Kim. *DEVSim++ © User's Manual. C++ Based Simulation with Hierarchical Modular DEVS Models*. Version 2.0. 1998.

[134] Daichi Kimura et al. "Evaluation of IT systems considering characteristics as system of systems". In: *System of Systems Engineering (SoSE), 2011 6th International Conference on*. IEEE. 2011, pp. 43–48.

[135] Kittelson et al. *Transit capacity and quality of service manual*. 100. Transportation Research Board, 2003.

[136] Anneke G Kleppe et al. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.

[137] Bernhard Kölmel et al. "Usability requirements for complex cyber-physical systems in a totally networked world". In: *Working Conference on Virtual Enterprises*. Springer. 2014, pp. 253–258.

[138] Kevin T Kornegay, Gang Qu, and Miodrag Potkonjak. "Quality of service and system design". In: *VLSI'99. Proceedings. IEEE Computer Society Workshop On*. IEEE. 1999, pp. 112–117.

[139] Christos Kotronis. *Plugins Bitbucket repository*. 2021. URL: https://bitbucket.org/ckotronis-phd/plugins.

[140] Christos Kotronis, Anargyros Tsadimas, and Mara Nikolaidou. "Providing Designers with Automated Decision-Making within SysML Models to Promote Efficient Model-Based Systems Design". In: *2021 IEEE International Systems Conference (SysCon)*. IEEE. 2021, pp. 1–8.

[141] Christos Kotronis et al. "Simulating sysml transportation models". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016, pp. 001674–001679.

[142] Christos Kotronis et al. "Managing criticalities of e-health iot systems". In: *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband (ICUWB)*. IEEE. 2017, pp. 1–5.

[143] Christos Kotronis et al. "A model-based approach for managing criticality requirements in e-health iot systems". In: *2018 13th annual conference on system of systems engineering (SoSE)*. IEEE. 2018, pp. 60–67.

[144] Christos Kotronis et al. "Exploring LoS in railway transportation systems using SysML". In: *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE. 2018, pp. 1–8.

[145] Christos Kotronis et al. "A model-based approach for the design of cyber-physical human systems emphasizing human concerns". In: *2019 IEEE International Congress on Internet of Things (ICIOT)*. IEEE. 2019, pp. 100–107.

[146] Christos Kotronis et al. "Evaluating Internet of Medical Things (IoMT)-Based Systems from a Human-Centric Perspective". In: *Internet of Things* 8 (2019), p. 100125. ISSN: 2542-6605.

[147] Christos Kotronis et al. "Leveraging Quality of Service and Cost in Cyber-Physical Systems Design". In: *International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer. 2019, pp. 208–217.

[148] Christos Kotronis et al. "Employing SysML to model and explore levels-of-service: The case of passenger comfort in railway transportation systems". In: *Systems Engineering* 23.1 (2020), pp. 82–99.

[149] Christos Kotronis et al. "Extending sysML to integrate cost analysis into model-based systems engineering". In: *IEEE Journal of Engineering Management* (2021). accepted for publication.

[150] Christos Kotronis et al. "Extending sysML to integrate cost analysis into model-based systems engineering". In: *Transactions on Internet of Things Journal* (2021). submitted for publication.

[151] Christos Kotronis et al. "Incorporating patient concerns into design requirements for IoMT-based systems: The fall detection case study". In: *Health informatics journal* 27.1 (2021).

[152] Matthew Krugh and Laine Mears. "A complementary cyber-human systems framework for industry 4.0 cyber-physical systems". In: *Manufacturing Letters* 15 (2018), pp. 89–92.

[153] M Arun Kumar, R Vimala, and KR Aravind Britto. "A cognitive technology based healthcare monitoring system and medical data transmission". In: *Measurement* 146 (2019), pp. 322–332.

[154] Sathish AP Kumar et al. "Securing IoT-based cyber-physical human systems against collaborative attacks". In: *2017 IEEE International Congress on Internet of Things (ICIOT)*. IEEE. 2017, pp. 9–16.

[155] Ivan Kurtev. "State of the art of QVT: A model transformation language standard". In: *International Symposium on Applications of Graph Transformations with Industrial Relevance*. Springer. 2007, pp. 377–393.

[156] Ohbyung Kwon, Jae Moon Shim, and Geunchan Lim. "Single activity sensor-based ensemble analysis for health monitoring of solitary elderly people". In: *Expert Systems with Applications* 39.5 (2012), pp. 5774–5783.

[157] L. D. Lago et al. "Dependability Assessment of SOA-Based CPS With Contracts and Model-Based Fault Injection". In: *IEEE Transactions on Industrial Informatics* 14.1 (Jan. 2018), pp. 360–369. ISSN: 1941-0050.

[158] J Lane. "Cost model extensions to support systems engineering cost estimation for complex systems and systems of systems". In: *7th Annual Conference on Systems Engineering Research*. Citeseer. 2009.

[159] P. G. Larsen et al. “Integrated tool chain for model-based design of Cyber-Physical Systems: The INTO-CPS project”. In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. Apr. 2016, pp. 1–6.

[160] Pedro Merino Laso, David Brosset, and John Puentes. “Monitoring approach of cyber-physical systems by quality measures”. In: *International Conference on Sensor Systems and Software*. Springer. 2016, pp. 105–117.

[161] Edward Lee. “The Past, Present and Future of Cyber-Physical Systems: A Focus on Models”. In: *Sensors (Basel, Switzerland)* 15 (Mar. 2015), pp. 4837–4869.

[162] Patrick Leserf, Pierre de Saqui-Sannes, and Jérôme Hugues. “Trade-off analysis for SysML models using decision points and CSPs”. In: *Software & Systems Modeling* (2019), pp. 1–17.

[163] Long Li et al. “Executable System-of-Systems architecting based on DoDAF meta-model”. In: *2012 7th International Conference on System of Systems Engineering (SoSE)*. IEEE. 2012, pp. 362–367.

[164] Marcos V Linhares et al. “Introducing the modeling and verification process in SysML”. In: *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. IEEE. 2007, pp. 344–351.

[165] Yang Liu, Yu Peng, et al. “Review on cyber-physical systems”. In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017), pp. 27–40.

[166] Howard Lykins, Sanford Friedenthal, and Abraham Meilich. “4.4. 4 Adapting UML for an Object Oriented Systems Engineering Method (OOSEM)”. In: *INCOSE International Symposium*. Vol. 10. 1. Wiley Online Library. 2000, pp. 490–497.

[167] Ray Madachy and David Jacques. “System Cost Modeling and SysML Integration in Model-Based Systems Engineering”. In: *INCOSE Chapter Meeting, San Diego, CA*. 2017.

[168] Azad M Madni and Shatad Purohit. “Economic analysis of model-based systems engineering”. In: *Systems* 7.1 (2019), p. 12.

[169] Azad M. Madni et al. *Next Generation Adaptive Cyber Physical Human Systems*. Tech. rep. SERC-2018-TR-112. Systems Engineering Research Center (SERC), 2018. URL: https://apps.dtic.mil/sti/pdfs/AD1099997.pdf.

[170] Biswajit Mahanty et al. “Human skin interactive self-powered piezoelectric e-skin based on PVDF/MWCNT electrospun nanofibers for non-invasive health care monitoring”. In: *Materials Today: Proceedings* 21 (2020), pp. 1964–1968.

[171] Sanda Mandutianu et al. “1.3. 3 Conceptual Model for Space Mission Systems Design”. In: *INCOSE International Symposium*. Vol. 19. 1. Wiley Online Library. 2009, pp. 110–131.

[172] Vasiliki Mantzana et al. “Identifying healthcare actors involved in the adoption of information systems”. In: *European Journal of Information Systems* 16.1 (2007), pp. 91–102.

[173] Benedikt Martens, Marc Walterbusch, and Frank Teuteberg. “Costing of cloud computing services: A total cost of ownership approach”. In: *2012 45th Hawaii International Conference on System Sciences*. IEEE. 2012, pp. 1563–1572.

[174] Dr. James N Martin. *Unified Architecture Framework (UAF)*. June 2021. URL: https://aerospace.org/story/unified-architecture-framework-uaf.

[175] Francisco E Martınez Martınez and Benjamın Colucci. “Final Report Application of SIMAN ARENA Discrete Event Simulation Tool in the Operational Planning of a Rail System”. In: (2002).

[176] Eva Martınez-Caro et al. “Healthcare service evolution towards the Internet of Things: An end-user perspective”. In: *Technological Forecasting and Social Change* (2018).

[177] Matthew John McGill. “UML Class Diagram Syntax: An Empirical Study of Comprehension.” In: (2001).

[178] *MagicDraw UML*. http://www.magicdraw.com/.

[179] Ehsan Moghadas, Javad Rezazadeh, and Reza Farahbakhsh. “An IoT Patient Monitoring based on Fog computing and Data Mining: Cardiac Arrhythmia Usecase”. In: *Internet of Things* (2020), p. 100251.

[180] Nazila Mohammadi and Maritta Heisel. “A framework for systematic refinement of trustworthiness requirements”. In: *Information* 8.2 (2017), p. 46.

[181] Senthilkumar Mohan, Chandrasegar Thirumalai, and Gautam Srivastava. “Effective heart disease prediction using hybrid machine learning techniques”. In: *IEEE Access* 7 (2019), pp. 81542–81554.

[182] Daniel Moldovan, Georgiana Copil, and Schahram Dustdar. “Elastic systems: Towards cyber-physical ecosystems of people, processes, and things”. In: *Computer Standards & Interfaces* 57 (2018), pp. 76–82. ISSN: 0920-5489.

[183] K Monisha and M Rajasekhara Babu. “A Novel Framework for Healthcare Monitoring System Through Cyber-Physical System”. In: *Internet of Things and Personalized Healthcare Systems*. Springer, 2019, pp. 21–36.

[184] Y. Mordecai, O. Orhof, and D. Dori. “Model-Based Interoperability Engineering in Systems-of-Systems and Civil Aviation”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.4 (Apr. 2018), pp. 637–648. ISSN: 2168-2216.

[185] Marco Mori et al. “Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile”. In: *Journal of Software: Evolution and Process* 30.3 (2018), e1878.

[186] Aurelijus Morkevicius et al. “MBSE Grid: A Simplified SysML-Based Approach for Modeling Complex Systems”. In: *INCOSE International Symposium*. Vol. 27. 1. Wiley Online Library. 2017, pp. 136–150.

[187] Dimitris Mourtzis and Ekaterini Vlachou. “Cloud-based cyber-physical systems and quality of services”. In: *The TQM Journal* 28.5 (2016), pp. 704–733.

[188] R.C. Moyer et al. *Contemporary financial management*. Nelson Education, 2012.

[189] Mohammed Abdalla Osman Mukhtar, Azween Abdullah, and Alan G Downe. “Preliminary overview about relations QVT: Query/View/Transformation model transformation language”. In: *2011 National Postgraduate Conference*. IEEE. 2011, pp. 1–5.

[190] Miroslav Muzny et al. “Wearable sensors with possibilities for data exchange: Analyzing status and needs of different actors in mobile health monitoring systems”. In: *International journal of medical informatics* 133 (2020), p. 104017.

[191] Andrew Nash and Daniel Huerlimann. “Railroad simulation using OpenTrack”. In: *WIT Transactions on The Built Environment* 74 (2004).

[192] Mara Nikolaidou and Christos Michalakelis. “Techno-economic analysis of SysML models”. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE. 2017, pp. 1–6.

[193] Mara Nikolaidou et al. “A consistent framework for enterprise information system engineering”. In: *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*. IEEE. 2006, pp. 492–496.

[194] Mara Nikolaidou et al. "Simulating SysML models: Overview and challenges". In: *2015 10th System of Systems Engineering Conference (SoSE)*. IEEE. 2015, pp. 328–333.

[195] Mara Nikolaidou et al. "Incorporating patient concerns into design requirements for IoMT-based systems: The fall detection case study". In: *Health Informatics Journal* 27.1 (2021), p. 1460458220982640.

[196] NoMagic. *Cameo Simulation Toolkit*. 2015. URL: https://www.3ds.com/products-services/catia/products/no-magic/addons/cameo-simulation-toolkit/.

[197] Object Management Group. *Object Management Group*. https://www.omg.org/index.htm. 2019. (Visited on 02/02/2018).

[198] Object Management Group - BPMN. *Business Process Model And Notation*. 2018. URL: https://www.omg.org/spec/BPMN/2.0/ (visited on 07/27/2018).

[199] Object Management Group, Ed. *OMG Systems Modeling Language (OMG SysML) Version 1.6 beta*. 2019. URL: https://www.omg.org/spec/SysML/ (visited on 03/2019).

[200] Sergio F Ochoa, Giancarlo Fortino, and Giuseppe Di Fatta. "Cyber-physical systems, internet of things and big data". In: *Future Generat. Comput. Syst.* 75 (2017), pp. 82–84.

[201] Ian Oliver and Vesa Luukkala. "On UML's composite structure diagram". In: *Fifth Workshop on System Analysis and Modelling*. 2006.

[202] OMG. "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification". In: *Transformation* April (2008), pp. 1–230. URL: http://www.omg.org/spec/QVT/1.0/PDF/.

[203] OMG. "Unified Architecture Framework (UAF)". In: (2017).

[204] Ed OMG. "OMG Systems Modeling Language (OMG SysML)-Version 1.4". In: (2008).

[205] *OpenModelica*. https://openmodelica.org/.

[206] Andrew Y-Z Ou et al. "Using human intellectual tasks as guidelines to systematically model medical cyber-physical systems". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2016, pp. 004394–004399.

[207] Viktorija Ovchinnikova and Erika Nazaruka. "The Validation Possibility of Topological Functioning Model using the Cameo Simulation Toolkit." In: *ENASE*. 2016, pp. 327–336.

[208] Marie-Pierre Pacaux-Lemoine et al. "Towards human-based industrial cyber-physical systems". In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2018, pp. 615–620.

[209] Xing Pan, Baoshi Yin, and Jianmi Hu. "Modeling and simulation for SoS based on the DoDAF framework". In: *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*. IEEE. 2011, pp. 1283–1287.

[210] Barry Papke, Saulius Pavalkis, and Gan Wang. "Enabling Repeatable SE Cost Estimation with COSYSMO and MBSE". In: *INCOSE International Symposium*. Vol. 27. 1. Wiley Online Library. 2017, pp. 1699–1713.

[211] *ParaMagic Plugin*. visited on 20/07/2019. URL: https://www.nomagic.com/product-addons/magicdraw-addons/paramagic-plugin.

[212] Christiaan JJ Paredis et al. "5.5. 1 An overview of the SysML-modelica transformation specification". In: *INCOSE International Symposium*. Vol. 20. Wiley Online Library. 2010, pp. 709–722.

[213] Christopher J. Pavlovski and Joe Zou. "Non-functional Requirements in Business Process Modeling". In: *Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling - Volume 79*. APCCM '08. Wollongong, NSW, Australia: Australian Computer Society, Inc., 2008, pp. 103–112. ISBN: 978-1-920682-60-6.

[214] Paul Pearce and Matthew Hause. "Iso-15288, oosem and model-based submarine design". In: (2012).

[215] Ken Peffers et al. "A design science research methodology for information systems research". In: *Journal of management information systems* 24.3 (2007), pp. 45–77.

[216] Magda Pitsiava-Latinopoulou and Panagiotis Iordanopoulos. "Intermodal Passengers Terminals: Design standards for better level of service". In: *Procedia-Social and Behavioral Sciences* 48 (2012), pp. 3297–3306.

[217] Leyland F Pitt, Richard T Watson, and C Bruce Kavan. "Service quality: a measure of information systems effectiveness". In: *MIS quarterly* (1995), pp. 173–187.

[218] Iman Poernomo. "The meta-object facility typed". In: *Proceedings of the 2006 ACM symposium on Applied computing*. 2006, pp. 1845–1849.

[219] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[220] Aneta Poniszewska-Maranda et al. "Studying usability of AI in the IoT systems/paradigm through embedding NN techniques into mobile smart service system". In: *Computing* 101.11 (2019), pp. 1661–1685.

[221] K Post, G Walley, and J Che. "Integrating Descriptive Models with an Analytical Model Culture–Lessons Learned at Ford". In: *INCOSE IW* (2014).

[222] Hamed Pouryousef, Pasi Lautala, and Thomas White. "Railroad capacity tools and methodologies in the US and Europe". In: *Journal of Modern Transportation* 23.1 (2015), pp. 30–42.

[223] Jun Qi et al. "Advanced Internet of Things for personalised healthcare systems: A survey". In: *Pervasive and Mobile Computing* 41 (2017), pp. 132–149.

[224] H Qurratuaini. "Designing enterprise architecture based on TOGAF 9.1 framework". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 403. 1. IOP Publishing. 2018, p. 012065.

[225] Md Rahman, Naushin Nower, et al. "Requirements Model for Cyber-Physical System". In: *arXiv preprint arXiv:1705.03095* (2017).

[226] Ana Luısa Ramos, José Vasconcelos Ferreira, and Jaume Barceló. "Model-based systems engineering: An emerging approach for modern systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.1 (2011), pp. 101–111.

[227] E Recommendation. "800: Definitions of terms related to quality of service". In: *International Telecommunication Union's Telecommunication Standardization Sector (ITU-T) Std* (2008).

[228] Axel Reichwein and CJ Paredis. "Overview of architecture frameworks and modeling languages for model-based systems engineering". In: *Proc. ASME*. Vol. 1275. 2011.

[229] Mark Richters and Martin Gogolla. "OCL: Syntax, semantics, and tools". In: *Object Modeling with the OCL*. Springer, 2002, pp. 42–68.

[230] Pascal Roques. "SysML vs. UML 2: A detailed comparison". In: *MoDELS 2011* (2011), pp. 16–21.

[231] Mark von Rosing et al. *Business Process Model and Notation-BPMN*. 2015.

[232] Thomas Ruin, Eric Levrat, and Benoıt Iung. "Modeling framework based on SysML and AltaRica data flow languages for developing models to support complex maintenance program quantification". In: *IFAC Proceedings Volumes* 45.31 (2012), pp. 169–174.

[233] Mike Russell. "Using MBSE to enhance system design decision making". In: *Procedia Computer Science* 8 (2012), pp. 188–193.

[234] Michael Ryschkewitsch, Dawn Schaible, and Wiley Larson. "The art and science of systems engineering". In: *Systems Research Forum*. Vol. 3. 02. World Scientific. 2009, pp. 81–100.

[235] Arman Sajedinejad et al. "SIMARAIL: simulation based optimization software for scheduling railway network". In: *Simulation Conference (WSC), Proceedings of the 2011 Winter*. IEEE. 2011, pp. 3730–3741.

[236] Hessam S Sarjoughian, Chao Zhang, and Greggory Scherer. "DEVS-Suite Simulator Guide: TestFrame and Database". In: (2020).

[237] M Pravin Savaridass et al. "Development of smart health monitoring system using Internet of Things". In: *Materials Today: Proceedings* (2020).

[238] Jutta Schade, Thomas Olofsson, and Marcus Schreyer. "Decision-making in a model-based design process". In: *Construction management and Economics* 29.4 (2011), pp. 371–382.

[239] Wladimir Schamai et al. "Virtual verification of system designs against system requirements". In: *International Conference on Model Driven Engineering Languages and Systems (MDELS)*. Springer. 2010, pp. 75–89.

[240] Gunar Schirner et al. "The future of human-in-the-loop cyber-physical systems". In: *Computer* 46.1 (2013), pp. 36–45.

[241] Hans Schlenker. "Distributed constraint-based railway simulation". In: *Applications of Declarative Programming and Knowledge Management*. Springer, 2005, pp. 215–226.

[242] Oliver Schonherr and Oliver Rose. "First Steps Towards a General SysML Model for Discrete Processes in Production Systems". In: *Proceedings of the 2009 Winter Simulation Conference*. Austin, TE, USA, Dec. 2009, pp. 1711–1718.

[243] K Schuitemaker, JG Braakhuis, and M Rajabalinejad. "A model based safety architecture framework for Dutch high speed train lines". In: *System of Systems Engineering Conference (SoSE), 2015 10th*. IEEE. 2015, pp. 24–29.

[244] Fabio Scippacercola et al. "SysML-based and Prolog-supported FMEA". In: *2015 IEEE international symposium on software reliability engineering workshops (ISSREW)*. IEEE. 2015, pp. 174–181.

[245] Bran Selic. "Tutorial: an overview of UML 2". In: *Proceedings of the 28th International conference on Software engineering*. 2006, pp. 1069–1070.

[246] Bran V Selic, Marie-Pierre Gervais, and François Terrier. *Modelling Foundations and Applications*. Springer, 2010.

[247] Chungman Seo and Bernard P Zeigler. "DEVS namespace for interoperable DEVS/SOA". In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE. 2009, pp. 1311–1322.

[248] Dale Shermon. *Systems cost engineering: program affordability management and cost control*. Gower Publishing, Ltd., 2009.

[249] Mei-Cheng Shih, C Tyler Dick, and Christopher PL Barkan. "Impact of passenger train capacity and level of service on shared rail corridors with multiple types of freight trains". In: *Transportation Research Record: Journal of the Transportation Research Board* 2475 (2015), pp. 63–71.

[250] Lucy Shinners et al. "Exploring healthcare professionals' understanding and experiences of artificial intelligence technology use in the delivery of healthcare: An integrative review". In: *Health Informatics Journal* (2019), p. 1460458219874641.

[251] Darius Šilingas and Rimantas Butleris. "Towards implementing a framework for modeling software requirements in MagicDraw UML". In: *Information Technology and Control* 38.2 (2009).

[252] Herbert A Simon. *The sciences of the artificial*. MIT press, 2019.

[253] Laron Smith et al. *TRANSIMS: Transportation analysis and simulation system*. Tech. rep. Los Alamos National Lab., NM (United States), 1995.

[254] Richard Soley et al. "Model driven architecture". In: *OMG white paper* 308.308 (2000), p. 5.

[255] Sulayman K Sowe et al. "Cyber-physical-human systems: Putting people in the loop". In: *IT professional* 18.1 (2016), pp. 10–13.

[256] Victor Sower and Christopher Sower. *Better business decisions using cost modeling*. Business Expert Press, 2015.

[257] Space and Missile Systems Center, U.S.A.F. *SMC Systems Engineering Primer and Handbook*. http://www.acqnotes.com/Attachments/SMC%20System%20Engineering%20Handbook.pdf. 2004.

[258] OMG Available Specification. *Object Constraint Language*. 2006.

[259] OMG Available Specification. "Omg unified modeling language (omg uml), superstructure, v2. 1.2". In: *Object Management Group* 70 (2007).

[260] Jonathan Spruytte et al. "Modeling equipment hierarchy and costs for ICT solutions". In: *Transactions on Emerging Telecommunications Technologies* 30.3 (2019), e3583.

[261] OMG. *Systems Modeling Language (SysML) Specification, Version 1.5*. May 2017. URL: http://www.omg.org/spec/SysML/1.5/PDF.

[262] Md Shamim Talukder et al. "Predicting antecedents of wearable healthcare technology acceptance by elderly: A combined SEM-Neural Network approach". In: *Technological Forecasting and Social Change* 150 (2020), p. 119793.

[263] Behnam Moshkini Tehrani, Jun Wang, and Chao Wang. "Review of Human-in-the-Loop Cyber-Physical Systems (HiLCPS): The Current Status from Human Perspective". In: *Computing in Civil Engineering 2019: Data, Sensing, and Analytics*. American Society of Civil Engineers Reston, VA, 2019, pp. 470–478.

[264] K. Tei et al. "Model-Driven-Development-Based Stepwise Software Development Process for Wireless Sensor Networks". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45.4 (Apr. 2015), pp. 675–687. ISSN: 2168-2216.

[265] Xavier Thirioux et al. "A framework to formalise the MDE foundations". In: *International Workshop on Towers of Models (TOWERS 2007)*. University of York. 2007, pp. 14–30.

[266] Kleanthis Thramboulidis and Foivos Christoulakis. "UML4IoT-A UML profile to exploit IoT in cyber-physical manufacturing systems". In: *arXiv preprint arXiv:1512.04894* (2015).

[267] Kleanthis Thramboulidis, Danai C Vachtsevanou, and Alexandros Solanos. "Cyber-physical microservices: An IoT-based framework for manufacturing systems". In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE. 2018, pp. 232–239.

[268] Lucio Tirone, Emanuele Guidolotti, and Lorenzo Fornaro. "A Tailoring of the Unified Architecture Framework's Meta-Model for the Modeling of Systems-of-Systems". In: *INCOSE International Symposium*. Vol. 28. 1. Wiley Online Library. 2018, pp. 1691–1705.

[269] Lucio Tirone et al. "Application Of The Unified Architecture Framework For The Definition Of A Generic System Architecture Of A Combat System." In: *CIISE*. 2017, pp. 45–53.

[270] Transportation Research Board. "Highway Capacity Manual". In: *National Research Council, Washington, DC* 113 (2010).

[271] Gerhard Troche. "Activity-based rail freight costing: a model for calculating transport costs in different production systems". PhD thesis. KTH, 2009.

[272] Joel A Tropp and Anna C Gilbert. "Signal recovery from random measurements via orthogonal matching pursuit". In: *Information Theory, IEEE Transactions on* 53.12 (2007), pp. 4655–4666.

[273] Anargyros T Tsadimas. "Model-based enterprise information system design: a SysML-based approach". PhD thesis. Harokopio University of Athens. School of Digital Technology. Dept. of Informatics and Telemaatics, 2018.

[274] Anargyros Tsadimas, Mara Nikolaidou, and Dimosthenis Anagnostopoulos. "Extending SysML to explore non-functional requirements: the case of information system design". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 2012, pp. 1057–1062.

[275] Anargyros Tsadimas, Mara Nikolaidou, and Dimosthenis Anagnostopoulos. "Model-Based System Design Using SysML: The Role of the Evaluation Diagram". In: *Web Design and Development: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2016, pp. 278–301.

[276] Anargyros Tsadimas et al. "Simulating simulation-agnostic SysML models for enterprise information systems via DEVS". In: *Simulation Modelling Practice and Theory* 66 (2016), pp. 243–259.

[277] Urban Rail Transport, S.A. *Athens-Piraeus Electric Railways*. http://www.stasy.gr/index.php?id=33&no_cache=1&L=1. 2018.

[278] Arjan van Velzen et al. "Proposing a more comprehensive future total cost of ownership estimation framework for electric vehicles". In: *Energy policy* 129 (2019), pp. 1034–1046.

[279] John R Venable, Jan Pries-Heje, and Richard L Baskerville. "Choosing a design science research methodology". In: (2017).

[280] John Venable and Richard Baskerville. "Eating our own cooking: Toward a more rigorous design science of research methods." In: *Electronic Journal of Business Research Methods* 10.2 (2012).

[281] David D. Walden et al., eds. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. 4th ed. Hoboken, NJ: Wiley, 2015. ISBN: 978-1-118-99940-0.

[282] Ivo Wallimann-Helmer et al. "An Integrated Framework for Ethical and Sustainable Digitalization". In: *2021 Eighth International Conference on eDemocracy & eGovernment (ICEDEG)*. IEEE. 2021, pp. 156–162.

[283] Haoqi Wang et al. "A semantic model for axiomatic systems design". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 232.12 (2018), pp. 2159–2184.

[284] Haoqi Wang et al. "Unified design approach for systems engineering by integrating model-based systems design with axiomatic design". In: *Systems Engineering* 23.1 (2020), pp. 49–64.

[285] Michael E Watson et al. "Human-centered design using system modeling language". In: *Journal of Cognitive Engineering and Decision Making* 11.3 (2017), pp. 252–269.

[286] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Elsevier, 2011.

[287] Tim Weilkiens. *SYSMOD-The Systems Modeling Toolbox-Pragmatic MBSE with SysML*. Lulu. com, 2016.

[288] T Weilkiens et al. "Evaluating and comparing MBSE methodologies for practitioners". In: *2016 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE. 2016, pp. 1–8.

[289] Deliya B Wesley et al. "A socio-technical systems approach to the use of health IT for patient reported outcomes: Patient and healthcare provider perspectives". In: *Journal of Biomedical Informatics: X* 4 (2019), p. 100048.

[290] Tim Wielkiens. "Systems Engineering with SysML/UML: Modeling". In: *Analysis, Design, Morgan Kaufmann* (2008).

[291] Stefan Wiesner et al. "Requirements engineering for cyber-physical systems". In: *IFIP International Conference on Advances in Production Management Systems*. Springer. 2014, pp. 281–288.

[292] Sabine Wolny et al. "Thirteen years of SysML: a systematic mapping study". In: *Software and Systems Modeling* 19.1 (2020), pp. 111–169.

[293] A Wayne Wymore. *Model-based systems engineering*. Vol. 3. CRC press, 2018.

[294] Xu Xiao-Ming, Li Ke-Ping, and Yang Li-Xing. "Discrete event model-based simulation for train movement on a single-line railway". In: *Chinese Physics B* 23.8 (2014), p. 080205.

[295] Lan Yang, Kathryn Cormican, and Ming Yu. "An ontology model for systems engineering derived from iso/iec/ieee 15288: 2015: Systems and software engineering-system life cycle processes". In: *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng* 11.1 (2016), pp. 1–7.

[296] Zhaoxia Yang et al. "A Study of Railway Transportation Simulation System". In: *Traffic And Transportation Studies (2002)*. 2002, pp. 1385–1392.

[297] John A Zachman. "The zachman framework for enterprise architecture". In: *Primer for Enterprise Engineering and Manufacturing.[si]: Zachman International* (2003).

[298] Bernard P Zeigler, Alexandre Muzy, and Ernesto Kofman. *"Theory of modeling and simulation: discrete event & iterative system computational foundations"*. Academic press, 2018.

[299] Kuilin Zhang et al. *"Impact of high-speed passenger trains on freight train efficiency in shared railway corridors"*. Tech. rep. NEXTRANS Center (US), 2015.

[300] C Zheng et al. "Multidisciplinary integration during conceptual design process: A survey on design methods of cyber-physical systems". In: *DS 84: Proceedings of the DESIGN 2016 14th International Design Conference*. 2016, pp. 1625–1634.

[301] Ji Zhou et al. "Human–cyber–physical systems (HCPSs) in the context of new-generation intelligent manufacturing". In: *Engineering* 5.4 (2019), pp. 624–636.

[302] Yufeng Zhu et al. "A methodology of model-based testing for AADL flow latency in CPS". In: *2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion*. IEEE. 2011, pp. 99–105.

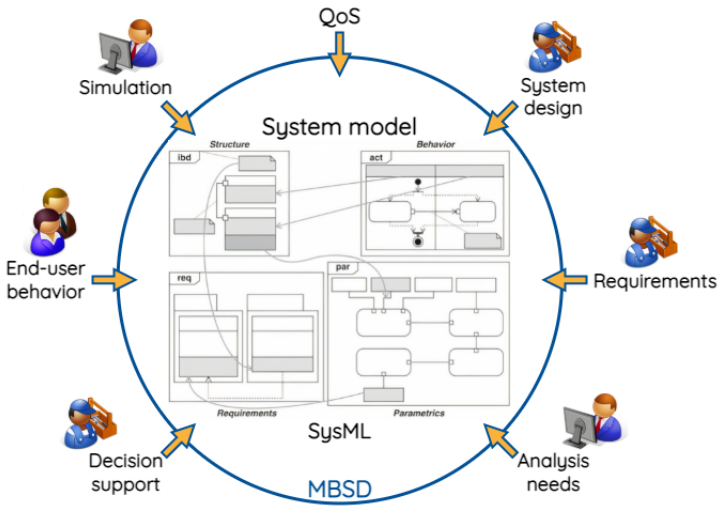
[303] Athanasios Zolotas et al. "Bridging proprietary modelling and open-source model management tools: the case of PTC integrity modeller and epsilon". In: *Software and Systems Modeling* 19.1 (2020), pp. 17–38.

[304] Iveta Zolotová et al. "Smart and cognitive solutions for Operator 4.0: Laboratory H-CPPS case studies". In: *Computers & Industrial Engineering* 139 (2020), p. 105471.

Συνοπτική παρουσίαση διδακτορικής διατριβής

Μοντελο-κεντρική Σχεδίαση Συστημάτων με επίγνωση της Παρεχόμενης Ποιότητας Υπηρεσίας βασισμένη στη Γλώσσα Μοντελοποίησης Συστημάτων SysML

Το θέμα της διδακτορικής διατριβής σχετίζεται με την μοντελο-κεντρική σχεδίαση συστημάτων (MBSD). Το MBSD είναι αποδοτικό για τη σχεδίαση σύνθετων συστημάτων, όπως είναι τα SoS. Μπορεί να συνεισφέρει στον ακριβή καθορισμό, την πλήρη αξιολόγηση, την εγκυρότητα, τη σωστή λειτουργία και πιθανή βελτίωση ενός συστήματος, προσφέροντας ένα κεντρικό εννοιολογικό μοντέλο αυτού.



Σχήμα 6.1: Μοντέλο συστήματος ως ενοποιημένο πλαίσιο MBSD.

Σύμφωνα με την εικόνα 6.1, στο MBSD, το μοντέλο είναι κεντρικό και πρέπει να χρησιμοποιηθεί για όλες τις σχεδιαστικές δραστηριότητες και από όλους τους διαφορετικούς εμπλεκόμενους (stakeholders όπως σχεδιαστές, τελικοί χρήστες, κλπ). Το μοντέλο αυτό περιγράφεται συνήθως από το πρότυπο της SysML, το οποίο υποστηρίζεται από το OMG και έχει υιοθετηθεί από διεθνείς οργανισμούς όπως το INCOSE. Η SysML είναι μία γραφική γλώσσα μοντελοποίησης που βασίζεται στη UML, και είναι ευρύτερα αποδεκτή από την επιστημονική κοινότητα και τη βιομηχανία, λόγω της εφαρμοσιμότητάς της σε διαφορετικές σχεδιαστικές φάσεις, και της αποτελεσματικότητάς της στη σχεδίαση ευρέος φάσματος πεδίων και συστημάτων. Η διεθνής βιβλιογραφία περιλαμβάνει ποικίλες δημοσιευμένες ερευνητικές προσπάθειες και εμπορικές λύσεις βασισμένες στη SysML, οι οποίες στοχεύουν στο MBSD.

Στην παρούσα έρευνα ερευνούμε την παρεχόμενη ποιότητα υπηρεσίας των συστημάτων (QoS) εντός του MBSD. Συγκεκριμένα, το QoS είναι η εύληπτη ένδειξη της παρεχόμενης ποιότητας υπηρεσιών που θέλουν οι χρήστες να έχει το σύστημα που χρησιμοποιούν και αντικατοπτρίζει τα επίπεδα λειτουργίας του συστήματος κάτω από συγκεκριμένες συνθήκες. Όμως, αναγνωρίζουμε δύο βασικά ζητήματα σχετικά με την διερεύνησή του μέχρι σήμερα.

Πρώτον, δεν έχει σημειωθεί πρόοδος σε MBSD όπου λαμβάνονται υπόψη απαιτήσεις παρεχόμενης ποιότητας υπηρεσίας (QoS), δεδομένου ότι δεν υποστηρίζεται με αποδοτικό τρόπο από τυποποιημένα και καθιερωμένα πλαίσια, όπως το SYSMOD και το UAF. Τέτοια πλαίσια και προσεγγίσεις εστιάζουν περισσότερο σε απαιτήσεις που μπορούν άμεσα να συσχετιστούν με το σύστημα, όπως την απόδοση

(performance). Επιπλέον, ως άμεσο stakeholder θεωρούν τυπικά το σχεδιαστή του συστήματος και όχι τον τελικό χρήστη. Το QoS όμως αντιστοιχεί άμεσα στις προσδοκίες του τελευταίου.

Δεύτερον, η SysML δεν υποστηρίζει άμεσα έννοιες συνυφασμένες με το QoS. Ωστόσο, επιτρέπει την αφαιρετική προδιαγραφή απαιτήσεων (requirements) και διαθέτει υποδομή για τη διαχείριση και την επαλήθευσή τους κατά τη διαδικασία σχεδίασης. Για το λόγο αυτό, δύναται να επεκταθεί μέσω παραμετροποιήσιμων μοντέλων (profiles) που διευκολύνουν την ένταξη χαρακτηριστικών QoS στα μοντέλα συστημάτων καθώς και την κάλυψη διαφορετικών πτυχών ή παραμέτρων QoS (όπως το κόστος).

Για να υποστηριχθεί λοιπόν το QoS, πρέπει να οριστούν ή/και να αξιοποιηθούν συγκεκριμένοι δείκτες ποιότητας με βάση τους οποίους κρίνεται εάν ικανοποιούνται οι απαιτήσεις QoS (ή όχι) και άρα εάν παρέχεται η ποιότητα που επιθυμούν οι χρήστες των συστημάτων. Για να πιστοποιηθεί ότι οι δείκτες ποιότητας μπορούν να υποστηριχθούν, αναγκαία είναι η μελέτη και ανάλυση της συμπεριφοράς και της απόδοσης των συστημάτων που σχεδιάζονται. Επιπροσθέτως, οι δείκτες ποιότητας αντιστοιχούν σε οικονομικά κόστη τα οποία είναι αναγκαίο να υπολογιστούν για την επίτευξη του επιθυμητού QoS.

Η παρούσα έρευνα στοχεύει στην ενσωμάτωση του QoS στο MBSD, αξιοποιώντας τη SysML και έχοντας ως βάση την διατύπωση του ενιαίου πλαισίου που προτείνεται στο έργο του Friedenthal et al. Συγκεκριμένα, υλοποιήθηκε ένα γενικό, ενιαίο και ολοκληρωμένο πλαίσιο που αξιοποιεί τις ιδιότητες του MBSD, καθώς και των προηγμένων δυνατοτήτων επέκτασης της SysML (SysML profiles), για τη σχεδίαση σύνθετων συστημάτων, με την ενσωμάτωση και αξιολόγηση απαιτήσεων QoS.

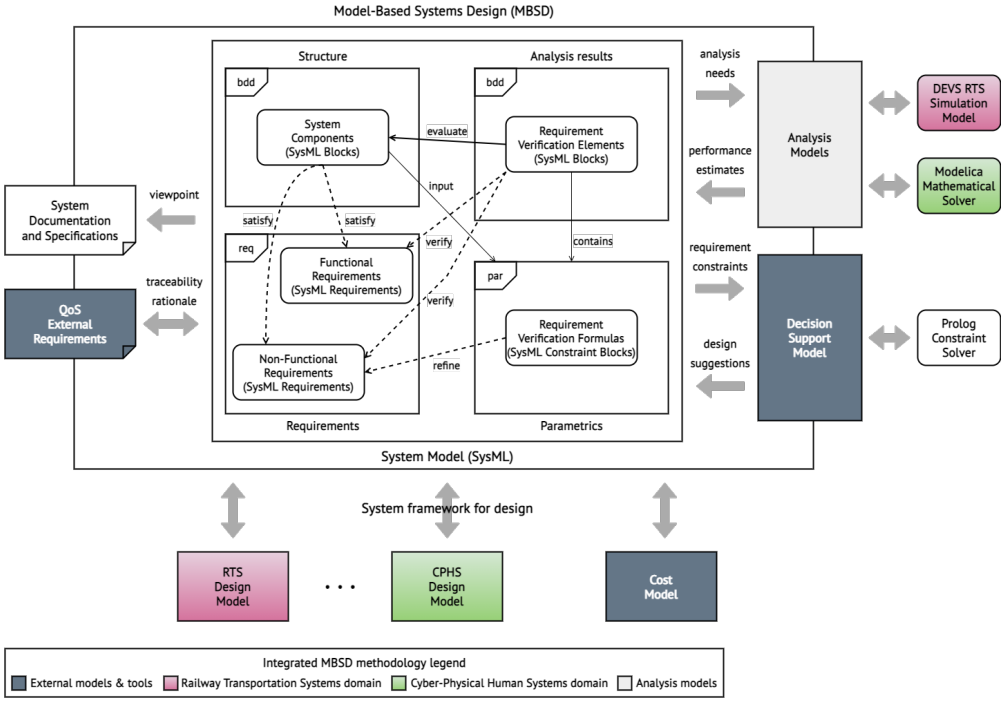
Μέσω του πλαισίου διευκολύνεται άμεσα ο σχεδιαστής συστημάτων στη λήψη αποφάσεων που αφορούν τη σχεδίαση και του προσφέρεται η δυνατότητα να παρατηρήσει τυχόν προβλήματα σε ένα σύστημα, να ελέγξει εάν πληρούνται οι απαιτήσεις ποιότητας και κόστους, καθώς και να λάβει ανάδραση από το περιβάλλον σχεδίασης (στην περίπτωση που δεν πληρούνται). Έτσι, ο σχεδιαστής δύναται να βελτιστοποιήσει το σύστημα και να εξελίξει την ποιότητα και το αντίστοιχο κόστος των υπηρεσιών που παρέχει. Με αυτόν τον τρόπο καλύπτει τις ανάγκες και σχετικές απαιτήσεις των τελικών χρηστών (π.χ., άνετη μετακίνηση σε ένα σύστημα μεταφορών, αξιόπιστη και οικονομική ιατρική παρακολούθηση σε ένα σύστημα τηλεϊατρικής, κλπ) και αυξάνεται η ικανοποίησή τους κατά τη χρήση του συστήματος. Δεδομένου ότι το QoS που πρέπει να παρέχει το υπό σχεδίαση σύστημα δεν είναι εύκολο να προδιαγραφεί ή προσδιοριστεί μελετώντας μόνον τα συστατικά στοιχεία του συστήματος, πρέπει να αναλυθεί η συμπεριφορά του και η εξέλιξή του στο χρόνο για να προσδιοριστεί το επίπεδο QoS που υποστηρίζει. Επομένως έπρεπε να ολοκληρωθούν οι ακόλουθες επεκτάσεις πλαισίου, όπως απεικονίζονται συγκεντρωτικά στην εικόνα 6.2.

QoS-external requirements. Είναι απαραίτητη η προδιαγραφή ενός μετα-μοντέλου/επέκτασης της SysML για την περιγραφή της έννοιας ποιότητα υπηρεσίας ως απαίτηση και συσχέτισης της με τα δομικά στοιχεία του συστήματος. Η περιγραφή αυτή πρέπει να ενσωματωθεί στο μοντέλο του συστήματος που μελετά κατά τη σχεδίαση ο σχεδιαστής.

Analysis models. Για να μπορέσει ο σχεδιαστής να αποφανθεί εάν ικανοποιούνται οι απαιτήσεις QoS, θα πρέπει οπωσδήποτε να μελετήσει τη συμπεριφορά του συστήματος, είτε με προσομοίωση, είτε με stress testing είτε με οποιοδήποτε άλλο διαθέσιμο τρόπο. Άρα θα πρέπει να διασυνδέεται το μοντέλο συστήματος με μοντέλα ανάλυσης με τη βοήθεια εξωτερικών εργαλείων. Ιδιαιτέρως σημαντικό είναι τα αποτελέσματα των μοντέλων αυτών να ενσωματώνονται στο μοντέλο συστήματος για να υπολογίζονται τα χαρακτηριστικά QoS και να επαληθεύονται οι απαιτήσεις.

Decision support model. Δεδομένου ότι καθίσταται δυνατή η ανάλυση του συστήματος, προστέθηκε ένα ακόμα εξωτερικό σύστημα για τη διασύνδεση με μοντέλο λήψης απόφασης, που ενεργοποιείται για να σχεδιαστούν τα συστατικά στοιχεία του συστήματος ώστε να πληρούνται οι απαιτήσεις. Το εξωτερικό σύστημα έχει εμπλουτιστεί με κανόνες σχεδίασης που έχουν εξαχθεί από τη διαδικασία ανάλυσης του συστήματος και είναι επαναχρησιμοποιήσιμοι από όλους τους σχεδιαστές.

Cost model. Αποφάσεις σχετικές με την ποιότητα υπηρεσίας κατά τη σχεδίαση επηρεάζονται και από παράγοντες κόστους, επομένως εισάγεται η περιγραφή του κόστους κτήσης (TCO), κόστους επένδυσης (CapEx) και λειτουργικού κόστους (OpEx) σε ότι αφορά αυτές τις αποφάσεις.



Σχήμα 6.2: Ενιαίο πλαίσιο MBSD SysML με επίκεντρο το QoS.

Domain-specific design models. Για να μπορεί να εφαρμοστεί το πλαίσιο σε διαφορετικά πεδία και συστήματα, απαιτείται η παροχή εξειδικευμένων επεκτάσεων (domain-specific SysML profiles) προσαρμοσμένων σε αυτά.

Το πλαίσιο για τη σχεδίαση σύνθετων συστημάτων μέσω SysML μοντέλων με έμφαση στην ενσωμάτωση και διαχείριση QoS, εφαρμόστηκε σε δύο διαφορετικά πρακτικά πεδία: α) των μεταφορών σταθερής τροχιάς (transportation domain) και β) ανθρωπο-κεντρικά (cyber-physical human domain). Σε κάθε πεδίο εφαρμογής μελετήθηκε συγκεκριμένο case study (μελέτη περίπτωσης) που αντιπροσωπεύει ένα πραγματικό σύστημα και έχει τα δικά του σχεδιαστικά προβλήματα, συγκεκριμένες απαιτήσεις και στόχους, καθώς και διακριτά δομικά, λειτουργικά, ποιοτικά και οικονομικά χαρακτηριστικά. Μέσω των περιπτώσεων αυτών, αποδεικνύεται η σκοπιμότητα, η εφαρμοσιμότητα και η γενικότητα του προτεινόμενου πλαισίου. Τα πεδία όπου εφαρμόστηκε το σχεδιαστικό πλαίσιο είναι τα ακόλουθα.
α) Λειτουργία συστημάτων μεταφορών σταθερής τροχιάς με συγκεκριμένες συνθήκες άνεσης επιβατών: Το Αττικό Μετρό επιλέχθηκε ως σύστημα σιδηροδρομικών μεταφορών σταθερής τροχιάς (RTS) με σκοπό να σχεδιαστεί, να αξιολογηθεί, να βελτιωθεί και να λειτουργήσει προσφέροντας ποιοτικές και αποδοτικές υπηρεσίες μετακίνησης στους πολίτες. Κατά τη σχεδίαση του συστήματος αυτού, το επίκεντρο ήταν η προδιαγραφή και η εκτίμηση της ποιότητας της άνεσης των επιβατών (passenger comfort) ενώ αναμένουν σε σταθμούς ή βρίσκονται μέσα σε τρένα. Η ποιότητα αυτή κατατάχθηκε σε επίπεδα υπηρεσίας αξιοποιώντας την έννοια του επιπέδου υπηρεσίας (LoS). Το συγκεκριμένο ορίστηκε και υπολογίστηκε κατά τη διαδικασία σχεδίασης του Αττικό Μετρό, προσφέροντας ενδείξεις της αναμενόμενης ποιότητας, όπως την αντιλαμβάνονται οι operators κατά τη λειτουργία του συστήματος. Επιπλέον, το LoS μελετήθηκε και βελτιώθηκε στην περίπτωση που στοιχεία του συστήματος το υποβάθμιζαν. Παράλληλα με τη διαδικασία αυτή, διενεργήθηκε οικονομική ανάλυση για τα λειτουργικά κόστη που αυξάνονται όσο επιτυγχάνεται καλύτερο LoS.
β) Απομακρυσμένη ιατρική παρακολούθηση λαμβάνοντας υπόψη τις απαιτήσεις ποιότητας των ασθενών: Η απομακρυσμένη ιατρική παρακολούθηση ηλικιωμένων ασθενών (REMS) αποτελεί σύστημα τηλεϊατρικής που ανήκει σε μία πιο ευρεία κατηγορία ανθρωπο-κεντρικών συστημάτων (CPHS). Στο REMS,

η ιατρική κατάσταση ασθενών που διαμένουν στο σπίτι τους παρακολουθείται μέσω ασύρματων τεχνολογιών (αισθητήρες, κλπ) από κατάλληλο προσωπικό (ιατροί), που βρίσκεται σε απομακρυσμένη τοποθεσία (νοσοκομείο). Στο σύστημα αυτό είναι κρίσιμη η προδιαγραφή και η μελέτη των αναγκών και απαιτήσεων ποιότητας και κόστους που απορρέουν από τους χρήστες κατά τη διαδικασία σχεδίασης. Οι απαιτήσεις αυτές ποσοτικοποιούνται και κατατάσσονται σε επίπεδα ποιότητας (levels). Το σχεδιαστικό αποτέλεσμα στο οποίο καταλήγει ο σχεδιαστής του συστήματος οδηγεί στην επαλήθευση (ή μη) των επιπέδων αυτών. Στην περίπτωση που δεν καλύπτονται οι ανάγκες και απαιτήσεις του ασθενή, δοκιμάζονται εναλλακτικές σχεδιαστικές λύσεις που οδηγούν στην υιοθέτηση του συστήματος και την ικανοποιητική χρήση του από τον τελικό χρήστη, παρέχοντας του ποιοτικές και οικονομικά συμφέρουσες υπηρεσίες.

Δεδομένων των βασικών επιλογών, στα πλαίσια της ερευνητικής δραστηριότητας της διατριβής για την επίτευξη των αναφερόμενων στόχων προέκυψαν οι ακόλουθες συνεισφορές.

- Επεκτάθηκε η SysML για την ενσωμάτωση απαιτήσεων QoS. Συγκεκριμένα, κατασκευάστηκε το QoS SysML profile, το οποίο περιλαμβάνει στοιχεία σχετικά με απαιτήσεις QoS καθώς και οντότητες για την ικανοποίηση των απαιτήσεων. Αναφέρεται ότι έχουν οριστεί επίπεδα ποιότητας υπηρεσίας (QoS levels) με τα οποία βαθμονομούνται οι απαιτήσεις, και επιπλέον δίνεται η δυνατότητα να ενσωματωθούν τα αποτελέσματα που προκύπτουν από την ανάλυση του συστήματος πίσω στο κεντρικό μοντέλο συστήματος. Με βάση τα αποτελέσματα αυτά, τα επίπεδα QoS πρέπει να επιτευχθούν για να ικανοποιηθούν οι απαιτήσεις.

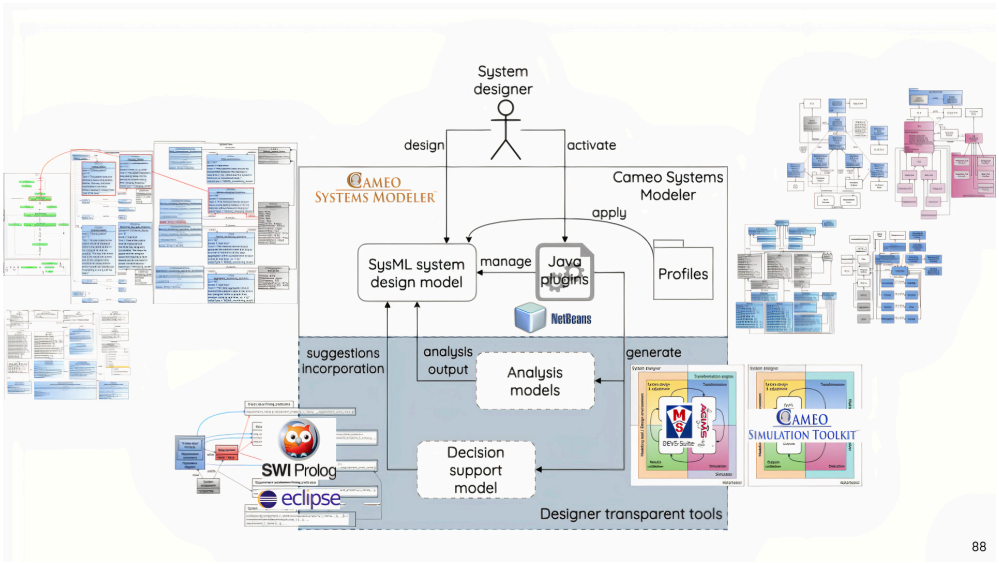
- Ενσωματώθηκαν μοντέλα ανάλυσης (analysis models) στο κεντρικό μοντέλο συστήματος. Κατά την εφαρμογή του προτεινόμενου πλαισίου στα RTS χρησιμοποιήθηκε το πλαίσιο DEVS για την προσομοίωση και παραγωγή αποτελεσμάτων. Στα CPHS συστήματα χρησιμοποιήθηκε η μαθηματική γλώσσα Modelica για την μαθηματική επίλυση και ανάλυση του συστήματος. Το κύριο χαρακτηριστικό είναι ότι τα αποτελέσματα που προέκυψαν είτε με προσομοίωση είτε με μαθηματικά μοντέλα, ανατροφοδοτούνται στο αντίστοιχο κεντρικό μοντέλο συστήματος.

- Υλοποιήθηκε η δυνατότητα αυτοματοποιημένης λήψης σχεδιαστικών αποφάσεων σε SysML μοντέλα. Συγκεκριμένα, κατασκευάστηκε σύστημα λήψης σχεδιαστικών αποφάσεων το οποίο ενεργοποιείται για να σχεδιάσει τα συστατικά στοιχεία συστήματος ώστε να ικανοποιούν συγκεκριμένες QoS απαιτήσεις. Μέσω του συστήματος λήψης αποφάσεων, παρέχονται σχεδιαστικές προτάσεις στο σχεδιαστή για την αυτόματη συμπλήρωση τιμών σε οντότητες του μοντέλου (π.χ. ορισμός κατάλληλης διάταξης (configuration) οντοτήτων) με σκοπό να ικανοποιηθούν ορισμένες απαιτήσεις. Το κύριο χαρακτηριστικό είναι ότι οι σχεδιαστικές λύσεις επανεισέρχονται στο μοντέλο.

- Επεκτάθηκε η SysML για την ενσωμάτωση εννοιών κόστους. Συγκεκριμένα, κατασκευάστηκε το cost profile, ένα γενικό και επεκτάσιμο μοντέλο στη SysML. Περιέχει σχεδιαστικές παραμέτρους κόστους που μπορεί να ανήκουν σε διαφορετικές κατηγορίες όπως κόστος κτήσης (TCO, CapEx και OpEx), καθώς και σε διαφορετικά επίπεδα ανάλυσης, απεικονίζοντας ολόκληρο το σύστημα ή και μέρος αυτού, από άποψη κόστους. Αναφέρεται ότι οι παράμετροι κόστους υπολογίζονται με αντίστοιχες συναρτήσεις που παρέχονται στο σχεδιαστή κατά τη σχεδίαση σε μορφή έτοιμης βιβλιοθήκης συναρτήσεων.

- Υλοποιήθηκαν και παρουσιάστηκαν εφαρμογές του προτεινόμενου πλαισίου και των επεκτάσεων που προσφέρει σε δύο διακριτά πεδία: 'Λειτουργία συστημάτων μεταφορών σταθερής τροχιάς με συγκεκριμένες συνθήκες άνεσης επιβατών' (RTS), και 'Απομακρυσμένη ιατρική παρακολούθηση λαμβάνοντας υπόψη τις απαιτήσεις ποιότητας των ασθενών' (CPHS). Συγκεκριμένα, αξιοποιήθηκαν τα μοντέλα και profiles που κατασκευάστηκαν, για να παραχθούν αντίστοιχα domain-specific profiles για κάθε σύστημα. Στο RTS περιγράφονται δομικές οντότητες όπως τρένα,

σταθμοί, γραμμές, καθώς και απαιτήσεις LoS σχετικά με την άνεση των επιβατών μέσω του profile αυτού. Στο CPHS, περιγράφονται οι δραστηριότητες του συμμετέχοντα, καθώς και οι λειτουργικές και οικονομικές ανάγκες και απαιτήσεις του.



Σχήμα 6.3: Υλοποίηση ενιαίου πλαισίου MBSD SysML με επίκεντρο το QoS.

Όσον αφορά την πρακτική υλοποίηση, το πλαίσιο κατασκευάστηκε με τα εργαλεία μοντελοποίησης και σχεδίασης MagicDraw UML Modeling Tool και Cameo Systems Modeler της No Magic, Inc. Για τα συγκεκριμένα εργαλεία, έχει αναπτυχθεί λογισμικό (plugins) γραμμένο σε γλώσσα προγραμματισμού Java στο εργαλείο Netbeans ώστε να διευκολύνει περαιτέρω τη διαμόρφωση των analysis models και decision support model και να τα εκτελεί. Να τονιστεί ότι ο σχεδιαστής απλά σχεδιάζει το σύστημα που επιθυμεί στα εργαλεία μοντελοποίησης και ενεργοποιεί τα plugins που χρειάζεται. Ολόκληρη η υπόλοιπη πολυπλοκότητα είναι 'κρυμμένη' από τον ίδιο με τα analysis models και decision support model να εκτελούνται στο background, σύμφωνα με την εικόνα 6.3.

Συμπερασματικά, στο πλαίσιο της διατριβής αντιμετωπίστηκε ουσιαστικά το θέμα της αναβάθμισης της διαδικασίας μοντελο-κεντρικής σχεδίασης συστημάτων, λαμβάνοντας υπόψη παραμέτρους QoS. Οι επεκτάσεις που προτάθηκαν και υλοποιήθηκαν δίνουν ένα γενικό, ολοκληρωμένο, ενοποιημένο και εφαρμοσμένο MBSD SysML πλαίσιο, εισάγοντας τα κατάλληλα στοιχεία για την περιγραφή και τη διαχείριση QoS. Το κύριο χαρακτηριστικό είναι ότι τα αποτελέσματα που προκύπτουν από την ανάλυση των μοντέλων ανατροφοδοτούνται στο κεντρικό μοντέλο. Τονίζεται ότι το πλαίσιο αυτό συνεισφέρει στην νέα έκδοση της SysML 2, προτείνοντας διακριτή οντότητα για το μοντέλο συστήματος η οποία διατηρεί τα αποτελέσματα που προκύπτουν από την ανάλυση του συστήματος ξεχωριστά, επιτρέποντας ανατροφοδότηση. Επιπλέον, το πλαίσιο εφαρμόστηκε σε δύο διακριτά πεδία εφαρμογής, τα RTS και CPHS. Στα πρώτα, βοήθησε τους operators τέτοιων συστημάτων να λάβουν στρατηγικές αποφάσεις για την επίτευξη επιπέδων ποιότητας υπηρεσίας (LoS) υπό αποδεκτό λειτουργικό κόστος στο ίδιο περιβάλλον σχεδίασης. Στα CPHS, οι συμμετέχοντες βρίσκονται στο κέντρο της σχεδίασης και τους παρέχεται η δική τους οπτική (human view) για να εκτιμήσουν εάν το σύστημα ικανοποιεί τις λειτουργικές και οικονομικές ανάγκες τους. Γενικά, για διαφορετικά πεδία και συστήματα (πέραν των RTS και CPHS) το προτεινόμενο πλαίσιο μπορεί να εφαρμοστεί σε επίπεδο MBSD και SysML. Τέλος, ο πυρήνας της ερευνητικής δουλειάς έχει δημοσιευθεί ως ανοιχτός κώδικας για να μπορεί να χρησιμοποιηθεί και από άλλους σχεδιαστές SoS για τους δικούς τους κόσμους και μοντέλα με άξονα το QoS.

Curriculum Vitae

Christos Kotronis was born in Preveza (Epirus, Greece) on Arpil 16th, 1992.

In 2014, he received his B.Sc. in Computer Science and Information Systems from the Department of Informatics and Telematics of Harokopio University of Athens, with a "Very Good" mark. In 2016, he received his M.Sc. in Web Engineering from the same department, with an "Excellent" mark. During his post-graduate studies at Harokopio University, he received two honorary scholarships for his academic performance.

Between 2016 and 2021, he was a Ph.D. candidate at the same department under the supervision of Professor Mara Nikolaidou. He was awarded a three-year (2017–2019) Engineering and Technology Sciences scholarship from the General Secretariat for Research and Technology (GSRT) and the Hellenic Foundation for Research and Innovation (HFRI) for conducting his doctoral research. This research, titled "QoS-Aware Model-Based Systems Design Using Systems Modeling Language", proposes a model-based framework, utilizing SysML as a modeling language to explore Quality of Service within system models. Case studies on transportation, e-health, and sensor networks are explored, taking into account the dynamic behavior of such Systems-of-Systems (SoS) under real-time conditions.

His current position is teaching assistant at the Department of Informatics and Telematics of Harokopio University of Athens, where he assisted with the laboratories for the following courses: Simulation, Operating Systems, Object-Oriented Programming, and Distributed Systems (2016-now).

His research interests lie in the field of Systems Engineering, Model-Based Systems Design, Quality of Service, and Systems Modeling Language. He has several publications in international conference proceedings and journals.

The reader is referred to Ch. Kotronis's Linkedin profile for a complete view of his updated extended CV and related information. This can be found at the following URL:
https://linkedin.com/in/christos-kotronis.