HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS

Ph.D. DISSERTATION

Model-Based Enterprise Information System Design: A SysML-based approach

Anargyros T. Tsadimas

ATHENS

January 2018

PHD THESIS

Model-Based Enterprise Information System Design: A SysML-based approach

Anargyros T. Tsadimas

SUPERVISOR: Maria Nikolaidou, Professor

PHD COMMITTEE:

Maria Nikolaidou, Professor
Dimosthenis Anagnostopoulos, Professor
Christos Michalakelis, Assistant Professor

EXAMINATION COMMITTEE

Maria Nikolaidou,
Professor
Informatics and Telematics
Harokopio University

Dimosthenis Anagnostopoulos,
Professor
Informatics and Telematics
Harokopio University

Christos Michalakelis,
Assistant Professor
Informatics and Telematics
Harokopio University

Aphrodite Tsalgatidou,
Associate Professor
Informatics and Telecommunications
National & Kapodistrian University of Athens

Maria Virvou,
Professor
Informatics
University of Piraeus

Thomas Kamalakis,
Associate Professor
Informatics and Telematics
Harokopio University

Iraklis Varlamis,
Assistant Professor
Informatics and Telematics
Harokopio University

Examination date: 9 January 2018

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

Me, the author of this document, **Anargyros T. Tsadimas**, i solemnly declare that:

- I am the owner of the copyrights of this original work and this work does not defame any person, neither offend the copyrights of others.
- I acept that the Library of the Harokopio University can change the contents of this work, deliver this work in electronic format through the Institutional Repository, copy this work using to any format or media and keep more than one copies for maintainability or security reasons.

Dedication

In memory of Triantafyllos

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Maria Nikolaidou for the continuous support of my Ph.D study and related research, for her patience, motivation, and her continuous efforts to find funding for my research. Her guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank Prof. Dimosthenis Anagnostopoulos, member of my advisory committee, for his insightful comments and encouragement, but also for the hard questions which incented me to widen my research, especially in the research area of simulation. Moreover, i would like to thank Asst. Prof. Christos Michalakelis, for his helpful comments and suggestions. A special mention goes to Georgios Karabatzos, greatly missed, that with his positive attitude to life inspired me to be a better scientist and person.

The cooperation with Dr. George-Dimitrios Kapos, Dr. Vassilis Dalakas and Christos Kotronis was excellent. I would like to thank them for their valuable contribution to our common research effort. I would also like to thank Loreta Mitsi especially for taking care of many administrative matters throughout my academic life.

Many thanks to Dr. Georgia Dede, Alexandros Dais, Dr. Nancy Alexopoulou and Dr. Ourania Hatzi because of the beautiful working environment when we shared a common office. Also, i would like to thank Ioannis Meletakis, Christos Sardianos and Ioannis Katakis for the excellent cooperation in our projects and their friendship.

It is a pleasure to thank my friend Dimitris Magdalinos for the experiences we shared in our efforts to be innovative.

I would also like to thank Christos Tsolkas, George Tzoumas and Tasos Spiliotopoulos for the endless conversations about the science and life.

I am grateful to my wife Erika, for her love and support especially at the last years of this effort, where she helped me to prioritize this work in order to be completed. Finally, i would like to thank my mother and sister for their support all of these years.

Contents

List of Figures

List of Tables

Abstract

Evidently, system architecture design is a complex process involving different stakeholders and concerns. When designing Enterprise Information Systems (EISs), both software and network infrastructure architecture should be designed in parallel, ensuring system efficiency, as they are interrelated.

Systems Modeling Language (SysML), initiated by the International Council on Systems Engineering (INCOSE) and the Object Management Group (OMG), is commonly used to support model-based system design. INCOSE is a not-for-profit membership organization that promotes integration and interoperability of methods and tools.

Managing design requirements, when composing systems or System of Systems (SoS), is a complex task, as they should be adapted during system evolution. A systematic review and classification of requirements is necessary in order to reclaim them in the evaluation process. Hence, Non-functional Requirements (NFRs), such as performance ones, should be focused during EIS architecture design, since their key role in system efficiency.

The scope of this research is to provide a model-based approach for EIS architecture design, utilizing SysML as a modeling language. To this end, the system designer is provided with alternative views, focusing software and hardware architecture and facilitating NFRs verification via the definition of a corresponding EIS SysML profile.

Although SysML provides support for requirements specification, corresponding tools lacked an automated requirements verification process. This thesis presents an integrated design environment, not only capable of defining alternative EIS architectures, but also enabling architectural evaluation using simulation. Simulation results are integrated with the system model enabling automated NFR verification process.

Finally, the proposed approach has been successfully tested in other domains such transportations and cost-analysis in the cloud.

SUBJECT AREA: Systems Engineering

KEYWORDS: Model-Based System Design, SysML, Non-functional Requirements, Requirements Verification, Simulation, Model Transformations, MDA.

Περίληψη

Η σχεδίαση της αρχιτεκτονικής των συστημάτων είναι μια πολύπλοκη διαδικασία, στην οποία υπάρχουν πολλοί εμπλεκόμενοι και διάφορετικά θέματα ενδιαφέροντος. Όταν σχεδιάζουμε Εταιρικά Πληροφοριακά Συστήματα (ΕΠΣ) θα πρέπει να σχεδιάζουμε παράλληλα την αρχιτεκτονική του λογισμικού και του δικτύου, μιας και σχετίζονται άμεσα μεταξύ τους, με σκοπό να διασφαλιστεί η αποτελεσματικότητα του συστήματος.

Η SysML, η οποία έχει προταθεί από την INCOSE και το OMG, είναι μια ευρέως αποδεκτή γλώσσα μοντελοποίησης η οποία υποστηρίζει πλήρως τη μοντελο-κεντρική σχεδίαση συστημάτων. Η INCOSE είναι ένας μη-κερδοσκοπικός οργανισμός ο οποίος υποστηρίζει τη διασύνδεση και τη διαλειτουργικότητα μεθόδων και εργαλείων, όσον αφορά τη σχεδίαση συστημάτων.

Η διαχείριση των απαιτήσεων κατά τη σύνθεση συστημάτων αποτελούμενα από υποσυστήματα SoS, είναι επίσης μια πολύπλοκη διαδικασία. Οι απαιτήσεις συγκεντρώνονται και αναπροσαρμόζονται κατά τη διάρκεια όλων των φάσεων της εξέλιξης του συστήματος κι αυτό κάνει ακόμη πιο δύσκολη τη διαδικασία της διαχείρισής τους. Για το λόγο αυτό απαιτείται μια συστηματική μελέτη και ταξινόμηση των απαιτήσεων, ώστε να είναι εφικτή η αξιοποίησή τους κατά τη διαδικασία της αποτίμησης. Ως εκ τούτου, κατα τη σχεδίαση της αρχιτεκτονικής του συστήματος πρέπει να δωθεί έμφαση στις Μη-Λειτουργικές απαιτήσεις (ΜΛΑ), όπως για παράδειγμα αυτές που σχετίζονται με την απόδοση, μιας και αυτές παίζουν καθοριστικό ρόλο στην αποτελεσματικότητα του συστήματος.

Ο σκοπός της παρούσας έρευνας είναι η παροχή μιας μοντελο-κεντρικής προσέγγισης για τη σχεδίαση της αρχιτεκτονικής των ΕΠΣ αξιοποιώντας σαν γλώσσα μοντελοποίησης τη SysML. Για το σκοπό αυτό, ο σχεδιαστής του συστήματος εφοδιάζεται με εναλλακτικές όψεις του συστήματος, οι οποίες εστιάζουν στην αρχιτεκτονική του λογισμικού και του υλικού και διευκολύνουν την επαλήθευση των ΜΛΑ. Αυτό επιτυγχάνεται με τον ορισμό ενός προφιλ ΕΠΣ στη SysML.

Παρόλο που η SysML υποστηρίζει την έννοια των απαιτήσεων, στα σχετικά σχεδιαστικά εργαλεία λείπει ένας αυτοματοποιημένος μηχανισμός επαλήθευσης των απαιτήσεων. Η διατριβή παρουσιάζει ένα ολοκληρωμένο σχεδιαστικό περιβάλλον, στο οποίο όχι μόνο είναι δυνατόν να οριστούν εναλλακτικές αρχιτεκτονικές του συστήματος , αλλά επίσης μπορούν

να αποτιμηθούν χρησιμοποιώντας προσομοίωση. Τα αποτελέσματα της προσομοίωσης ενσωματώνονται στο μοντέλο του συστήματος και με αυτόν τον τρόπο γίνεται εφικτή η διαδικασία της αυτοματοποιημένης επαλήθευσης των ΜΛΑ.

Εν τέλει, η προτεινόμενη προσέγγιση εφαρμόστηκε επιτυχώς σε και σε άλλα πεδία όπως οι μεταφορές και η ανάλυση κόστους στο cloud.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Συστημάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μοντελο-κεντρική Σχεδίαση Συστημάτων, SysML, Μη-Λειτουργικές Απαιτήσεις, Επαλήθευση Απαιτήσεων, Προσομοίωση, Μετασχηματισμοί Μοντέλων, MDA.

Συνοπτική Παρουσίαση της Διδακτορικής Διατριβής

Βασικό εμπόδιο για την αποδοτική επικοινωνία και την ολοκλήρωση ανάμεσα στις διαφορετικές μεθοδολογίες και εργαλεία που αφορούν τη σχεδίαση Πληροφοριακών συστημάτων αποτελεί η υιοθέτηση ανεξάρτητων μοντέλων για την αναπαράσταση του συστήματος. Μια μοντελο-κεντρική θεώρηση για τη μελέτη της τεχνολογίας του συστήματος μπορεί να συνεισφέρει προς αυτή την κατεύθυνση προσφέροντας ένα κεντρικό μοντέλο του συστήματος, το οποίο εξυπηρετεί όλες τις σχεδιαστικές δραστηριότητες στα διαφορετικά επίπεδα πολυπλοκότητας, προωθώντας παράλληλα τη διαλειτουργικότητα.

Ο στόχος της παρούσης έρευνας είναι η δημιουργία ενός εννοιολογικού μοντέλου για την μελέτη της τεχνολογίας των Εταιρικών Πληροφοριακών Συστημάτων, λαμβάνοντας υπόψιν καθιερωμένα πλαίσια και μεθοδολογίες, όπως το ευρέως διαδεδομένο πλαίσιο εταιρικής αρχιτεκτονικής Zachman [1] (Zachman Enterprise Architecture Framework) και τις αρχές που ορίζονται στο πρότυπο 42010 [2] του IEEE. Η παρούσα έρευνα εστιάζει στη σχεδίαση της αρχιτεκτονικής του πληροφοριακού συστήματος, λαμβάνοντας υπόψιν τις οπτικές και τους εμπλεκόμενους (στην πράξη στήλες και γραμμές του πίνακα Zachman) που ορίζονται σύμφωνα με το πλαίσιο εταιρικής αρχιτεκτονικής Zachman. Στόχος της είναι:

- η υιοθέτηση ενός μοντέλου για τη σχεδίαση πλήρους συμβατού με το πλαίσιο εταιρικής αρχιτεκτονικής Zachman και
- η επιλογή μια γλώσσας μοντελοποίησης που να το υποστηρίζει, ώστε να χρησιμοποιηθεί από τους σχεδιαστές συστημάτων

Το προτεινόμενο μοντέλο εξυπηρετεί όλες τις σχεδιαστικές δραστηριότητες, οι οποίες είναι η συγκέντρωση λειτουργικών και μη-λειτουργικών απαιτήσεων, η σχεδίαση της αρχιτεκτονικής των εφαρμογών και του δικτυακού υποστρώματος, η αποτίμηση της απόδοσης των προτεινόμενων λύσεων και βελτίωση των προτεινόμενων λύσεων.

Για την μοντελο-κεντρική σχεδίαση συστημάτων, η υιοθέτηση της Unified Modeling Language (UML) ή της Systems Modeling Language (SysML) ως γλώσσας μοντελοποίησης έχει γίνει αντικείμενο έρευνας από πολλούς ερευνητές και από διαφορετικά πεδία έρευνας, τα

οποία συνήθως καταλήγουν σε κάποιου είδους επέκτασής τους. Η υιοθέτηση της SysML για τη μοντελοποίηση συστημάτων προωθεί την ολοκλήρωση της σχεδίασης συστημάτων με άλλες δραστηριότητες, ειδικά όταν εμπλέκεται ανάπτυξη λογισμικού. Η SysML είναι μια γλώσσα μοντελοποίησης, η οποία υποστηρίζεται από το Object Management Group (OMG), και χρησιμοποιείται για τη σχεδίαση συστημάτων. Παρέχει διαφορετικά διαγράμματα για να περιγράψει τη δομή του συστήματος και τα συστατικά του, και για να καθορίσει τις πολιτικές ανάθεσης πόρων (allocation policies) οι οποίες είναι σημαντικές για τη σχεδίαση και για τον καθορισμό των απαιτήσεων. Η SysML υιοθετήθηκε ως γλώσσα μοντελοποίησης και στην παρούσα έρευνα.

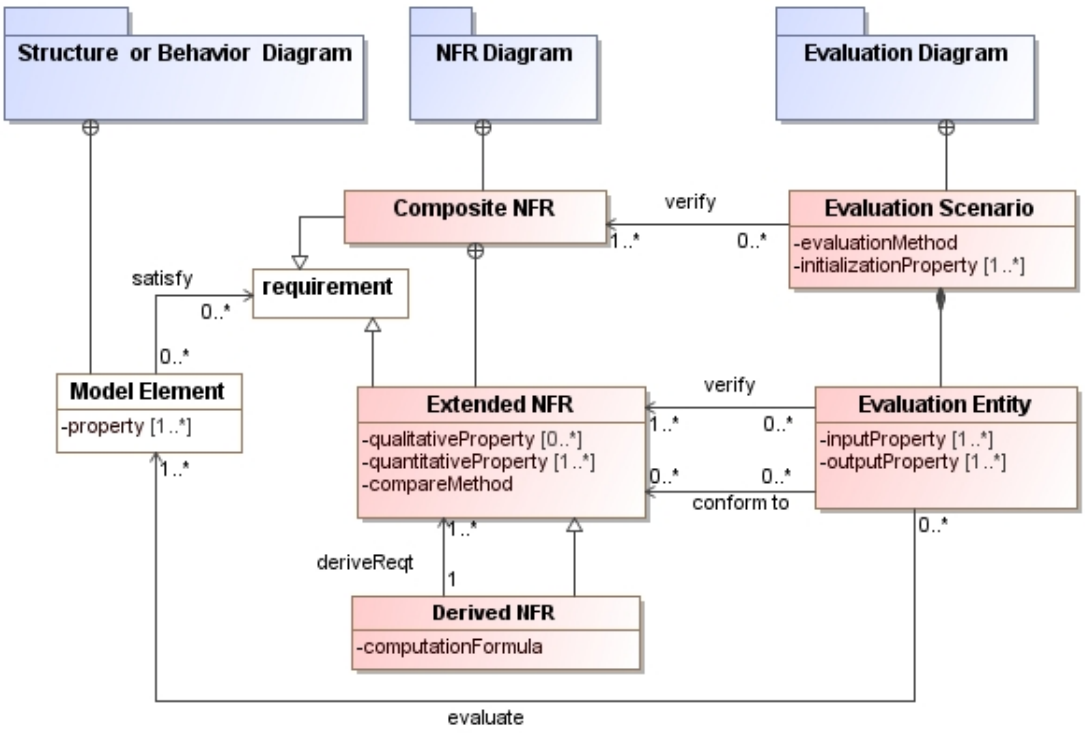
Οι βασικές δραστηριότητες σχεδίασης των συστημάτων είναι ο καθορισμός των απαιτήσεων, η σύνθεση μιας λύσης, η αποτίμηση της προτεινόμενης λύσης και η αναπροσαρμογή της λύσης εφόσον δεν ικανοποιούνται όλες οι απαιτήσεις κατά τη διαδικασία της αποτίμησης. Εφόσον η SysML μπορεί να προσαρμοστεί για ένα συγκεκριμένο πεδίο εφαρμογής (μέσω του μηχανισμού επέκτασης που ορίζεται από τη UML), μπορεί να χρησιμοποιηθεί αποδοτικά κατά τις δραστηριότητες τόσο της σύνθεσης μιας λύσης και της αναπροσαρμογής της, όσο και κατά τη διαδικασία του καθορισμού των λειτουργικών απαιτήσεων. Αυτό που υποστηρίζει η συγκεκριμένη ερευνητική προσπάθεια είναι:

- η αποτελεσματική διαχείριση μη-λειτουργικών απαιτήσεων (ειδικότερα των ποσοτικών) κατά τη διαδικασία της σχεδίασης και
- το «δέσιμο» αυτών των δραστηριοτήτων με τη δραστηριότητα της αποτίμησης της λύσης, που συνήθως γίνεται με τη χρήση άλλων γλωσσών μοντελοποίησης και μεθοδολογιών, με τη χρήση της SysML από το σχεδιαστή για όλες αυτές τις δραστηριότητες.

Παρόλο που οι ΜΛΑ παίζουν ένα σημαντικό ρόλο κατά τη διαδικασία της σχεδίασης, δεν υποστηρίζονται επαρκώς από τη SysML. Επιπρόσθετα, τα εργαλεία που υλοποιούν τη SysML δεν παρέχουν έναν αυτοματοποιημένο τρόπο για την επαλήθευση των μη-λειτουργικών απαιτήσεων. Η έρευνα που πραγματοποιήθηκε εστίασε σε μια επέκταση της SysML ώστε να είναι εφικτή η περιγραφή και η επαλήθευση (verification) των μη-λειτουργικών ποσοτικών απαιτήσεων. Για καταστεί δυνατό αυτό, έγιναν επεκτάσεις στη SysML ώστε:

- να υποστηρίζονται συγκεκριμένες κατηγορίες μη-λειτουργικών απαιτήσεων, και συγκεκριμένα αυτές που βασίζονται σε ποσοτικές παραμέτρους, και
- να υποστηρίζεται η διαδικασία της επαλήθευσής τους

Η SysML προτείνει τη χρήση των test cases προκειμένου να διαπιστωθεί αν μια απαίτηση ή ένα σετ απαιτήσεων επαληθεύονται ή όχι. Στο test case μπορεί να οριστεί ο τρόπος επαλήθευσης, ο οποίος μπορεί να είναι για παράδειγμα ένα διάγραμμα συμπεριφοράς (activity ή state machine διάγραμμα). Όσον αφορά τις ΜΛΑ που σχετίζονται με την απόδοση και εστιάζονται στην παρούσα έρευνα, αυτές περιγράφονται τόσο με ποιοτικά όσο και ποσοτικά χαρακτηριστικά. Όσον αφορά τα ποσοτικά χαρακτηριστικά, μπορεί να χρησιμοποιηθεί



Εικόνα 1: Επέκταση της SysML για την υποστήριξη των ΜΛΑ

κάποια ποσοτική μέθοδος όπως η προσομοίωση, ώστε να γίνει εφικτή η επαλήθευσή τους. Η έννοια του test case δεν υποστηρίζει ποσοτικές μεθόδους επαλήθευσης. Επιπρόσθετα, τα αποτελέσματα της μεθόδου επαλήθευσης θα πρέπει να διατηρούνται στο μοντέλο του συστήματος. Για τους παραπάνω λόγους, προτάθηκε η επέκταση της έννοιας των απαιτήσεων στη SysML με τα εξής:

- Σύνθετες ΜΛΑ οι οποίες αποτελούνται από επεκταμένες μη λειτουργικές απαιτήσεις (οι οποίες μπορεί να προκύψουν από άλλες απαιτήσεις με κάποια μαθηματική σχέση ή με βάση κάποια ευρετική μέθοδο), στις οποίες ορίζονται ποιοτικά και ποσοτικά χαρακτηριστικά καθώς και η μέθοδος σύγκρισης, η οποία χρησιμοποιείται για να κριθεί αν η απαίτηση ικανοποιείται ή όχι, σε σχέση με μια οντότητα του συστήματος.

- *Σενάρια Επαλήθευσης* τα οποία αποτελούνται από οντότητες του συστήματος οι οποίες συμμετέχουν στη μέθοδο επαλήθευσης (συνηθως προσομοίωση) μαζί με τις παραμέτρους εισόδου, έχοντας έτοιμες τις πατραμέτρους οι οποίες θα γεμίσουν με τα αποτελέσματα της προσομοίωσης.

Η Εικόνα 1 παρουσιάζει τις επεκτάσεις της SysML που προτάθηκαν.

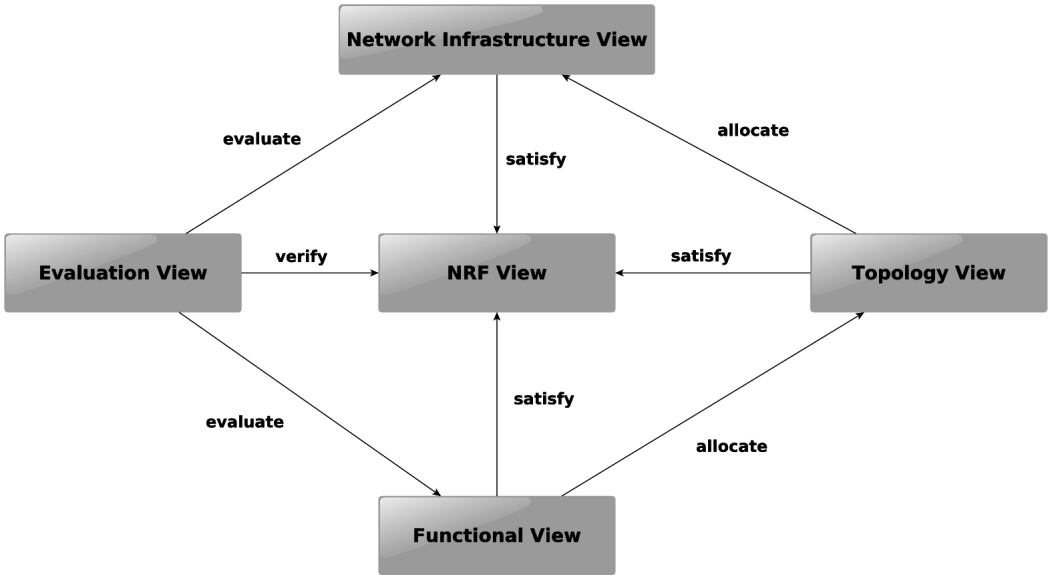
Η προσπάθεια αυτή κατέληξε στην υλοποίηση ενός SysML profile το οποίο ονομάζεται «Προφίλ Εταιρικών Πληροφοριακών Συστημάτων», χρησιμοποιώντας το σχεδιαστικό εργαλείο MagicDraw [3]. Προκειμένου να αποδειχθεί η χρησιμότητα του προφίλ, εφαρμόστηκε σε ένα παράδειγμα χρήσης (case study).

Με βάση το πλαίσιο Zachman και το πρότυπο ISO/IEC 42010 [2], προτάθηκαν συγκεκριμένες όψεις του συστήματος, οι οποίες ανταποκρίνονται στα τέσσερα στάδια της διαδικα-

σίας σχεδίασης των ΕΠΣ, οι οποίες είναι:

- η Όψη της Λειτουργικότητας (Functional View),
- η Όψη της Τοπολογίας (Topology View),
- η Όψη της Υποδομής του Δικτύου (Network Infrastructure View) και
- η Όψη των Απαιτήσεων (Requirements View)

Προκειμένου να υποστηριχθεί η διαδικασία της επαλήθευσης των μη λειτουργικών απαιτήσεων, προτάθηκε η σύσταση μιας νέας όψης του συστήματος - Όψη Αποτίμησης (Evaluation View) - η οποία τηρεί (α) τις ποσοτικές παραμέτρους των μη-λειτουργικών απαιτήσεων μαζί με τις άλλες παραμέτρους των οντοτήτων του συστήματος και (β) τα αποτελέσματα της μεθόδου αποτίμησης του συστήματος (τα οποία ενσωματώνονται ξανά στο μοντέλο του συστήματος), που στην προκείμενη περίπτωση είναι η προσομοίωση, ώστε να καταστεί εφικτή η επαλήθευση των μη-λειτουργικών απαιτήσεων. Η όψη αποτίμησης βασίζεται κι αυτή σε διαγράμματα της SysML. Η όψη αυτή βοηθάει το σχεδιαστή παρέχοντάς του ειδοποιήσεις για τις απαιτήσεις οι οποίες δεν επαληθεύονται καθώς και τις σχετιζόμενες οντότητες με αυτές, ώστε ο σχεδιαστής να κάνει/λάβει τις κατάλληλες ενέργειες/αποφάσεις προκειμένου να οδηγήσει το σύστημα ώστε να ικανοποιήσει όλες τις απαιτήσεις.



Εικόνα 2: Αρχιτεκτονικό μοντέλο των ΕΠΣ

Το αποτέλεσμα της διατριβής είναι ένα ολοκληρωμένο σχεδιαστικό περιβάλλον το οποίο όχι μόνο επιτρέπει τον ορισμό εναλλακτικών αρχιτεκτονικών του υπο-μελέτη συστήματος, αλλά προσφέρει και τη δυνατότητα της επαλήθευσης της εκάστοτε προτεινόμενης αρχιτεκτονικής μέσω προσομοίωσης. Τα αποτελέσματα της προσομοίωσης ενσωματώνονται στο περιβάλλον σχεδίασης, ώστε να αυτοματοποιηθεί η διαδικασία της επαλήθευσης των μη λειτουργικών απαιτήσεων, προσέγγιση η οποία υλοποιήθηκε για πρώτη φορά στα πλαίσια

της παρούσας ερευνητικής προσπάθειας. Τα ανοιχτά ζητήματα, που μελετήθηκαν στο πλαίσιο της διδακτορικής διατριβής συνοψίζονται στα ακόλουθα:

- η πρόταση μιας προσέγγισης η οποία να βασίζεται σε γνωστά πρότυπα και τυπικές γλώσσες όπως η SysML,
- η διευκόλυνση του σχεδιαστή συστήματος στο να εξερευνά εναλλακτικές σχεδιαστικές λύσεις και στην αποτίμηση (evaluation) του μοντέλο του συστήματος πριν την υλοποίηση,
- η εστίαση στον ορισμό και στις συσχετίσεις των μη-λειτουργικών απαιτήσεων απόδοσης του συστήματος,
- η αυτοματοποίηση της διαδικασίας της επαλήθευσης των απαιτήσεων χρησιμοποιώντας τυπικές μεθόδους για εκτίμηση απόδοσης, όπως την προσομοίωση,
- την υλοποίηση της προσέγγισης σε ένα συγκεκριμένο πεδίο εφαρμογής ώστε να αποδειχθεί η σκοπιμότητα μέσω μιας μελέτης περίπτωσης.

Εφόσον το πεδίο εφαρμογής ήταν τα ΕΠΣ, ένας επιπρόσθετος στόχος ήταν η παροχή ενδείξεων της απαιτούμενης ποιότητας υπηρεσίας (Quality of Service (QoS)) των συστατικών στοιχείων λογισμικού ώστε να διευκολυνθεί η διανομή λογισμικού σε υλικό. Αυτό έγινε με τη διερεύνηση των συσχετίσεων των μη-λειτουργικών απαιτήσεων μέσω των συστατικών στοιχείων λογισμικού και της ανάθεσής τους σε συστατικά στοιχεία υλικού, σε καθορισμένες τοπολογίες δικτύων.

Για την επίλυση των ανοικτών ζητημάτων, έγιναν οι ακόλουθες ενέργειες:

- Μια συστηματική βιβλιογραφική έρευνα των μεθοδολογιών που αφορούν στη σχεδίαση των ΕΠΣ [4]
- Μια μοντελο-κεντρική προσέγγιση για το σχεδιασμό ΕΠΣ και ο ορισμός ενός μεταμοντέλου ο οποίος να βασίζεται στη SysML που να εξυπηρετεί το σκοπό αυτό [5, 6]
- Ο ορισμός και η επαλήθευση των μη λειτουργικών απαιτήσεων απόδοσης [7, 8]. Συγκεκριμένα:
  - ο καθορισμός και ο υπολογισμός των παραγόμενων (derived) μη λειτουργικών απαιτήσεων απόδοσης,
  - η διαχείριση των μη λειτουργικών απαιτήσεων μέσω της προσθήκης μιας ξεχωριστής Όψης Αποτίμησης (Evaluation View).
- Η ολοκλήρωση της Όψης Αποτίμησης με ένα εξωτερικό εργαλείο προσομοίωσης [9, 10], μέσω:
  - αρχικοποίησης των εναλλακτικών σεναρίων προσομοίωσης, εκτέλεσης της προσομοίωσης και

  - εισαγωγής των αποτελεσμάτων προσομοίωσης στο μοντέλο της SysML.

- Υλοποίηση ενός ολοκληρωμένου εργαλείου σχεδίασης χρησιμοποιώντας το MagicDraw και τη DEVSJava [9, 11–13], μέσω:

  - της επιλογής ενός Discrete Event Simulator, όπως το Discrete Event System Specification (DEVS),
  - το μετασχηματισμό του μοντέλου της SysML σε μοντέλο DEVS χρησιμοποιώντας την Query / View / Transformation (QVT),
  - τον καθορισμό των βιβλιοθηκών προσομοίωσης των στοιχείων του μοντέλου.

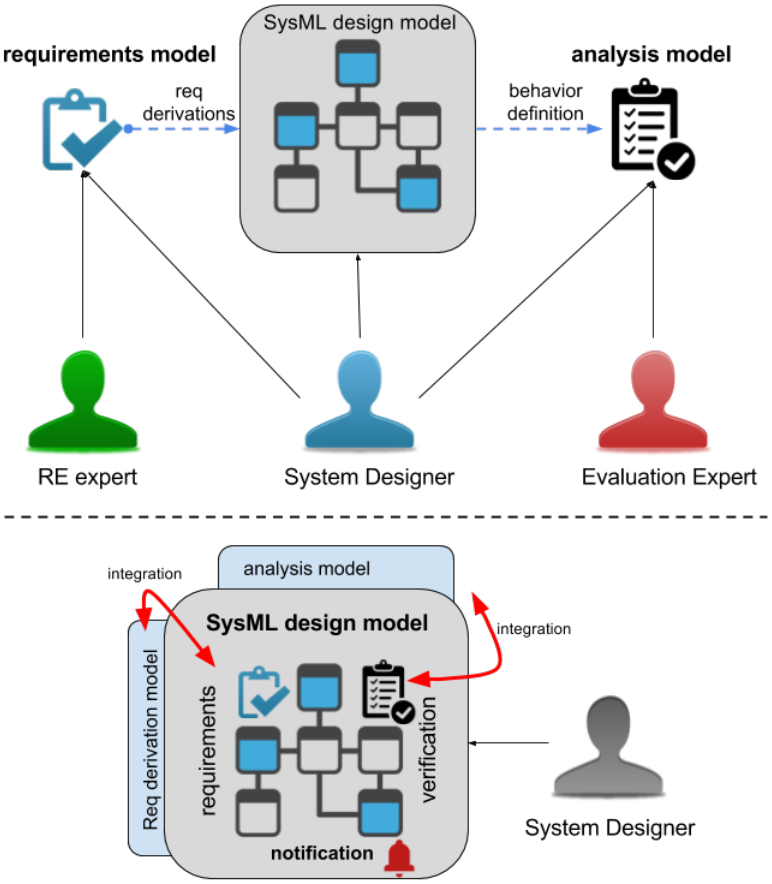
Η συνεισφορά της συγκεκριμένης ερευνητικής προσπάθειας, πέραν του πεδίου εφαρμογής που είναι τα Πληροφοριακά Συστήματα, αφορά τα ακόλουθα:

- τη διαχείριση ποσοτικών μη-λειτουργικών απαιτήσεων ώστε να καθορίζουν τόσο τη συμπεριφορά όσο και την απόδοση ενός συστήματος, το τρόπο ορισμού τους και τον καθορισμό των αλληλεπιδράσεων μεταξύ τους,

- την κατηγοριοποίηση των μη-λειτουργικών απαιτήσεων που αφορούν θέματα απόδοσης,

- την προσθήκη της όψης αποτίμησης, η οποία αφενός διευκολύνει την επαλήθευση των μη-λειτουργικών απαιτήσεων και αφετέρου επιτρέπει την τήρηση ιστορικού των διαφόρων σεναρίων τα οποία δοκιμάστηκαν, σαν μια βάση γνώσης η οποία να διευκολύνει τις αποφάσεις του σχεδιαστή.

Συμπερασματικά, υποκινούμενοι από το γεγονός της έλλειψης αποτελεσματικών μηχανισμών επαλήθευσης των ποσοτικών μη λειτουργικών απαιτήσεων, οι οποίες ορίζονται σε SysML μοντέλα, δόθηκε έμφαση στην λεπτομερή αναπαράσταση των μη λειτουργικών απαιτήσεων, οι οποίες μπορούν να περιγραφούν με ποσοτικά χαρακτηριστικά και στην επαλήθευσή τους χρησιμοποιώντας ποσοτικές μεθόδους όπως η προσομοίωση. Για το σκοπό αυτό επεκτάθηκε η SysML. Η προτεινόμενη προσέγγιση εφαρμόστηκε στο πεδίο των ΕΠΣ, εστιάζοντας στην σχεδίαση της αρχιτεκτονικής τους και στις απαιτήσεις απόδοσης. Το ολοκληρωμένο περιβάλλον σχεδίασης το οποίο υλοποιήθηκε ώστε να υποστηρίζει την συγκεκριμένη προσέγγιση, υπογραμμίζει το ρόλο των μοντέλων και των προτύπων προς την κατεύθυνση προτεινόμενων λύσεων οι οποίες υποστηρίζουν την ανταλλαγή γνώσης και τη συνδυασμένη χρήση μεθόδων και εργαλείων ώστε να διευκολύνουν τον σχεδιαστή του συστήματος παρέχοντάς του ανατροφοδότηση για την απόδοση του συστήματος. Τέλος, σαν επεκτάσεις αυτής της δουλειάς, παρουσιάστηκαν εφαρμογές της μεθοδολογίας σε διαφορετικά πεδία, όπως η τεχνοοικονομική ανάλυση στο cloud και ο ορισμός επιπέδων ποιότητας υπηρεσίας στα συστήματα μεταφορών.



Εικόνα 3: Σύνοψη συνεισφοράς

Preface

During the period of entanglement with my MSc thesis, I had the opportunity to learn about system engineering in a theoretical aspect. To design an information system involves many aspects, such as collecting data from the users (user requirements), designing the architecture, implementing the real system, testing and evolving it.

This thesis, entitled "Model-Based Enterprise Information System Design: A SysML-based approach" deals with the architecture design of EISs. From the first moments till now, many revisions has been done, in order to adopt contemporary trends that affect the design process, such as modeling languages. For example, at the very first steps of this thesis, UML was adopted as the modeling language. However, since SysML was released and became the common language to describe systems and SoS, we consider to use SysML, as more appropriate to describe our approach. The reasons behind this selection are analytically described in the following chapters.

A system designer first tries to depict the system components (subsystems) and the relations between them in a human readable format. Systems, and especially Information Systems (ISs) are implemented through software development process. This process usually is supported by computer software tools, such as requirements definition tools, system design tools, performance evaluation tools, etc. To evaluate a system before its implementation, there is a necessity of a system model, that would be capable of holding all the characteristics related to the evaluation.

In an introductory chapter, an overview of this thesis is presented along with the defined objectives and the contribution. Afterwards, chapter 2 refers on concepts that concern IS in general and especially their design. Definitions of architecture frameworks and model-based system design methodologies are presented to provide the research background of this thesis. By the same token, chapter 3 depicts proposed methodologies for model-based system design, discusses SysML-based approaches, requirements and the way these are involved/handled in this process. In order to clarify the novelty of this thesis, a comparison of the related work is presented.

An overview of the proposed approach about the exploration of EIS architecture design is presented in chapter 4. An EIS profile is defined, that adopts the model-based systems

engineering concepts and defines specific views according to different system perspectives forming the stakeholders' viewpoints. Given these points, our contribution is described in detail, following a two phase approach: on one hand the design phase, presented in chapter 5, is dealing with the specific design views and the definition of NFRs, and on the other hand, the evaluation phase, presented in chapter 6, is performed via the NFRs verification.

Thereupon, to prove the feasibility of the proposed approach, a case study is presented in chapter 7. A critical view about the contribution, pros and cons of the proposed approach is discussed in chapter 8 and finally conclusions and future directions reside in chapter 9.

Throughout this thesis, there were many times that the Ph.D. candidate was wondering about the value of his work. This is a personal fight, through which anyone can obtain not only technical skills but also spiritual ones. A wide range of emotions has been experienced, from enthusiasm, e.g., when a publication was accepted to depression, e.g., when realizing that someone else has achieved something that you are trying to solve. Of course, like everything in life, the route traveled is worthing independently of the destination.

> "You have brains in your head. You have feet in your shoes. You can steer yourself in any direction you choose. You're on your own. And you know what you know. You are the guy who'll decide where to go."
>
> — Dr. Seuss

Chapter 1

Introduction

Contents

1.1 General

When building large-scale ISs, focus is usually given in Systems Engineering (SysE), while the combination of software and hardware, and the way it might affect overall system performance, is often neglected. During system design, software architecture issues are usually dealt with, as a discrete stage of the software engineering methodology applied. The management of related design decisions and the way it might be influenced by NFRs has been explored. Though, software architecture design decisions are influenced by network design, while NFRs, as performance requirements, can usually be satisfied by their effective combination. In practice, both software and network infrastructure architecture should be designed in parallel, as integrated components of the overall system architecture, to efficiently explore their interrelations and ensure NFR satisfaction.

While one can find many efforts in the related literature, the challenge to achieve a more efficient system still remains open. Our proposal is to provide a design environment with evaluation capabilities for EISs design. Requirements provide the means in order to evaluate system's performance, based on their verification. They are used as conditions that have to be satisfied, ensuring that the system would provide the expected behavior. Requirements Engineering (RE) not only refers to the processes of defining, documenting and maintaining requirements but also to the subfields of systems and software engineering concerned with these processes. Under those circumstances, a classification of requirements and their

extensions with quantifiable properties and verification methods is proposed.

1.2 Objectives & Contribution

In the beginning of this thesis, the following goals [12] have been identified as open issues, that formed the objectives of this research:

- to propose an approach based on well-known standards such as SysML and formal languages based on OMG's standards.
- to facilitate the system designer exploring alternative design solutions and evaluating the system model before its implementation.
- to focus in the definition of system requirements.
- to automate requirements verification process using formal methods for performance evaluation, such as simulation.
- to implement the proposed approach in a specific system domain in order to prove the feasibility of the approach by a case study.

Since the selected domain of case study was EIS, an additional goal was to provide indications about the aggregated QoS of software components helping software-to-hardware allocations. Therefore, to meet all these goals, the following steps were accomplished:

i. A review of the EIS engineering methodologies in the literature [14, 15] has been published.

ii. A model-based approach for EIS design and the corresponding metamodel have been proposed [15–17].

iii. A SysML profile for EIS has been defined in [5, 6].

iv. A definition and the verification of NFRs and especially performance ones has been given [15]. Specifically:

   - definition and calculation of derived non-functional requirements has been achieved.
   - management of NFR verification through the proposition of a discrete view, namely Evaluation view is feasible.

v. An integration of the so-called Evaluation view with an external simulation tool [5–7], via:

   - initialization of alternative simulation scenarios and simulation execution, and
   - the integration of the simulation results into the SysML model

   has been implemented.

vi. An implementation of an integrated platform supporting model-based EIS design using MagicDraw and DEVSJava [9, 11, 13], was provided via:

- the selection of a Discrete Event Simulator, such as DEVS
- the transformation of the SysML model into the DEVS model, using QVT
- the definition of the simulation library components of the system model.

The contribution of this thesis will be analytically described at the next chapters.

1.3 Overview

SysE is an interdisciplinary field of engineering that focuses on the way we design and manage complex engineering projects over their life cycles. Model-Based Systems Engineering (MBSE) is a methodology for designing systems using models. A system model is the conceptual model that describes and represents a system [18]. MBSE implies that the models are composed of an integrated set of representations. Tools and methodologies that support MBSE assume that the representations of system behavior and structure are integrated in a single multi-layer model. Each model element can be represented in many views to create a variety of design and architectural representations.

Although UML is a standard modeling language to support MBSE [1], SysML, as an extension of UML, is more appropriate when talking about systems and SoS. On MBSE uses a graphical language to generate and record details pertaining to system's requirements, design, analysis, verification and validation. Additionally, RE refers to the process of formulating, documenting and maintaining software requirements and to the field of Software Engineering (SE) concerned with this process. Specifically, software systems RE is the process of identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation [23]. Requirements are defined throughout phases of system development. A NFR is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. NFR verification is a process that is related to system design.

As mentioned, the scope of this research is to provide a design environment that is capable of defining system architecture, evaluating the system and notify the designer about the non verified non-functional requirements. This process should be transparent to the designer in order to support him to effectively build a system architecture taking into account the imposed hardware and software requirements. As many stakeholders (each of them being a specific engineer) are involved in the system design process, the proposed approach provides separate views, enriched with NFRs, for each stakeholder, as instances of a common model. Figure 1.1 presents an overview of the proposed approach, which is in accordance to INCOSE's objective to *promote integration and interoperability of methods and tools*.

[1] and was used in the early stages of our research [19–22]

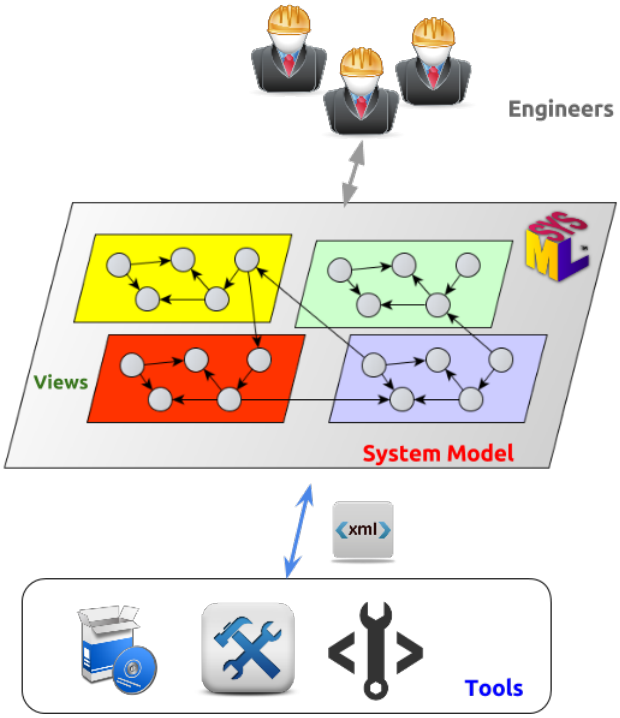


Figure 1.1: Phd Overview

The basic tasks identified during any system design activity are *requirements definition*, *solution synthesis*, *solution evaluation* and *solution re-adjustment* [24]. Based on predefined requirements, the system designer build a solution on system synthesis. In order to decide if a solution is acceptable, evaluation is used. Until an accepted solution is reached, re-adjustments are performed.

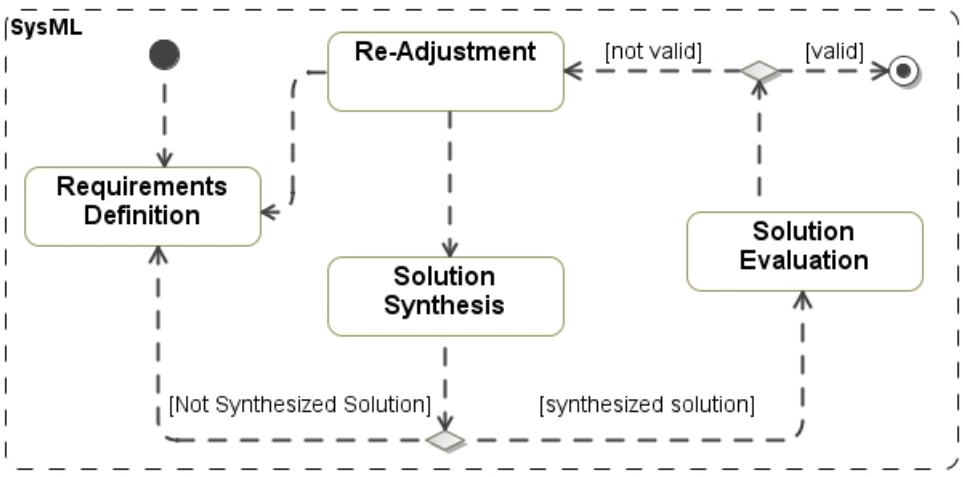


Figure 1.2: Basic System Design Activities

Specifically, to evaluate a system, an analytical method, e.g., mathematical equations describing system behavior, or simulation could be used. In the case of a complex system, such

as ISs, simulation is more appropriate for performance measurements. Hence, a related issue that is addressed also here, is the generation of simulation models based on SysML representations [25], simulation and the incorporation of its results back into the system model. The latter aids the illustration of potential mismatch(es) with the pre-defined requirements, after system evaluation. Automated system evaluation is performed via the verification of NFRs.

1.4 Research Methodology

Through this research, *Design Science* research methodology was used [26]. According to it, there are six different phases that are proposed to follow in order to complete the research. Research is an iterative process that begins with the problem identification and the motivation. The next step is to define the objectives, something already presented at section 1.2. What follows is to design and develop the artifact that supports the objectives and demonstrates how the produced artifact would solve the problem (chapters 4, 5, 6). The evaluation of this research is discussed in chapter 7 with the aid of a case study. Communicating research has been done with publications, mentioned after the references section.

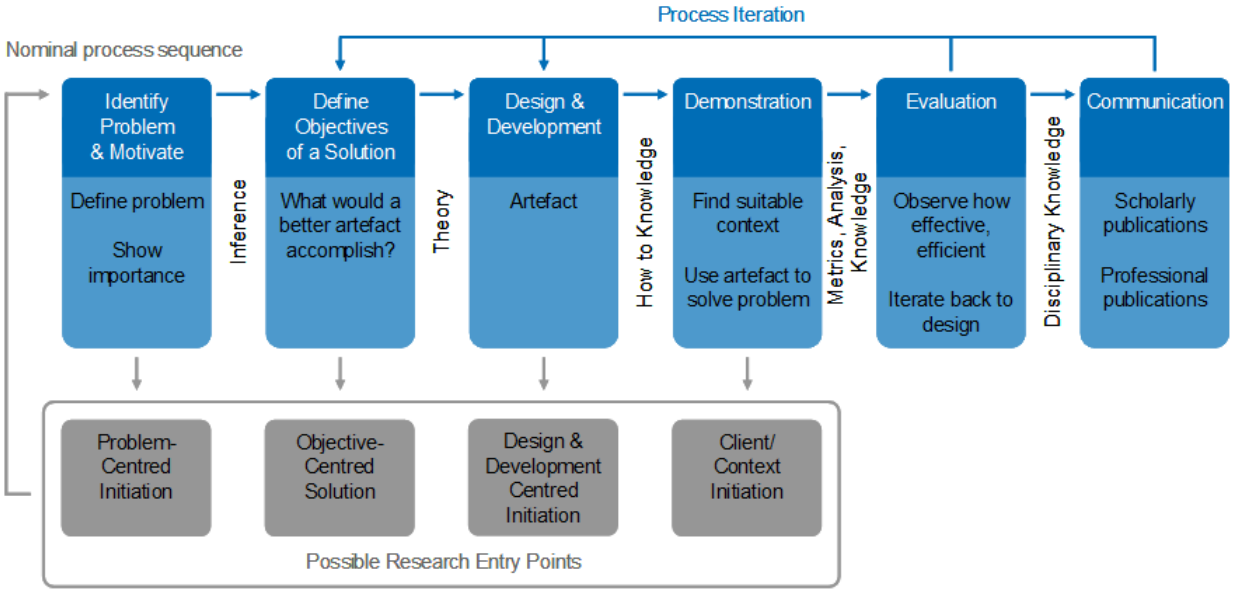


Figure 1.3: Design Science Research Methodology

1.5 Structure

After this short introductory chapter, the thesis is structured as follows. Chapter 2 presents the background in the related research areas of ISs, MBSE and systems evaluation, where selected definitions and standards are briefly presented. The related work, especially the methodologies for model-based EIS design, the defined SysML profiles, the efforts about simulating SysML models, as well as requirements engineering with emphasis on NFRs and their verification, are discussed in chapter 3. Chapter 4 presents our approach to explore EIS architecture design, based on the concepts of MBSE, while 5 lays out the design views, where system designer defines the software and hardware architectures of an EIS. In addition, evaluation view is described in detail in chapter 6, where automated simulation code generation is produced and execution results are incorporated into the system model. To help the reader understand our proposal, its application is presented in chapter 7 with a case study where the proposed approach is applied. There, the design process is given from the system designer's perspective, as a step-by-step procedure, in order to evaluate the designed architecture and verify the requirements. Finally, in chapter 8 a discussion about the proposed approach is presented while conclusions and future work reside in chapter 9.

Chapter 2

Background

Contents

2.1 Outline

This chapter makes an introduction on concepts that concern IS in general and especially their design. Definitions of architecture frameworks and model-based system design methodologies are presented to familiarize the reader with the research area concepts. Introduction on requirements and their classification is briefly depicted. Moreover, modeling languages such as UML and SysML are introduced.

2.2 Information Systems Engineering

A *system* is an artifact created by humans that consists of components or blocks that pursue a common goal that cannot be achieved by each of its single elements. In the context of IS, a block can consist of software, hardware, persons, or any other units [18]. IS is the study

of complementary networks of hardware and software that people and organizations use to collect, filter, process, create and distribute data [27]. Information systems encompasses a variety of disciplines such as:

- the analysis and design of systems,
- computer networking,
- information security,
- database management and
- decision support systems.

SysE concentrates on the definition and documentation of system requirements in the early development phase, the preparation of a system design, and the verification of the system as to compliance with the requirements, taking the overall problem into account: operation, time, test, creation, cost and planning, training and support, and disposal [18]. According to NASA Systems Engineering handbook [28] SysE is a methodical, disciplined approach for the design, realization, technical management, operations, and retirement of a system. SysE integrates all disciplines and describes a structured development process, from the concept to the production then to the operation phase and finally to the system's disposal. It examines both technical and economical aspects in order to develop a system that meets the users' needs. As such, systems engineering stands above specific disciplines, such as software development, for example. This holistic line of thinking can also include solutions to problems that emerge only as a new system is introduced.

Information Systems Engineering (ISE) is the process by which IS are designed, developed, tested, and maintained. The technical origins of information systems engineering can be traced to conventional information systems design and development, and the field of systems engineering. Information systems engineering is by nature structured, iterative, multidisciplinary, and applied. It involves structured requirement analyses, functional modeling, prototyping, software engineering, and system testing, documentation, as well as maintenance [29].

2.2.1 Architecture Frameworks

An architecture framework establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community. Especially the domain within a company or other organization is covered by Enterpsise Architecture Frameworks (EAFs).

Enterprise Systems (ES) are large-scale application software packages that support business processes, information flows, reporting and data analytics in complex organizations. While ES are generally Packaged Enterprise Application Software (PEAS) systems they can also be pre-ordered, custom developed systems created to support a specific organization's needs [30].

An EIS is any kind of information system improving the business processes of an enter-

prise by integration. This typically means high quality of service, related with big data and the capability to support some large and possibly complex organizations or enterprises. An EIS must be used by all parts and all levels of an enterprise [31].

In general, an architecture framework provides principles and practices for creating and using the architecture description of a system. It structures architects' thinking by dividing the architecture description into domains, layers or views, and offers models - typically matrices and diagrams - for documenting each view [32].

An overview of Enterprise Architectures is presented at [33]. This work presents the main characteristics of major architecture frameworks and modeling languages for model-based systems engineering. It states that "a system modeling language such as SysML might therefore evolve not only as a language to describe systems on a high level of abstraction but also as a language to glue heterogeneous models together".

The way an enterprise architecture is created and employed is defined by an EAF. EAFs ([34], [32]) are characterized as an attempt to integrate strategies, processes, methods, models and tools towards enterprise information system engineering [35]. There are a lot of EIS engineering methodologies in the literature [24], each of them covering specific EIS engineering aspects. However, in order to integrate all of them in practice, the support of different system models cannot be avoided. In many cases, these models are not compatible, or even not known to others. Details will be presented in the related work chapter.

The desired integration of people, strategies, processes, methods, models and tools could be accomplished by adopting model-based EIS engineering (MB-EISE). In such a case, a central system model must be defined capturing all system requirements and decisions that fulfill them at different levels of abstraction. Since the central system model serves all engineering activities, it should be technology-neutral, multi-layered, modular and composite, facilitating the integration of system sub-models corresponding to different perspectives and their progressive refinement. Relevant methodologies and tools addressing discrete engineering issues may be applied to specific system sub-models.

In [36], the concept of using Zachman framework [1] as the basis for establishing a central EIS model for Model-based Enterprise Information System Engineering (MB-EISE) was introduced. As such, Zachman matrix serves as a canvas to integrate different concerns, issues and methods towards MB-EISE, while specific methods may use parts of it as a reference point. We also identified some basic guidelines individual model-based methodologies should fulfill, in order to be integrated into the Zachman matrix, focusing on how to establish the EIS sub-model corresponding to each of them. In an effort to practically apply these concepts, in a large scale organization, it became clear that the process of effectively forming the central EIS model was a complex one. In addition, a key obstacle identified was the lack of a common understanding about the purpose of the central model by different stakeholders involved in EIS engineering. This resulted in EIS sub-models, which served well individual methods corresponding to them, but had poor interoperability since it was unclear how specific methods should be interrelated.

To further establish the perception of MB-EISE based on Zachman framework, in the following we identify primary EIS engineering activities and explore the way they can be supported by specific Zachman matrix rows and columns resulting in an first level approach describing model-based EIS engineering process. To this end, we propose:

i. a first-level description identifying the primary EIS engineering activities served by Zachman matrix rows.

ii. a conceptual model for MB-EISE according to ANSI/IEEE 1471 standard [37], which may assist designers to formulate the central EIS model.

iii. a common, first-level description of MB-EISE activities performed based on each cell-related view. Each of these activities consists of specific tasks that may be implemented by a specific EIS engineering method.

Special attention was paid on defining EIS views and viewpoints for each cell in order to enhance information exchange between them.

To explore the proposed concepts in practice, the System Network cell of the Zachman matrix is used as example, already discussed in [36]. Model-based EIS architecture design is focused in this cell. EIS architecture design activity is described based on common first-level MB-EISE activity model proposed. Identified tasks may contribute to related individual method and tool integration. System Network meta-model is adjusted to support individual EIS architecture design tasks and enhance inter-cell communication. The experience obtained when applying the proposed concepts during the renovation of the legacy system of a public large-scale organization is also discussed in chapter 7.

2.2.2 Requirements Engineering

A requirement specifies the user expectations concerning the behavior of the system. Managing design requirements, when composing systems or SoS, is a complex task, as they should be adapted during system evolution [38, 39]. According to Byrne [40], a requirement denotes a capability or a condition that should be satisfied by the system under study and may be either *functional* (i.e., specifying a function that a system must perform) or *non-functional* (i.e., specifying a condition that a system must achieve).

According to Wymore [41] there are six core categories of system design requirements, most of which, such as performance or cost, are non-functional. These categories are:

- I/O requirement,
- technology requirement,
- performance requirement,
- cost requirement,
- trade-off requirement, and
- system test requirement.

This is evident, since most design decisions depend mainly on the conditions that a system should operate rather than the description of its structure [42]. Thus, alternative design decisions and NFRs imposed to a system should be explored in parallel [43, 44]. Furthermore, system architectures should be evaluated [45] and properly adjusted until all imposed requirements are verified in different levels of detail. For example performance requirements may be defined for the system as a whole or for specific system components.

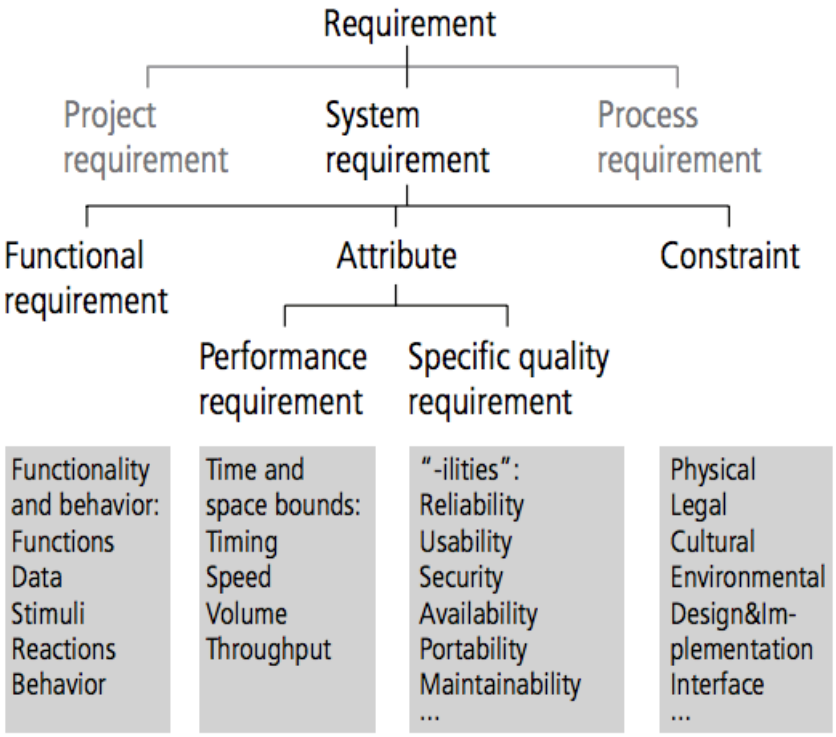


Figure 2.1: A concern-based taxonomy of requirements [1]

Requirements, as stated, are divided into two main categories: *functional* and *non-functional* [40, 46, 47]. NFR is a broadly used term, while there are significant efforts on how to handle them [42]; however, there is no consensus about their nature, since various classifications exist in the literature [40] [47]. NFRs play a significant role during system design, since they depict the conditions under which specific system components should operate, leading to alternative design decisions. A concern-based taxonomy of requirements is presented in [46] and illustrated in Figure 2.1. Non-functional requirements are often called qualities of a system.

2.3 Model-based System Engineering

Model-Based Engineering (MBE) is about elevating models to a central and governing role in the engineering process for the specification, design, integration, validation, and operation of a system [24]. Model-Based System Design (MBSD) is supported by a number of

[1] This taxonomy is presented in [46]

methodologies [24, 48] and is effectively accommodated by SysML [49]. SysML, endorsed by OMG and INCOSE[1], facilitates the description of a broad range of systems and systems-of-systems in a hierarchical fashion, while it is fully supported by most UML modeling tools.

It enables the description of allocation policies and provides a discrete diagram for requirements specification. To describe specific system domains, a SysML profile should be specified, using standard UML extension mechanisms, as stereotypes and constraints [51].

Model-based design of information systems is explored by methodologies such as the ones presented in [4, 52–54]. UML and recently SysML are adopted in all of them as the system modeling language. As indicated in most of them, when building large-scale information systems, software engineering is usually focused, while the combination of software and hardware and the way it might affect overall system performance is often neglected. Design decisions related with software architecture are influenced by network infrastructure design, while NFRs, as performance requirements, can usually be satisfied by effective allocation of software components to hardware. In practice, both software and network infrastructure architecture should be designed in parallel to efficiently explore their interrelations and ensure non-functional requirement satisfaction.

MBSE [55] provides a central system model that captures all system requirements and decisions that fulfill them at different levels of abstraction. The central system model serves all engineering activities. In such a case, a multi-level, composite and technology-neutral central model for EIS should be defined, taking into account different perspectives and aspects of EIS. Existing well-known frameworks may be used for such a purpose.

Leading MBSE Methodologies

Estefan [24] provides a cursory description of some of the leading MBSE methodologies used in industry today. A brief synopsis of each methodology is described in the following paragraphs. These are Object-Oriented Systems Engineering Method by INCOSE and Rational Unified Process for Systems Engineering (RUP-SE) by IBM.

INCOSE Object-Oriented Systems Engineering Method (OOSEM) The Object-Oriented Systems Engineering Method (OOSEM) integrates a top-down, model-based approach that uses SysML to support the specification, analysis, design, and verification of systems. OOSEM is based on object-oriented concepts in conjunction with traditional top down systems engineering methods and other modeling techniques. As such, it enables the systems engineer to precisely capture, analyze, and specify the system and its components and ensure consistency among various system views. The modeling artifacts can also be refined and reused in other applications to support product line and evolutionary development approaches. The

[1]The INCOSE is a not-for-profit membership organization founded to develop and disseminate the interdisciplinary principles and practices that enable the realization of successful systems. Its mission is to share, promote and advance the best of systems engineering from across the globe for the benefit of humanity and the planet [50].

methodology part of OOSEM has since evolved into the Rational Unified Process (RUP). The activities depicted in Figure 2.2 are consistent with typical systems engineering "Vee" process that can be recursively and iteratively applied at each level of the system hierarchy.

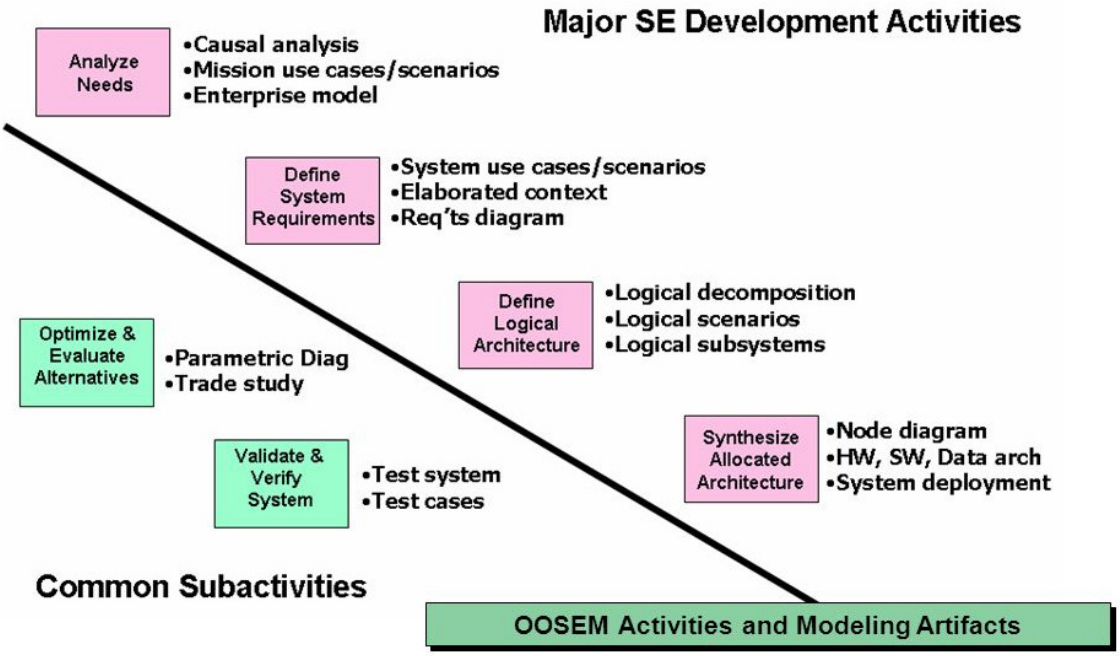


Figure 2.2: OOSEM Activities and Modeling Artifacts

IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD) RUP is a methodology that is both a process framework and process product from IBM Rational and it has been used extensively in government and industry to manage software development projects [56]. The RUP is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of timeboxed iterations. (The Inception phase may also be divided into iterations for a large project.) Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.

Although most iterations will include work in most of the process disciplines (e.g. Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project. RUP-SE was created to specifically address the needs of systems engineering projects. The main content elements of the RUP are the following:

- Roles ("WHO") – A role defines a set of related skills, competencies, and responsibilities.
- Work Products ("WHAT") – A work product represents something resulting from a task, including all the documents and models produced while working through the process.
- Tasks ("HOW") – A task describes a unit of work assigned to a role that provides a meaningful result.

Within each iteration, the tasks are categorized into a total of nine (9) disciplines (Figure 2.3):

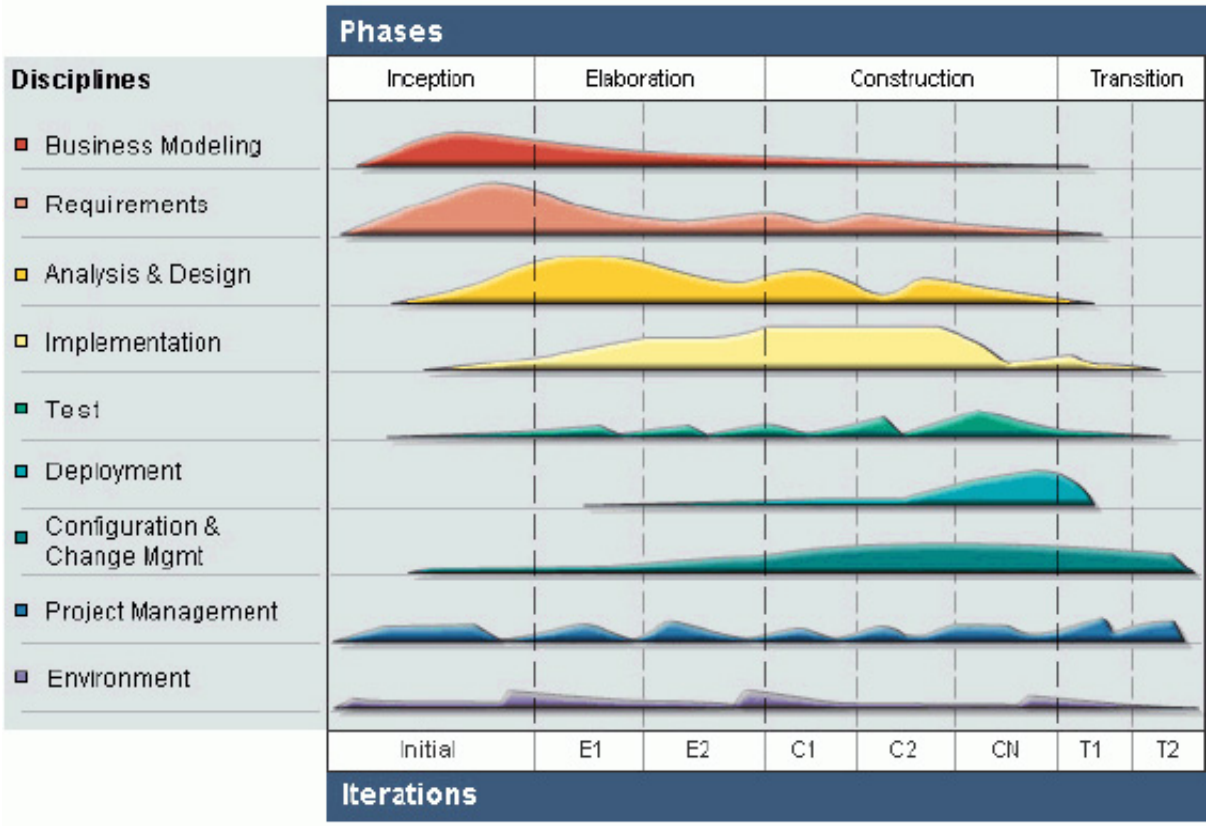


Figure 2.3: The Rational Unified Process framework

Engineering Disciplines:

i. Business modeling
ii. Requirements
iii. Analysis and design
iv. Implementation
v. Test
vi. Deployment

Supporting Disciplines:

i. Configuration and change management
ii. Project management
iii. Environment

2.3.1 Model-based System Design

Design, as a term, provides a structure to any artifact. The idea is to decompose a system into parts, assign responsibilities, ensure that parts fit together to achieve a global goal. Design refers to both an activity and the result of the activity. An activity for example, acts as a bridge between requirements and the implementation of the system. Moreover, it gives a structure to the artifact e.g., a requirements specification document must be designed. A structure helps to better understand the goal. So a design activity refers to the decomposition of a system to subsystems, in order to focus on specific aspects. Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraint [57].

There are four core activities in systems architecture design:

i. *Architectural Analysis* is the process of understanding the environment in which a proposed system or systems will operate and determines the system requirements. The input or requirements to the analysis activity is derived from any number of stakeholders and include items such as:

   - what the system will do when it is operational (the functional requirements)
   - how well the system will perform in runtime (the NFRs)
   - development-time non-functional requirements such as maintainability and transferability
   - business requirements and environmental contexts of a system that may change over time, such as legal, social, financial, competitive, and technology concerns

ii. *Architectural Synthesis* or design is the process of creating an architecture. Given the requirements determined by the analysis, the current state of the design and the results of any evaluation activities, the design is created and improved.

iii. *Architecture Evaluation* is the process of determining how well the current design or a portion of it satisfies the requirements derived during analysis. An evaluation can occur whenever an architect is considering a design decision, it can occur after some portion of the design has been completed, it can occur after the final design has been completed or it can occur after the system has been constructed.

iv. *Architecture Evolution* is the process of maintaining and adapting an existing software architecture to meet requirement and environmental changes. As software architecture provides a fundamental structure of a software system, its evolution and maintenance would necessarily impact its fundamental structure. As such, architecture evolution is concerned with adding new functionality as well as maintaining existing functionality and system behavior.

Moreover, architecture requires critical supporting activities. These supporting activities take place throughout the core software architecture process. They include knowledge management and communication, design reasoning and decision making, and documentation.

2.3.2 System Models Management

MBSE, as previously stated, is based on system models. Methods and tools supporting the design process are using viewpoints and for each viewpoint a corresponding model is defined. This subsection presents standards methods, practices, modeling languages and standards to manage system models.

OMG

OMG is an international, open membership, not-for-profit computer industry standards consortium. OMG develop enterprise integration standards for a wide range of technolo-

gies and an even wider range of industries. OMG's modeling standards enable powerful visual design, execution and maintenance of software and other processes. Originally aimed at standardizing distributed object-oriented systems, OMG focuses on modeling (programs, systems and business processes) and model-based standards [58].

The OMG was formed to help reduce complexity, lower costs, and hasten the introduction of new software applications. The OMG is accomplishing this goal through the introduction of the Model Driven Architecture (MDA) architectural framework with supporting detailed specifications.

MDA

MDA is an approach to system development, which increases the power of models in that work. It is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification [59]. The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors. The MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.

A *viewpoint* on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. Here ”abstraction” is used to mean the process of suppressing selected detail to establish a simplified model. The MDA specifies three viewpoints on a system, a *computation independent viewpoint*, a *platform independent viewpoint* and a *platform specific viewpoint*. A viewpoint model or *view* of a system is a representation of that system from the perspective of a chosen viewpoint. A *platform* is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns. Therefore, any supported application by that platform could use them without concern of the platform related functionality. Table 2.1 presents the three viewpoints and the corresponding models that MDA defines.

Figure 2.4 presents the three viewpoints through the system development process, starting from Computation Independent Models (CIMs) that describe business objects and activities independently of supporting systems, to Platform Independent Models (PIMs) that describe how business processes are supported by systems seen as functional black boxes and finally to Platform Specific Models (PSMs) which describe system components as implemented by specific technologies.

Table 2.1: MDA viewpoints and models

Concept	Viewpoint	Model
Computation Independent	Focuses on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined.	A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.
Platform Independent	Focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used.	A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.
Platform Specific	Combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.	A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Of particular importance to MDA are the notions of metamodel and model transformation. Metamodels are defined at the OMG using the Meta-Object Facility (MOF) standard. A specific standard language for model transformation called QVT has been defined by OMG. In that way, MOF provides the canvas in order to define meta-models in PIMs and PSMs and the way to go over between them. MDA is about using modeling languages as programming languages rather than merely as design languages. Programming with modeling languages can improve the productivity, quality, and longevity outlook [60].

One of the main aims of the MDA [60] is to separate design from architecture. As the concepts and technologies used to realize designs and architectures have changed at their own pace, decoupling them allows system developers to choose from the best and most fitting in both domains. The design addresses the functional requirements while architecture provides the infrastructure through which NFRs like scalability, reliability and performance are realized.

[1] source: https://caminao.wordpress.com/system-engineering/models-perspectives/mde/

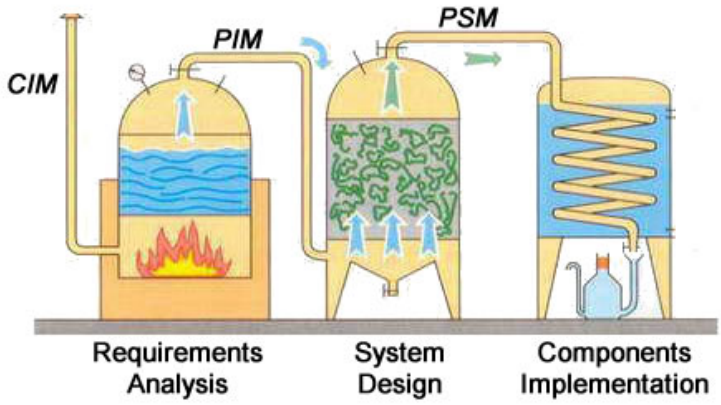


Figure 2.4: A Straightforward Understanding of MDA [1]

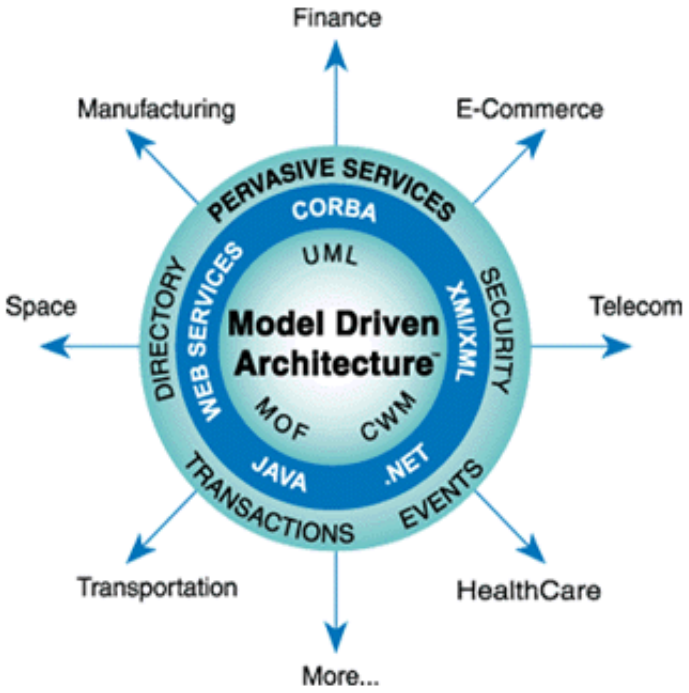


Figure 2.5: MDA [2]

A central and unique model describes aspects and properties of the system. This model can be used to capture the design aspects and the architectural aspects, independently of the concepts and technologies. A PIM in SE a model of a software system or business system, that is independent of the specific technological platform used to implement it.

MDA [59] provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG's established modeling standards: UML; MOF; and Common Warehouse Meta-model (CWM) (depicted in the center of Figure 2.5 and forming the first layer of MDA architecture). Platform-independent application descriptions built using these modeling standards can be realized using any major open or proprietary platform, including CORBA, Java, .NET, and Web-based platforms (forming the second layer). Another standard, XML Metadata Interchange (XMI), allows communication between the pro-

[2] source: http://www.omg.org/mda/mda_audio/mda_rollovers/mda_left_new2.gif

prietary platforms. The third layer contains the services that manage events, security, directories, and transactions. The final layer offers specific frameworks in fields (Finance, Telecommunications, Transportation, Space, Medicine, Commerce, Manufacturing, etc).

UML

The UML is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software during 1994–95, with further development led by them through 1996. In 1997, it was adopted as a standard by the Object Management Group (OMG), and has been managed by this organization ever since. In 2000, the Unified Modeling Language was also accepted by the International Organization for Standardization (ISO) as an approved ISO standard. Since then it has been periodically revised to cover the latest revision of UML. UML defines 13 types of diagrams which are capable of modeling the static and dynamic aspects of a system.

A profile in the UML provides a generic extension mechanism for customizing UML models for particular domains and platforms. Extension mechanisms allow refining standard semantics in strictly additive manner, preventing them from contradicting standard semantics. Profiles are defined using stereotypes, tag definitions, and constraints which are applied to specific model elements, like Classes, Attributes, Operations, and Activities. Specifically, a profile is a collection of such extensions that collectively customize UML for a particular domain (e.g., aerospace, healthcare, financial) or platform (J2EE, .NET).

OCL

Object Constraint Language (OCL) [61] is a formal specification language, part of the UML standard. It is a declarative language for describing rules that apply to UML models. OCL is a key component of the new OMG standard recommendation for transforming models, the QVT specification [62]. OCL supplements UML by providing expressions that have neither the ambiguities of a natural language nor the inherent difficulty of using complex mathematics. OCL is also a navigation language for graph-based models.

QVT

In the model-driven architecture, QVT is a standard for model transformation defined by the OMG. The QVT specification [62] has a hybrid declarative/imperative nature, with the declarative part being split into a two-level architecture:

i. The user-friendly Relations metamodel and language which supports complex object pattern matching and object template creation
ii. A Core metamodel and language is defined using minimal extensions to EMOF and OCL

EMOF stands for Essential MOF and is the part of the MOF 2 specification that is employed for defining simple metamodels using simple concepts.

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 a source model to a target model. To accomplish this, the two models should conform to MOF

SysML

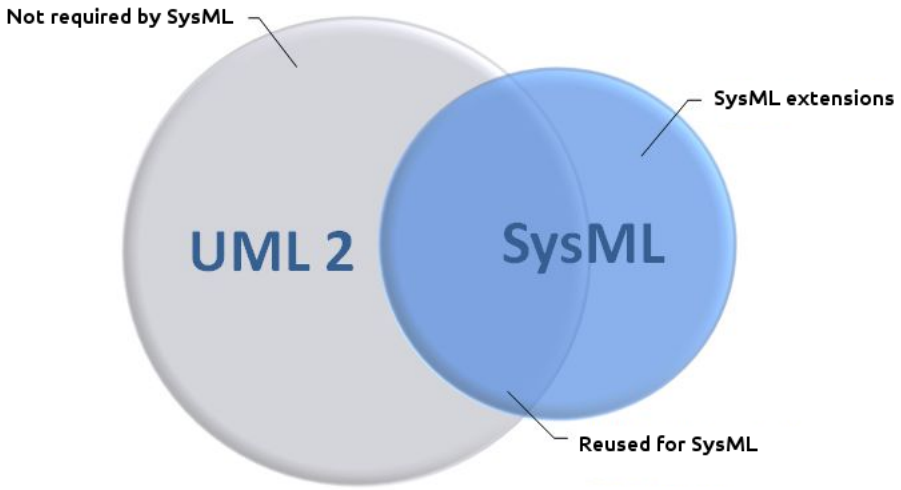


Figure 2.6: SysML and UML [1]

SysML is a general purpose visual modeling language for systems engineering applications. SysML supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities. It was originally developed by an open source specification project, and includes an open source license for distribution and use. In addition, SysML is defined as an extension of a subset of the Unified Modeling Language UML using UML's profile mechanism, as presented in Figure 2.6. Moreover, SysML offers systems engineers several noteworthy improvements over UML, which tends to be software-centric. These improvements include the following [63]:

- SysML's semantics are more flexible and expressive. SysML reduces UML's software-centric restrictions and adds two new diagram types, *requirement* and *parametric* diagrams. The former can be used for *requirements engineering*; the latter can be used for *performance analysis* and *quantitative analysis*.
- SysML is a smaller language that is easier to learn and apply [64]. Since SysML removes many of UML's software-centric constructs, the overall language measures smaller both in diagram types and total constructs.

[1] source: http://wiki.objetdirect.com/wiki/images/7/7a/RelationsSysML_UML2.jpg

- SysML allocation tables support common kinds of allocations. Whereas UML provides only limited support for tabular notations, SysML furnishes flexible allocation tables that support requirements allocation, functional allocation, and structural allocation. This capability facilitates automated verification and validation (V&V) and gap analysis.
- SysML model management constructs support models, views, and viewpoints. These constructs extend UML's capabilities and are architecturally aligned with IEEE Recommended Practice for Architectural Description of Software Intensive Systems (IEEE-Std-1471-2000).

SysML reuses seven of UML 2's fourteen diagrams, and adds two diagrams (requirement and parametric diagrams) for a total of nine diagram types. SysML also supports allocation tables, a tabular format that can be dynamically derived from SysML allocation relationships. A table which compares SysML and UML 2 diagrams is available in the SysML FAQ [65].

The advantages of SysML over UML for systems engineering become obvious if you consider a concrete example, like modeling an automotive system. With SysML you can use Requirement diagrams to efficiently capture functional, performance, and interface requirements, whereas with UML you are subject to the limitations of Use Case Diagram to define high-level functional requirements. Likewise, with SysML you can use Parametric diagrams to precisely define performance and quantitative constraints like maximum acceleration, minimum curb weight, and total air conditioning capacity. On the contrary, UML provides no straightforward mechanism to capture this sort of essential performance and quantitative information.

SysML can be rendered as a domain specific modeling language. Domain specific modeling is a way of how system design and develop. It uses the domain specific language used to represent the different parts of system. SysML was evolved to provide simple and effective constructs to address modeling issues of complex system engineering problems. As SysML reuse subset of UML, it seems to be a good approach to describe its architecture with respect to UML as shown in Fig2.6, where UML and SysML are represented by two intersecting circles [64]

The most important SysML constructs are:

- **Structural**:
  - *Blocks*: the modular unit of structure in SysML that is used to define a type of system, system component, or item that flows through the system. The block not only has structure features like sub-blocks or attributes but also has behavior features including states, activities and operations. A block may be composed of instances of other blocks, called *Parts*.
  - *Ports* are interaction points between a Block or a Part and the environment through which data and signals are exchanged.Ports are Block properties. They are connected with one another with connectors.

  - Package, Block Definition, Internal Block Definition, Parametric Diagrams

- **Behavioral**: Activities support the behavior specification. They take a set of input data to produce a set of output data. (Activity, Sequence, State Machine, Use Case Diagrams)

- **Crosscutting**:

  - *Allocations*: Mapping of elements onto the various structures of a model
  - *Requirements*: Properties that must be satisfied.

Requirements and SysML

Requirements in SysML are described in an abstract, qualitative manner, since they are specified by two properties, *id* and *text*, corresponding to a simple description. However, SysML specification suggests to use the stereotype mechanism to define additional properties for specific requirement categories. Requirements should be satisfied by entities belonging to other diagrams (*SysML satisfy* relation). Requirements are interrelated through a large relationship set, indicating the way they affect each other.

Last but not least, SysML [49] facilitates the description of systems or systems-of-systems for model-based design, providing different system views serving specific design activities. *Block definition diagrams* can be used to depict alternative system design views in multiple layers of detail (for example the software and hardware architecture of an information system). The concept of resource allocation (for example allocating a software component to a hardware component), crucial for system design, facilitates the establishment of relations between such views. Since SysML can be extended or restricted to describe a specific system domain, it may be effectively serve solution synthesis and solution re-adjusted activities, enabling the effective modeling and design of complex systems. So it could act as a canvas for all basic system design activities of Figure 1.2.

2.4 System Evaluation

Focus is given on evaluating systems defined by models. System evaluation is a process that intents to ensure that a specific system meets the defined objectives of its creation and operates as expected. Depending on the system, different evaluation criteria can be tested. Some common criteria are the performance of the system, the security, the availability, the usability etc. To be able to define such criteria, the concept of *requirement* is exploited. In this case, a requirement denotes a condition that should be satisfied by the system under study. Capturing system requirements is not an easy task, and it has to be accomplished by the cooperation of many stakeholders. Requirements engineering is one of the most important phases of the system development process [66].

2.4.1 Requirements Verification

In order to be able to evaluate a system, the defined requirements should be verifiable. To *verify* a requirement means to prove that this requirement has been satisfied. Verification can be done by logical argument, inspection, modeling, simulation, analysis, expert review, test or demonstration [67]. In our case, which are the IS, the verification is based on modeling and simulation. This means that a representation of the system (a system model) will be used to form a simulation model. Model elements *satisfy* requirements, so the verification process checks that all defined requirements are verified.

The NFRs are concerned with QoS. Examples of NFRs are response time, availability and cost. Their verification must be performed using quantitative methods, as for example simulation. Simulation is a widely accepted model-based method to evaluate complex system behavior, especially when non-functional requirements should be verified [24]. Since NFRs (for example performance requirements) are described using both qualitative and quantitative properties, simulation, as a quantitative method, is very effective to produce the necessary data for their verification.

2.4.2 Simulation

Simulation is identified as an appropriate technique for the estimation of system model's performance [68]. The use of formal methods [69] could play the role for testing quantitatively NFRs and especially performance ones if the system model could be expressed in an abstract mathematical model. These methods rely on performing appropriate mathematical analysis to contribute to the reliability and robustness of a design. Evaluating resource allocations policies could rely either on real-time measurements or running simulation on a system model.

Discrete event simulation (DES) is the process of codifying the behavior of a complex system as an ordered sequence of well-defined events [70]. In this context, an event comprises a specific change in the system's state 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.

DEVS is a modular and hierarchical formalism for modeling and analyzing general systems that can be discrete event systems. The DEVS formalism describes a system as a mathematical expression using set theory. It is a theoretically well-defined system formalism [71]. There are two kinds of models in DEVS: atomic and coupled models. An atomic model depicts a system as a set of input/output events and internal states along with behavior functions regarding event consumption/production and internal state transitions. A coupled model consists of a set of atomic models, information of message connections between the atomic models, and input/output ports [72].

In [73,74], 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 [74]. 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 QVT

iv. the generation of DEVS executable code for a DEVS simulation environment with an extensible markup language Extensible Markup Language (XML) interface

2.5 Summary

This chapter outlined the information that is required in order to define the research area in which this thesis contributes. Definitions, architectures defined for ISs, model-based system engineering principles and evaluation techniques were presented. In next chapter, methodologies for model-based system design, SysML-based approaches, simulation of SysML models and efforts on verifying NFRs are discussed.

Chapter 3

Related Work

Contents

3.1 Outline

As related work we consider the work that has been done in proposed methodologies for model-based system design, approaches based on SysML, efforts about simulating SysML models and in the area of RE and especially on how NFRs are verified. Finally, a comparison to related work is presented to make clear what is missing in order to provide an integrated environment for EIS design capable of making performance evaluation through requirements verification.

3.2 Rational Unified Process Methodology

One related methodology for model-based EIS design is RUP-SE [75], that targets information system engineering in RUP and was initially based on UML. During its evolution, it adopted SysML for model-driven information system design [48]. SysML block entities may be employed to describe software, hardware or workers within the system or systems under consideration, while SysML diagrams are used to describe different viewpoints. NFRs are

defined during allocating software to hardware components. In RUP-SE this is accomplished in the context of Joint Realization Tables (JRTs), which are associating logical and distribution views, while NFRs are defined as properties (e.g. table columns) of each specific association. For example, response time requirements can be defined when allocating processes to localities (distribution of enterprise resources). SysML requirement entity, while used to depict functional requirements, is not adopted for NFR description. The description of derived NFRs is also not emphasized. Furthermore, NFR verification is not addressed within the context of RUP-SE.

Model Levels	**Model Viewpoints**					
	Worker	**Logical**	**Information**	**Distribution**	**Process**	**Geometric**
Context	Role definition, activity modeling	Use case diagram specification	Enterprise data view	Domain-dependent views		Domain-dependent views
Analysis	Partitioning of system	Product logical decomposition	Product data conceptual schema	Product locality view	Product process view	Layouts
Design	Operator instructions	Software component design	Product data schema	ECM (electronic control media design)	Timing diagrams	MCAD (mechanical computer-assisted design)
Implementation	Hardware and software configuration					

Figure 3.1: The RUP SE architecture framework

The RUP-SE system architecture framework is deployed in two dimensions, as shown in Figure 3.1. The first dimension defines a set of viewpoints that represent different areas of concern that must be addressed in the system architecture and design. Analytically, *Worker* viewpoint expresses roles and responsibilities of system workers regarding the delivery of system services. *Logical* viewpoint concerns the logical decomposition of the system into a coherent set of UML subsystems that collaborate to provide the desired behavior. *Physical* viewpoint regards the physical decomposition of the system and specification of physical components. *Information* viewpoint focuses on the information stored and processed by the system. *Process* viewpoint examines the threads of control that carry out the computation elements. Lastly, *Geometric* viewpoint denotes the spatial relationship between physical components.

In addition to viewpoints, building a system architecture requires levels of specification, forming the second dimension. As the architecture is developed, it evolves from a general, abstract specification to a more specific, detailed specification. Consistent with RUP guidelines, there are four architectural model levels in RUP-SE, as depicted in Figure 3.1. The level of abstraction at which each model may be constructed, from the more general -hiding or encapsulating detail- to the more specific -exposing more detail and explicit design decisions [76]. Moving down model levels adds specificity, not accuracy, to the models. At each level, you need to be as accurate as possible in specifying model elements, because accuracy at each level adds to the understanding of the system and discipline of the process. Mov-

ing down the levels, each view is a more specific decision, resulting in configuration items at the implementation level. It is important to note that the model elements at one level establish the requirements at the next level. Each model level (Figure 3.1) realizes requirements discovered at a higher level. For example:

- The analysis model level shows how requirements specified in the context model level are met.
- The design model level shows how requirements arising from the system analysis model level are met.
- The implementation model level meets design specifications.

3.3 SysML profiles

In the relative literature, there are many efforts that employ SysML for model-based system design in different domains. Among these, some efforts focus on simulating SysML models [77, 78], while others focus on the verification process [79, 80].

Modeling and Analysis of Real Time and Embedded systems (MARTE) UML profile by OMG [81] supports model-based design of real-time and embedded systems. Non-Functional Properties (NFP) are introduced to specify non-functional quantitative properties (e.g., throughput, delay, memory usage), associated to specific system design entities. MARTE profile focuses on performance and scheduling properties of real-time systems. Non-Functional constraints are introduced to define conditions the NFP should conform to. The Value Specification Language (VSL) is utilized for this purpose formulating semantically well-formed algebraic and time expressions. Defining constraints with VSL enables their automated validation, verification and traceability, using external tools. Requirement association and derivation may also be depicted using NFR constraints expressed in VSL.

MARTE profile, which is based in UML, does not support the notion of requirement that was introduced in SysML. Strategies to apply SysML and MARTE profile, in a complementary manner, were suggested in [82] in a high-level fashion, indicating the potential to combine NFP and VSL expressions defined in MARTE, with SysML requirements for the description of non-functional system characteristics. In any case, NFR verification is left to external tools, although NFR constraints can be useful in identifying the conditions that should be evaluated for this purpose.

A similar approach for NFR description is adopted in User Requirement Notation (URN) standard by International Telecommunication Union (ITU) [83], also supported by a UML profile, where performance characteristics are defined as discrete entities, associated to telecommunication system elements, and described using qualitative parameters.

Syndeia, (recently known as Systems LIfecycle Management (SLIM)) [84] is a commercial collaborative model-based systems engineering workspace that uses SysML as the front-end

for orchestrating system engineering activities from the early stages of system development. The SysML-based system model serves as a unified, conceptual abstraction of the system independent of the specific design and analysis tools that shall be used in the development process. It is designed to provide plugins to integrate the system model (in SysML) to a variety of design and analysis tools. Until now, only the integration of SysML and other model repositories, such as Product Lifecycle Management (PLM) tools is implemented. Integration with MATLAB/Simulink ,Mathematica and OpenModelica is offered in a varierty of commercial tools, but these tools are used as math solvers and not as a verification method of a complete SysML model in a specific domain. The work done in the framework of this thesis shares the vision of Syndeia, but our approach targets to transparency, hiding the analysis tools, like the simulation environment from the designer, making possible to verify system requirements of a SysML model, which is enhanced with a domain specific profile.

Knorreck et al. [85],introduced TEmporal Property Expression language (TEPE), a graphical expression language, which is based in SysML parametric diagrams, representing functional and NFP in a formal way, making them amenable to automated verification. Logical and temporal relations between block attributes and signals. In this work, Automated Verification of reAl Time softwARe (AVATAR) methodology is used to capture requirements, design the system using SysML blocks and behavior described with state machines. Finally formal verification is done. A toolkit, called *TTool* supports these profiles and methodologies. *TTool* is interfaced to verification tools that implement reachability analysis and model-checking. DesIgn sPace exLoration based on fOrmal Description teChniques, Uml and SystemC (DIPLODOCUS), a simulation engine, is integrated in *TTool*, which features the animation and interactive simulation of UML diagrams. This effort is applied in real-time systems. AVATAR improves SysML's capabilities to express time-constraint systems. *TTool* uses *UPPAAL* for formal verification. *IFx toolkit3* [86] which provides simulation and timed-automata based model-checking. OCL is used for well-formed rules. *IFx toolkit3* also provides simulation capabilities within *TTool* framework.

3.4 Simulating SysML Models

Modelica is a standardized general purpose systems modeling language for analyzing the continuous and discrete time dynamics of complex systems based on solving differential algebraic equations. SysML-Modelica Transformation, enables and specifies a standardized bi-directional transformation between the two modeling languages that will support implementations to efficiently and automatically transfer the modeling information transfer between SysML and *Modelica* models without ambiguity [87].

The SysML4Modelica profile endorsed by OMG [87] enables the transformation of SysML models to executable Modelica simulation code. To embed simulation capabilities within SysML, ModelicaML profile is used [88]. QVT is used for the transformation of SysML models

defined using ModelicaML profile to executable Modelica models. A corresponding MOF 2.0 meta-model for Modelica is defined. The overall approach is fully compatible with model-driven engineering concepts, making it suitable of efficient SoS engineering.

In [89], manufacturing line system models are defined in SysML and transformed using ATLAS Transformation Language (ATL) to be simulated using *ARENA* simulation software. With the definition of a SysML profile, Arena-specific properties modeling manufacturing systems are incorporated within SysML block definition and activity diagrams [90]. Corresponding ARENA simulation libraries are incorporated with *ARENA* environment, and properly instantiated to construct the simulation model executed within *ARENA* tool. As far as simulation is concerned only system structure is defined in SysML diagrams. System simulation behavior is defined within *ARENA* manufacturing system libraries. SysML-to-ARENA model transformation is performed using ATL based on model-based software engineering principles, while a corresponding MOF-based meta-model for *ARENA* manufacturing system libraries is defined. The exploitation of simulation output towards system model validation is not discussed.

There are many tools and methods suggested to simulate SysML models and integrate SysML with different simulation languages either for continuous or discrete event simulation [91–93].

3.5 Requirements in SysML

Requirements in SysML are described, as *class* stereotypes, in an abstract, qualitative manner, since they are specified by two properties, *id* and *text*, corresponding to a simple description. Requirements can be grouped in packages based on common characteristics, as their category (for example functional or non-functional) or the activities they are related to (for example software or hardware requirements) forming a multi-level hierarchy.

Regarding requirement definition, SysML provides a discrete diagram to describe requirements and the relations between them, while a set of predefined relations are supported. Furthermore, requirements are explicitly related to system components, which should satisfy them, indicating the functionality they should support or the conditions they should operate in. Moreover, SysML includes a variety of entities to describe requirements and their relation to system components in multiple layers of detail.

Furthermore, SysML provides the means for requirement description and the way to define how the requirements are interrelated, for example which requirement is derived but not how this derivation happens when dealing with quantitative properties of requirements. In [94] an extension on SysML requirements diagram is presented in order to classify and group requirements, but it is too general and does not deal with quantitative requirements and their interrelations with other quantitative ones.

In like manner, SysML includes specific relationships to associate requirements with other requirements (indicating the way they affect each other) or other model elements. The *con-*

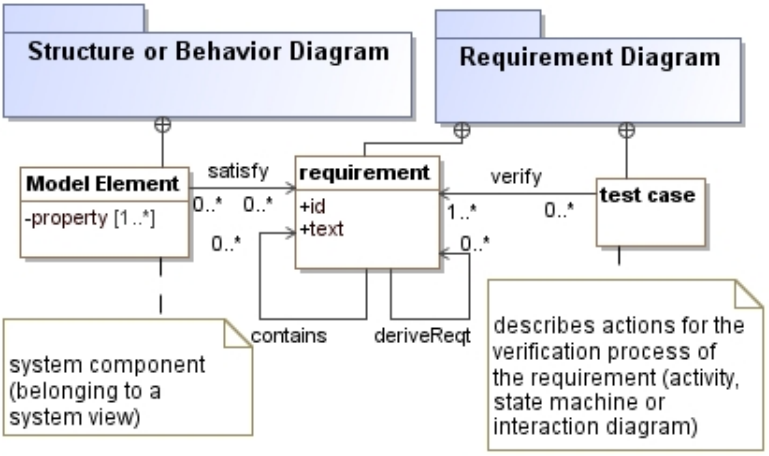


Figure 3.2: SysML Requirement representation

tainment relationship, defined between requirements, indicates that the composite requirement is realized if and only if all the contained ones are realized. In this way, an abstract requirement may be composed of many specific ones, or a complex requirement may be described in a more detailed fashion. In the case of system design, the notion of composite requirements is essential to indicate the way a requirement defined for the whole system may be described in terms of the detailed requirements defined for system components. The *deriveReqt* relationship indicates that a specific requirement is derived by others. Since relationships do not have properties, the way derived requirements are specified is not depicted.

Requirements should be satisfied by model elements belonging to other diagrams (*SysML satisfy* relationship). For this purpose, requirements may participate into other diagrams, enabling the exploration of the relationship between requirements and design decisions.

Additionally, SysML provides the means to describe a set of tests, which should be performed to verify whether a requirement is satisfied by system components. To depict such an activity, the *test case* entity, included in Requirement diagrams, is introduced. A test case is related to one or a set of requirements for their verification, while it is described through a behavior diagram (for example activity or state machine diagram) corresponding to the activity (as a set of tests) performed to verify related requirements. The way requirements are handled in SysML is summarized in Figure 3.2.

Moreover, SysML defines constraint blocks, which provide a mechanism for integrating engineering analysis such as performance and reliability models with other SysML models. Constraint blocks can be used to specify a network of constraints that represent mathematical expressions.

3.6 SysML Requirements Verification

In [79], SysML extensions are proposed for information system design, which are implemented within the context of a custom tool called CASSI. CASSI targets information system

integration, while three different design views are supported, depicted using SysML external and internal block diagrams. The allocation of system components between different views is also supported. SysML requirement entity is not used to associate requirements to system elements. Though, information system configurations defined using CASSI are evaluated using simulation to verify performance and availability requirements. This is accomplished using an external simulator. The behavior of system components is described within CASSI using sequence diagrams, transformed to simulation model by an external transformation tool. Although, NFRs can be verified, this is performed by the system designer using external tools. Evaluation results are not integrated within the SysML system model and NFR verification is not performed using it.

In [95] focus is given on using the SysML4Modelica profile for embedded systems engineering. In the proposed profile, SysML requirement entity is extended with testable characteristics. Testable stereotype may be used for quantitative NFR definition. Testable requirements are associated to conditions under which the requirement is verified with the use of experiments or test cases. Verification conditions are defined as part of a *test case*, which in turn may be simulated using Modelica simulation language in external simulators to ensure that a design alternative satisfies related requirements [80]. Requirement verification is performed in an external modelica tool (MathModelica) through visual diagrams created during simulation. The proposed approach succeeds in converting SysML system models to executable simulation models and enable visual requirement verification. One limitation of this framework is that test cases and requirement verification process are implicitly handled by a domain-specific tool, in this case Virtual Verification of Designs against Requirements (vVDR) [96].

3.7 What is missing?

Table 3.1: A Comparative Overview of SysML Simulation Approaches

Profile	System Domain	Simulator	Profile Characteristics	Transformation language	MDA conformance	Code generation	SysML model validation/ req. verification
MARTE Profile	Real-Time Embedded Systems	Non-Specified	Focus on performance and time requirement description and verification utilizing VSL	Non Specified	Medium	Non specified	Requirement verification performed within SysML models (MDEReqTrace integrates requirement verification data from external tools within SysML)
CASSI Tool	Information Systems	Petri-Nets	Focus on describing performance requirements System behavior is described using Sequence diagrams	Non Specified	Low	Semi automated	Requirement verification performed by an custom external tool
TTool Toolkit	Real-Time Embedded Systems	Y-Chart Timed-Automata	Focus on requirement description using TEPE System behavior is described using State Machine diagrams	Non Specified	Medium	Fully automated	Requirement verification performed by external tools
SysML to Arena Tools	Manufacturing Line Systems	Arena	Focus on the description of the specific domain to incorporate simulation-related characteristics	ATL	High	Fully automated	Not Specified
SysML4 Modelica	General (emphasis on real-time systems)	Modelica	Focus on describing performance requirements System behavior under exploitation is defined as Test Cases using Modelica ML	QVT	High	Fully automated	Requirement verification performed by external tools
DEVSys Framework	General (case study: Information Systems)	DEVS	Focus on embedding simulation output within SysML models System behavior is described using State Machine, Parametric and Activity diagrams	QVT	High	Fully automated	Requirement verification performed within SysML models

As already stated, to perform the requirements verification process, the system model should be able to provide a simulation model/code in an automated manner in order to measure the system's performance. Having in mind existing approaches, it is evident that there is a strong interest in simulating SysML models in an automated fashion to serve SoS engineering and especially SoS design. Since different system domains should be effectively supported, it is expected that different simulation methods and tools will be employed. Though, it is imperative that a standardized methodology/framework, based on OMG standards, should be proposed to guide experts to develop tools targeting specific domains and simulation environments. Most recent approaches seam to follow the same basic steps:

i. Definition of the simulation/domain specific profiles. In this process, efforts should concentrate on defining simulator-specific profiles that may be combined with domain specific profiles. Furthermore, the exploration of a simulator-agnostic profile is suggested for discrete-event and continuous simulators respectively, taking into account that existing approaches utilize the same SysML diagrams.

ii. Transformation of SysML to simulation models in a standardized fashion, utilizing languages as QVT and ATL. Simulator-specific profiles should be accompanied by corresponding MOF-based meta-models for the corresponding simulators. The definition of such meta-models openly available may also promote simulator interoperability. Corresponding initiatives, as those employed by Modelica and DEVS community are already successful.

iii. Utilization of the simulation output to validate SysML models and verify corresponding requirements defined in such models. In order to simplify requirement verification process, we endorse the suggestion of Syndeia to conduct requirement verification within SysML modeling tools, independently of the simulation methods and tools. The incorporation of simulation results within the SysML model should be facilitated for this purpose. Such enhancements simplify the evaluation process, allowing the system designer to focus on the examination of the unverified requirements and, consequently, the detection of the necessary solution re-adjustments.

As derived from the examination of existing approaches, depicted in Table 3.1, there are two key issues in requirements verification during model-based system design that have not been fully addressed:

i. the estimation of system models behavior in a generic and -at the same time- automated manner, and

ii. the designation of the requirements that have not been verified in the original system model.

Regarding the estimation of system models behavior, SysML provides a set of diagrams for describing a single system's behavior (use case, activity, sequence, state machine). How-

ever, each diagram focuses on a different aspect of the system's behavior and the syntax of SysML does not enforce a strict combination of these aspects towards a unified executable behavioral model. On the other hand, simulation profiles for SysML focus on the semantics and structures of specific simulation frameworks, leading to solutions that cannot be applied in general. A systematic approach to assess these issues has not been proposed or adopted yet.

To this end, the details of existing simulation profiles for SysML should be examined thoroughly and processed to derive common concerns and structures. The latter should be further explored against the inherent concepts and attributes of the behavioral SysML diagrams, to conclude to a set of extensions and restrictions for SysML (i.e. a profile) that would enable the general, but conceptually precise and machine-usable definition of the behavior of systems.

Regarding requirements specification, simulation has been identified as an appropriate technique for the estimation of system models' performance. Hence, the obtained simulation results should be incorporated within the original system model and a comparison against the predefined, performance-related, requirements should be performed within the SysML modeling environment. However, many approaches perform requirements verification using external tools, due to acquaintance with them and also due to the lack of quantified requirements handling in the SysML requirements diagram. This thesis introduces the concept of the incorporation of the simulation results into the design environment. Next chapters describe how we reclaim this concept.

In a similar manner as above, approaches proposing solutions for quantified requirements specifications should be examined in detail and in regard with the concepts of different SysML modeling elements (e.g. blocks, states, ports, actions). This would enable the definition of a general profile, capable of defining precise and quantified requirements. Therefore, generic and automated requirements verification within the SysML model could be enabled, once system performance estimation has been added in the model. The proposal of a general profile is out of the scope of this thesis, as we focus on EIS. The interested reader could refer to [97], where a comprehensive understanding of the similarities and differences of existing approaches is presented and identifies current challenges in fully automating SysML models simulation process.

To conclude, in order to enhance the design capabilities of a system architect, requirement verification should be conducted within SysML modeling environment independently of the methods and tools adopted to evaluate alternative system designs. Furthermore, evaluation results should be incorporated within the SysML system model to be utilized by the system designer in alternative design decisions.

3.8 Summary

This chapter presented the related work that has been conducted in the area of model-based system design, and specifically the popular SysML profiles, efforts about simulating SysML models and approaches utilizing verification of requirements for system validation were discussed. Additionally, a comparison of the related work revealed the research challenges that this thesis tries to resolve. Next chapter presents our proposal for a model-based approach for architecture design of EIS. Related work that formed the basis for our approach is also discussed.

Chapter 4

A MBSD Approach for EIS Architecture

Contents

4.1 Outline

This chapter presents an approach to explore EIS architecture design. The proposed approach is based on the concepts of MBSE as defined by INCOSE. There are numerous EIS engineering methodologies in the literature, each covering different aspects. However, in order to integrate them in an Enterprise Architecture, model-based engineering can be adopted. In such a case, a central system model is defined supporting all engineering activities. Zachman's matrix may be used as a basis for constructing such a model. Based on this assumption, we propose a systematic approach for the support of model-based EIS engineering process using Zachman matrix as EIS central model. Our approach is based on these frameworks, showing the concepts of views and viewpoints we adopted and the stakeholders identified throughout the design process. To support the proposed approach, a SysML profile is defined. The profile overview and the defined views are analytically presented in the following sections.

4.2 Using Zachman Framework as a canvas for EIS engineering

Evidently, system architecture design is a complex process involving different stakeholders and concerns [98, 99]. The identification of functional requirements, e.g. software and hardware components and their capabilities [100], are not enough to ensure efficient system operation. Since, as mentioned, NFRs [46] are critical during architecture design [101], thus they should be emphasized. Visualization helps the involved stakeholders to understand and utilize the architecture design decisions [102]. Proposed architecture scenarios should be evaluated [45] and properly adjusted, to achieve an acceptable solution. Discrete architecture design tasks and corresponding stakeholders are served by independent, interrelated views. Each view focuses on a specific design concern and is defined by a corresponding system sub-model, which is part of the overall information system model serving EIS architecture design.

4.2.1 Analysing Zachman matrix

In the following, *views* and corresponding *viewpoints*, *stakeholders* and *concerns* are defined based on the principles of Institute of Electrical and Electronics Engineers (IEEE) 42010 standard [99] and the conceptual model for model-based enterprise information system engineering using the Zachman framework [4]. In Zachman framework, the model focuses on six different perspectives serving discrete primary engineering activities according to Zachman matrix row rationale and six different aspects according to Zachman matrix column rationale (Figure 4.1). Thus, EIS engineering framework consists of 36 EIS views, defined according to the combination of perspectives and aspects. For each EIS view, a viewpoint is defined serving the corresponding stakeholder's perspective on a specific aspect. Each aspect (for example *function*) is treated independently within the limits of the specific engineering activity (for example *design*) based on a corresponding EIS sub-model, while specific methodologies and tools may be applied within EIS viewpoint corresponding to each Zachman matrix cell. For example, RUP methodology [103, 104] could be employed for application design within *System Function* cell. In a similar fashion, system architecture corresponds to the *Structure* aspect of Zachman framework, thus, system architecture design should be treated independently within *System Network* cell.

The Zachman framework itself may provide some guidelines on the dependencies between discrete design activities. In this case, it is evident that there is a need for the exchange of information between software design methodologies, corresponding to *System Function* cell, and software architecture design methodologies, corresponding to *System Network* cell. The exchange of information and the transformation between EIS models serving different activities and aspects may be feasible using the concept of external entities, defined within each EIS model. External entities indicate the required information coming or passed to other methodologies and facilitate EIS models integration. Inter-model consistency is accomplished by creating mappings between external entities of respective models. The cor-

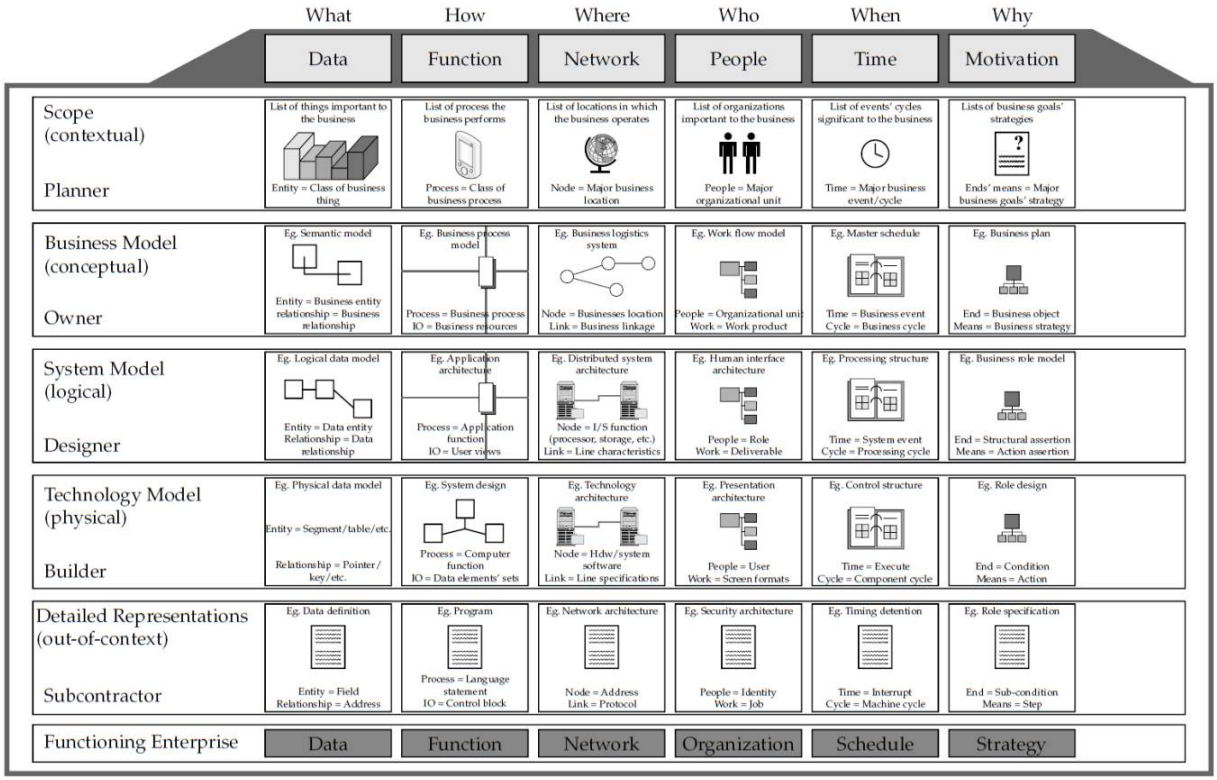


Figure 4.1: The Zachman framework matrix

responding stakeholder is responsible for describing internal entities of each discrete EIS model. Internal and external entities are adopted in order to exchange data between design views (described in chapter 5) and *Evaluation* view (described in chapter 6), in our approach. Since Zachman framework provides a holistic model of enterprise information infrastructure, we argue that each matrix row may serve model-based implementation of a discrete primary engineering activity, as defined in [105] and proposed by INCOSE [98], addressing the needs of corresponding stakeholders (see Figure 4.2).

Here, a brief discussion about the Zachman framework rows and how they relate to EIS is attempted. The first two rows, namely *Scope*, denoting business purpose and strategy, and *Business Model* (Figure 4.1), describing enterprise functionality, are intensively business-oriented and are expressed in business oriented vocabularies [32]. They may serve two discrete primary EIS engineering activities, namely *Defining Enterprise Objectives* and *Establishing Enterprise Functionality* respectively. Definition of *Enterprise Objectives* may comprise to specific activities, such as *Policy Management*, *Enterprise Environment Management*, *Investment* and *Risk Management*, and others characterized in IEEE 15288 as enterprise processes [105]. Establishing enterprise functionality focuses on describing the provided services and corresponding requirements imposed by different stakeholders. The third row, namely *System Model*, which describes how the system will satisfy the requirements yielding from business objectives, may serve *EIS Design* (both at software and hardware level). EIS design facilitates requirements analysis and architecture design of both applications/data and EIS architec-

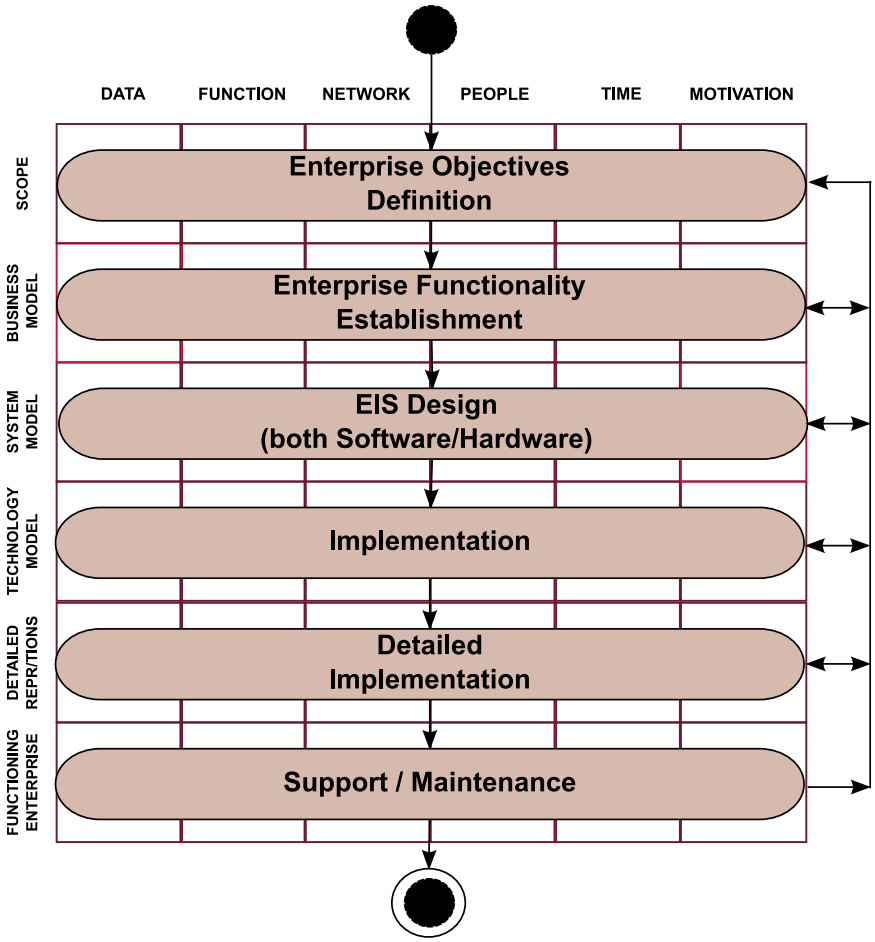


Figure 4.2: MB-EISE primary activities based on the Zachman framework

ture. The next two rows, namely *Builder Model*, representing how the system is implemented and Out-of-Context including implementation-specific details, may serve *Implementation* and *Detailed Implementation* respectively [98]. Finally, the last row, *Operational*, which is the functioning system, may serve *Support and Maintenance* activities, also included in EIS engineering cycle.

All primary engineering activities, as described in Figure 4.2, are interrelated and recursively executed, since EIS engineering is an iterative process targeting the continuous improvement of EIS [98]. Model-driven implementation of these primary engineering activities based on Zachman matrix rows, accommodates the concurrent execution of them based on the EIS sub-model of the corresponding row, provided that they may obtain the information needed by other Zachman matrix rows. Such an approach also facilitates the progressive engineering of EIS in different levels of detail, performed in cumulative cycles. Rules governing the Zachman framework, as defined in [1], are applied during model-based EIS engineering as well. EIS sub-models corresponding to each row are interrelated. The respective requirements are progressively refined, starting from enterprise objectives to the functional EIS supporting it.

Each primary engineering activity should be explored taken into account related requirements identified by the respective stakeholders. A requirement denotes a capability or condi-

tion that must (or should) be satisfied and may specify a function that a system must perform, or a condition that a system must achieve [100]. Thus, requirements are divided into two main categories, i.e. functional and non-functional [40], [47]. In other words, the Zachman matrix consists of six different rows, identifying EIS different aspects, each of which reveals different requirements related to the specific aspect.

We argue that for each primary engineering activity, these six different EIS viewpoints should be defined, each one related to a different EIS aspect. *Data* aspect describes the entities involved, while *Function* viewpoint shows how the entities are processed resulting to application implementation. *Network* viewpoint indicates where the entities are located resulting to EIS architecture. *People* viewpoint indicates users related aspects, while *Time* viewpoint reveals the way identified entities are synchronized. All these viewpoints are used to explore functional requirements, which are related to the functionality of the system.

4.2.2 NFR handling in Zachman matrix

Obviously, NFR is a broadly used term. Unfortunately, there is no consensus about the nature of NFRs since various classifications of them exist in the literature [40] [47]. During this thesis, we follow the common concept that the basic aspects of NFR can be depicted in three sub-categories, namely *performance*, *constraint* and *specific quality* [46]. The *Motivation* row of Zachman matrix relates to the reasons that lead to the specific functionality of an EIS. We argue, thus, that not-functional requirements should be handled by a corresponding viewpoint, as also suggested in [106].

4.2.3 Utilizing Zachman Framework in EIS architecture design

The conceptual model for model-based EIS engineering using Zachman matrix according to ANSI/IEEE 1471 standard [37] is depicted in Figure 4.3. The complete software development process is out of the scope of this research, where focus is on the architecture design. Having this in mind, we are able to frame the latter into Zachman matrix. The activity of designing system architecture is performed based on the EIS view corresponding to *System Network* cell, which facilitates:

i. the definition of EIS architecture (e.g. a system-oriented view of distributed applications),
ii. the definition of system performance and availability requirements,
iii. the definition of system access points,
iv. the description of platform-independent distributed infrastructure (e.g. network architecture and hardware configuration) and
v. the association of software components to network nodes (resource allocation), in order to ensure performance and availability requirements.

The mapping between the EIS architecture design and Zachman framework are based on the following assumptions:

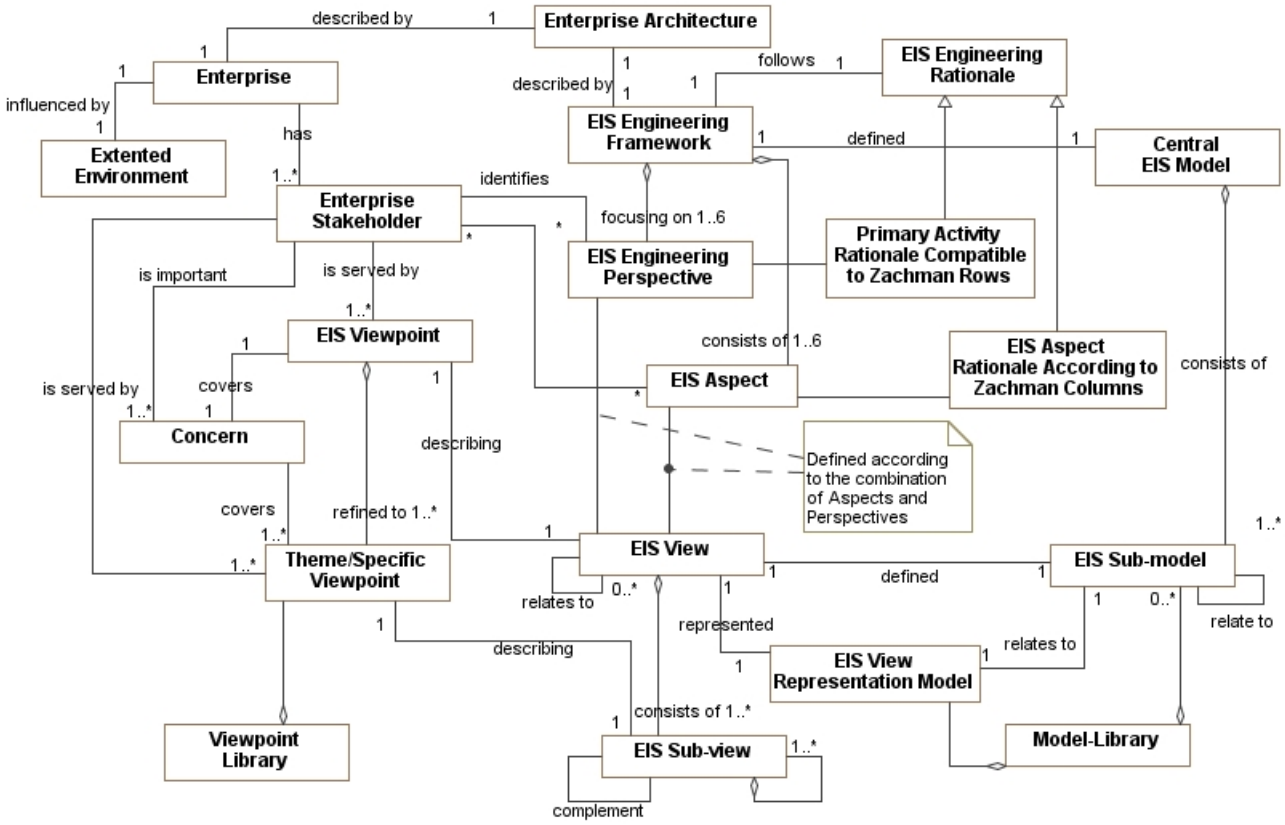


Figure 4.3: MB-EISE conceptual model

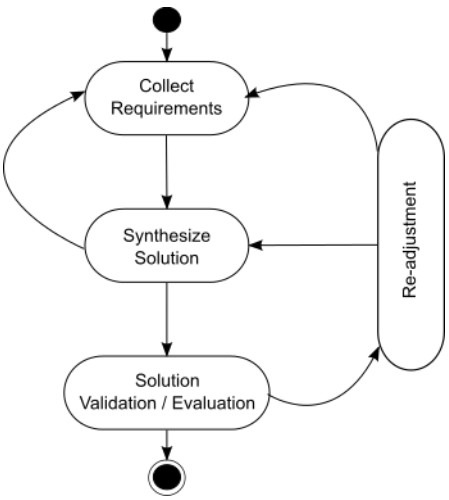


Figure 4.4: Basic engineering tasks performed based on each cell-related view

- The basic tasks identified during the EIS architecture design activity (depicted in Figure 1.2) are in alignment with basic engineering tasks performed for each Zachman matrix cell, as depicted in Figure 4.4

- The *Collect Requirements* task is accomplished through discrete stages i.e. *Functionality Definition* and *Requirements Definition* within *System Network* cell. The former depicts functional requirements extracted from *People*, *Data* and *Function* cells (see Figure 4.1) of the system model row. The latter concerns non functional requirements related to EIS architecture design, extracted from the *Motivation* cell of *System Model* row. Requirements included in this cell are either propagated from the upper layers of Zachman framework or specifically defined for system design and may relate to issues not relevant to EIS architecture design. Only architecture design related requirements are propagated within EIS *System Network* view.

- *Solution* is synthesized through two interactive steps i.e. *Topology Definition* and *Network Infrastructure Definition* [107, 108]. Topology Definition facilitates resource allocation and replication. This task is performed taken into account the definition of system access points in terms of hierarchically related locations performed in upper *Network* cells (*Business* row *Network* cell in particular). The term *site* is used to characterize any location (i.e. a building, an office, etc.) It resembles the term *locality* from RUP-SE. As such, a *site* is a composite entity which can be further analyzed into sub-sites, forming thus a hierarchical structure.

- *Network Infrastructure Definition* refers to the aggregate network, described through a hierarchical structure comprising Local Area Networks (LANs). Devices, such as servers and workstations are associated with LANs at the lowest level of the hierarchy. Network nodes are either workstations allocated to users or computers running server processes. *Topology* and *Network Infrastructure* Definition tasks are interrelated. Both should be performed in the same hierarchical levels of detail. At the lowest level, network nodes should be related to processes/data replicas. In essence, interaction between these two tasks represents an interdependence in terms of derived requirements. *Requirements* derived during *Topology* Definition affect *Network Infrastructure* Definition and vice versa. Therefore, *Requirements Definition* is performed in parallel with *Topology* and *Network Infrastructure* Definition as well. Developing requirements and architectural artifacts in parallel has already been addressed in the literature [109].

- After the solution deployment, *validation* is performed using simulation. *Solution evaluation* will determine whether the overall process will end in case the solution is satisfied or readjustments will be performed through the recurrence of the previous steps.

- The *Requirements Collection* and *Solution Synthesis* tasks are described through a discrete view. As such, four corresponding views are defined, namely:

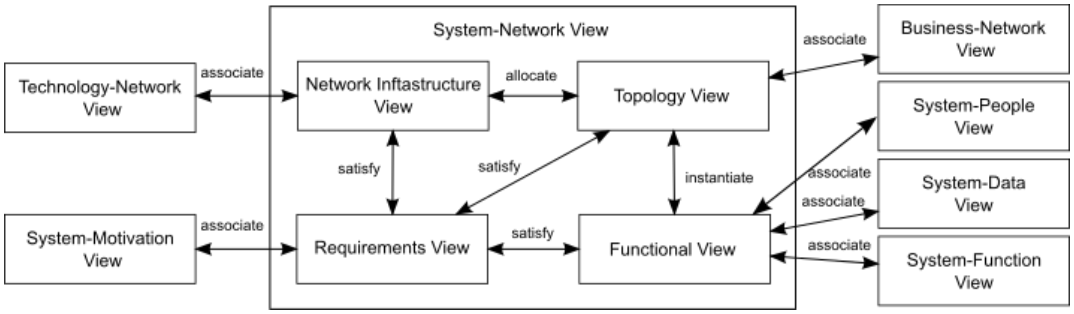


Figure 4.5: EIS Sub-Views corresponding to the System Network cell

i. *Functional* view
ii. *Topology* view
iii. *Network Infrastructure* view
iv. *Requirements* view

These views constitute sub-views of the *System Network* view of Zachman framework. Solution *validation* and *evaluation* are performed using information included in all of them, used to build a discrete view, called *Evaluation* view.

- Interrelations between corresponding tasks are reflected upon the introduced views. These interrelations along with the dependencies between the aforementioned views and the related models of the corresponding Zachman cells are depicted in Figure 4.5. Dependencies with external Zachman cells are bidirectional. *Functional* view, obviously, is influenced by and influences *People*, *Data* and *Function* cells of *System Model* row, while *Requirements* view interacts with *System Motivation* view. *Topology* view is bidirectionally related to *Business Network* cell, while *Network Infrastructure* view to *Technology Network* cell. *System Network* views are illustrated in Figure 4.5 in a blackbox manner. A white-box perspective of them will unfold through the description provided in the following sections, further elucidating view interdependencies.

4.3 Proposed Approach

4.3.1 A conceptual model for Information System Architecture Design

EIS architecture design consists of the definition and optimization of a system architecture comprised of software and hardware components, ensuring that all software components are identified, properly allocated and that hardware components are properly combined to support the efficient operation of software components, providing the desired performance. A conceptual model for model-based EIS architecture design is presented in Figure 4.6, where the model is based on the principles of IEEE 42010 [2] standard. Figure 4.6 also follows the concept of using the Zachman matrix to progressively construct a common system model for the integration of all EIS engineering issues (described in subsection 4.2.3) and correspond-

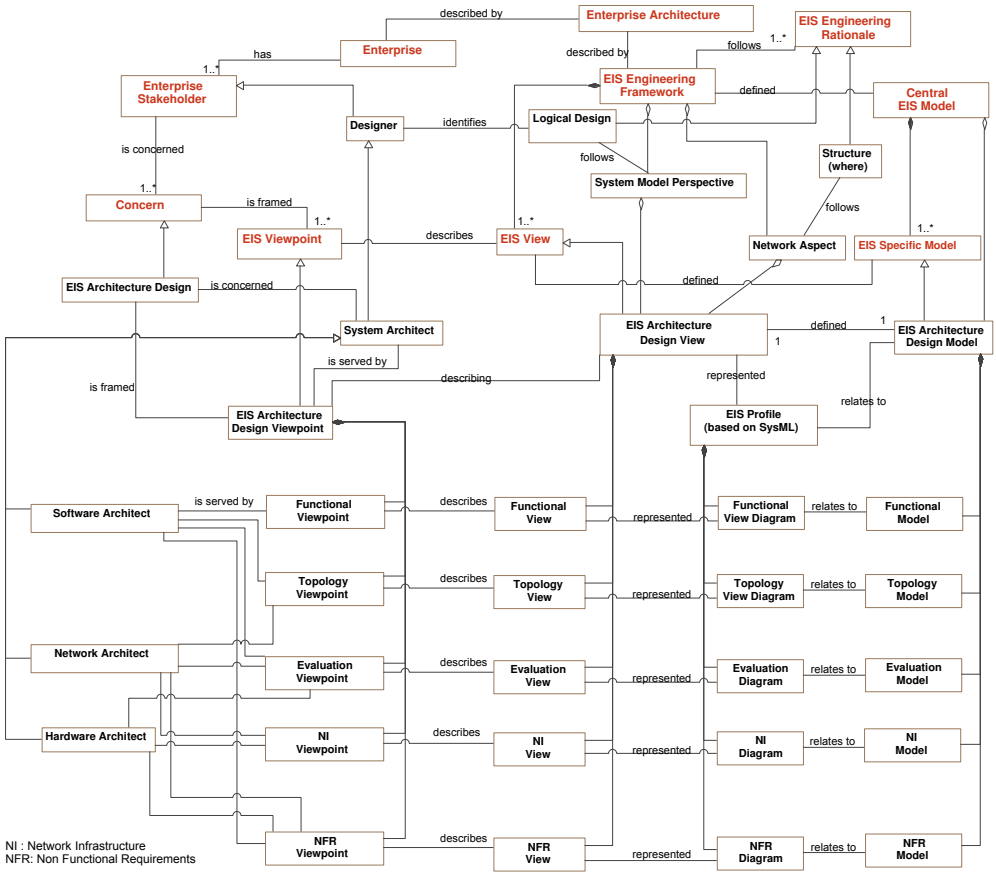


Figure 4.6: A Conceptual Model for Information System Architecture Design

ing methodologies in order to promote interoperability [4].

EIS architecture design view is defined as an EIS view, which is part of the *System Model*, e.g. the perspective serving the *designer* as a stakeholder, and focuses on *Network* aspect emphasizing *Structure* rational. To promote interoperability and integration, it is crucial to provide a typical definition of the meta-model describing each EIS view. Thus a corresponding *EIS architecture design model* is defined for *EIS architecture design* view. This model is a part of the overall system model (indicated in the Figure 4.6 as *Central EIS Model*) and is further decomposed to sub-models corresponding to any sub-views defined for EIS architecture design. For each EIS view, a corresponding representation model should be defined, along with the necessary mappings to EIS view sub-model. A SysML profile, named *EIS profile*, is defined as a representation model for EIS architecture design view and is described analytically in chapters 5 and 6.

As already stated, for each EIS view, a viewpoint is defined serving the corresponding stakeholder's perspective on a specific concern [110]. Thus *EIS architecture design* viewpoint describes *EIS architecture design* view serving *system architect* as a stakeholder, concerned with *EIS architecture design*. Since EIS architecture design is a complex concept, it should be decomposed into more specific concerns dealt with by experts with specific experience, characterized as *system architects*. Thus, the *system architect* as a stakeholder should be conceived more as role than a specific person responsible for system architecture design.

The term *architect* implies that the specific role will be involved in the description of the structure of the information system in some way, though one expert may obtain all these roles, or, more frequently, these roles may be played by experts, for example a software engineer, which are responsible for other tasks as well. As EIS architecture design is decomposed into more specific concerns, corresponding viewpoints forming the overall EIS architecture design viewpoint are explored. In practice, many different experts may contribute in EIS architecture design tasks, as *software architects, network architects and hardware architects*. *Software architect*'s main responsibilities are:

- to limit the choices available during development,
- recognize potential reuse in the organization or in the application,
- subdivide a complex application,
- understand the interactions and dependencies among components and
- communicate these concepts to developers

Network architect is responsible for the design of the distributed architecture of the organization's network. *Hardware architect* is responsible for interfacing with *enterprise architect* or client stakeholders, to determine their needs to be realized in hardware. He/she is generating the highest level of hardware requirements.

Views supporting system design activity tasks

As shown above, the basic tasks identified during any system design activity are *Requirement definition*, *Solution synthesis*, *Solution evaluation* and *Solution re-adjustment* [24] (shown in Figure 4.4). Based on predefined requirements, the system designer build a solution on system synthesis. In order to decide if a solution is acceptable, evaluation is used. Until an accepted solution is reached, re-adjustments are performed.

In the case of EIS architecture design, solution synthesis encompasses *Functionality*, *Topology* and *Network Infrastructure* definitions [4]. *Functionality* definition focuses on software architecture design, *Topology* definition on software allocation process and *Network Infrastructure* definition on hardware architecture design. For each of these concerns, a corresponding viewpoint is defined to explore functional requirements and corresponding design decisions. The *software architect* stakeholder concerned with software architecture design and software allocation is served by *Functional* and *Topology* viewpoints to contribute in the construction of EIS architecture. In a similar fashion, *hardware architect* is contributing to hardware configuration using *Network Infrastructure* view and the *network architect* builds the network architecture based on *Topology* and *Network Infrastructure* viewpoints. In case more than one stakeholders are served by a specific viewpoint, this is an indication that their cooperation is needed.

NFR definition should also be independently treated, since the conditions, under which the system should operate, play a significant role in design decisions. For each of these definitions, a corresponding NFR viewpoint has been defined. In this manner, any of the system

architects (for example software architect) is enabled to realize the affect of specific design decisions (for example the allocation of software to hardware resource) to NFRs imposed to them (for example performance) and vise-versa. Using the corresponding NFR view, the system designer is enabled to explore non-functional requirements relationships, while, using other views, the relationship between non-functional requirements and design decisions is explored [8]. Our approach supports the progressive and independent execution of EIS architecture composition tasks in parallel, while the impact of design decisions is expressed in terms of NFRs.

Furthermore, *EIS architecture evaluation* should be performed. In order to evaluate the designed solution, NFRs definition is used, focusing on system performance and availability requirements essential for EIS architecture design. Then, solution evaluation is performed and evaluation results are used to check whether NFRs are satisfied. If not, then EIS architecture readjustment is performed until an acceptable EIS architecture synthesis is identified. To manage the evaluation process and maintain evaluation results a discrete *Evaluation* viewpoint is defined.

All the aforementioned viewpoints related to EIS architecture design, the corresponding concerns and stakeholders are summarized in Table 4.1. They are characterized as internal viewpoints and are typed using italics. Each of these viewpoints is associated with *external* viewpoints, not concerned with EIS Architecture design, but explicitly or implicitly related to it. Thus information exchange or synchronization between corresponding views of internal and external viewpoints is necessary.

Functional viewpoint serves software architect in order to facilitate him/her to design the software architecture. External viewpoints that are related to Functional viewpoint include:

i. *Application Design* viewpoint which serves software designer, which is responsible for the design of a specific application of the organization, and is the person that will communicate his/her ideas with developers.
ii. *Logical Data Model* viewpoint which serves information designer to design the data model and
iii. *Human Interface Design* viewpoint which serves information system analyst to design the human interfaces.

Topology viewpoint serves network and *software architects* in order to do the resource allocation. Topology viewpoint is related to *Business Offices* viewpoint, in order to serve *enterprise architect* to define the access points of the system. *Enterprise architect* is responsible for having a holistic view of the organization's strategy, processes, information and information technology assets. His/her role is to take this knowledge and ensure that the business and IT are in alignment.

Network Infrastructure viewpoint serves *network* and *hardware architects* to design the network infrastructure architecture. They co-operate with solutions architects who compose the solution based on available technology, which is defined on EIS *Technology Architecture* viewpoint. A *solutions architect* is a very experienced architect with cross-domain, cross-functional

and cross-industry expertise. He/she outlines solution architecture descriptions (mainly in Functional analysis), then monitors and governs the implementation.

NFR viewpoint serves network, *hardware* and *software architects* to define NFRs of EIS architecture. They could co-operate with business analyst, which is served by *Design Requirements* viewpoint to define the design of the requirements. *Business analyst* assess business models and their integration with technology.

Evaluation viewpoint serves network, *software* and *hardware architects* to evaluate the solution that has been composed using all other EIS architecture design viewpoints. They decide if the solution is accepted or not, so as re-adjustments to be done.

Table 4.1: EIS Viewpoints

Internal Viewpoints	**External Viewpoints**	**Concern**	**Stakeholder**
Functional		Software Architecture Design	Software Architect
	Logical Data Model	Data Model Design	Information Designer
	Application Design	Software Design	Software Designer
	Human Interface Design	Human Interface Design	IS Analyst
Topology		Resource Allocation	Network Architect Software Architect
	Business Offices	Access Points Definition	Enterprise Architect
Network Infrastructure		Net. Infra. Architecture Design	Network Architect Hardware Architect
	EIS Technology Architecture	Solution Composition (based on available technology)	Solutions Architect
NFR		EIS Architecture NFR Definition	Network Architect Hardware Architect Software Architect
	Design Requirements	Design Requirements Definition	Business Analyst
Evaluation		Solution Evaluation	Network Architect Software Architect Hardware Architect

4.3.2 Supporting the proposed approach

In EIS architecture design, each stakeholder would like to interact with his specific view, showing him the appropriate entities to facilitate him in order to make design decisions. To have a consistent model, an appropriate modeling language should be selected.

As previously stated, SysML supports the specification, analysis, design, verification and validation of a broad range of systems and SoS [64]. It also supports the concepts of requirement definition, requirements management, systems composition and communication and resource allocation, which are vital to depict EIS architecture design activities [64]. Moreover, SysML as part of OMG along with QVT and OCL could be used to define constraints and model transformations, so as to be able to verify a system model using formal validation methods, such as simulation. A large number of modeling tools, (open source, like Eclipse [111], Pa-

pyrus UML [112] and Modelio [113] and enterprise, like Magicdraw [3], Visual Paradigm [114] and Enterprise Architect [115]) support SysML. Moreover, SysML supports many kinds of diagrams like *Block* and *Internal Block Definition* diagrams, to describe complex systems (SoS) and Requirements diagram to handle requirements and their relationships.

Moreover, in a survey about what industry needs from architectural languages [116], UML is by far the most preferred modeling language for architectural description, but the architecture description languages have Insufficient expressiveness for non-functional properties.

Blocks are modular units of system description. Each block defines a collection of features to describe a system or other element of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit [64]. Thus, blocks are suitable to describe systems such as IS consisting of software and hardware blocks. The *Block Definition Diagram* in SysML defines features of blocks and relationships between blocks such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree. The *Internal Block Diagram* in SysML captures the internal structure of a block in terms of properties and connectors between properties [64]. Internal Block Diagrams are convenient to describe internal network architectures, such as LANs.

What's more, SysML makes use of a number of stereotyped dependencies, particularly in the *requirement* diagram and *use case* diagram [117]. Allocations define a basic allocation relationship that can be used to allocate a set of model elements to another, such as allocating behavior to structure or allocating logical to physical components. A requirement is related to other key modeling artefacts via a set of stereotyped dependencies. The *deriveReqt* and *satisfy* dependencies describe the derivation of requirements from other requirements and the satisfaction of requirements by design entities, respectively. The *verify* dependency shows the link from a test case to the requirement or requirements it verifies. In addition, the UML *refine* dependency is used to indicate that an SysML model element is a refinement of a textual requirement, and a *copy* relationship is used to show reuse of a requirement within a different requirement hierarchy. The *rationale* concept can be used to annotate any model element to identify supporting rationale including analysis and trade studies for a derived requirement, a design or some other decision.

Proposed views and viewpoints should be supported by a modeling tool. To follow the standards of MBSE, the formal way is to extend a standard metamodel, as INCOSE promotes the integration and interoperability of methods and tools. MOF profile mechanism can be used to extend UML meta-model, to support any domain specific languages. In our case, SysML, as an extension of UML is chosen for extension, as it effectively describes systems and SoS. Although SysML is the preferred modeling language for system engineering, an extension is necessary in order to describe EIS architecture. The next chapter (chapter 5) explains why SysML needs extension to effectively support the non-functional requirements definition, their derivation and a way to be verified. *Stereotype* mechanism is used for this

purpose and a profile, called EIS profile, has been implemented as an extension of SysML.

The proposed approach consists of the profile definition, which include the extensions of SysML to support NFRs and the definition of a performance evaluation method. Moreover, it includes the use of methods and tools, such as model transformations and simulation execution of a well established simulation environment. These processes are supported by a corresponding extension mechanism of a broad used modeling tool.

MOF Extension Mechanism

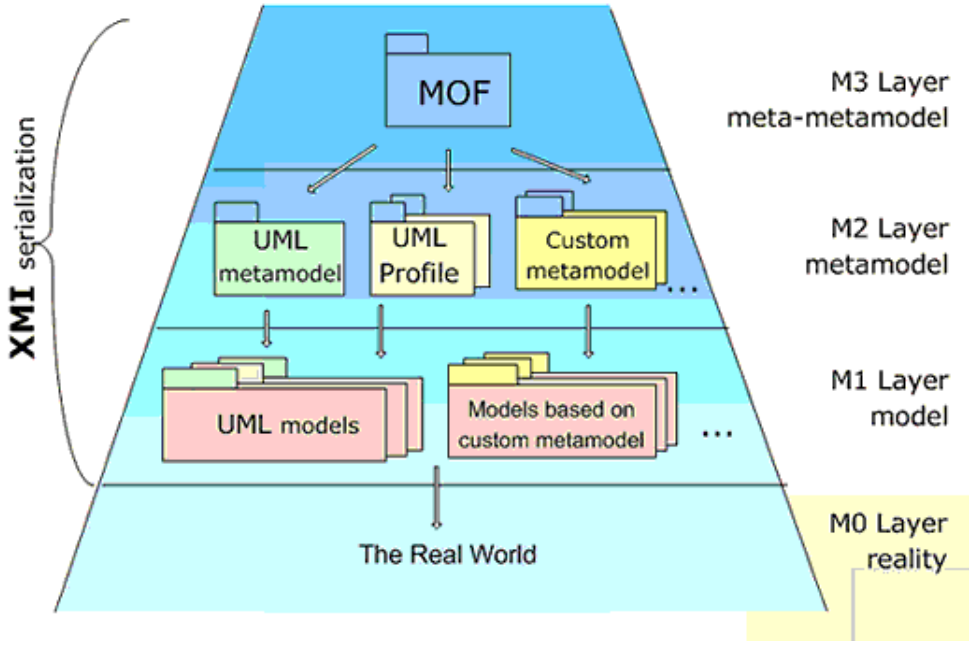


Figure 4.7: MDA four-layer architecture [1]

As stated earlier, to support the proposed approach, extensions should be done to existing modeling languages as SysML. There is a formal way to support extensions. MOF 2.0 [118] states that in order to create models using a specific language, an appropriate metamodel should be defined. The following paragraphs explain the extension mechanism that is provided by MOF.

MOF is designed as a four-layered architecture (see Figure 4.7). It provides a meta-meta model at the top layer, called the *M3* layer. This M3-model is the language used by MOF to build metamodels, called M2-models. The most prominent example of a Layer 2 MOF model is the UML metamodel, the model that describes the UML itself. These M2-models describe elements of the M1-layer, and thus M1-models. These would be, for example, models written in UML. The last layer is the M0-layer or data layer. It is used to describe real-world objects. Another OMG foundation standard XMI, which defines mapping from MOF-defined metamodels to XML documents and schemas. XML enables meta-metamodel, metamodel and model sharing through XMI.

[1] source: http://www.jot.fm/issues/issue_2006_11/article4/

For UML, the metamodeling approach means that ”a metamodel is used to specify the model that comprises UML”. OMG states that the meta-metamodeling layer forms the foundation of the metamodeling hierarchy. The primary responsibility of this layer is to define the language for specifying a metamodel [119]. The layer is often referred to as M3, and MOF is an example of a meta-metamodel. MOF is used as the meta-metamodel not only for UML, but also for other languages, such as CWM. The UML Superstructure metamodel [120] is specified by the UML package on the diagram in Figure 4.8. UML is defined as a model that is based on MOF. Each model element of UML is an instance of exactly one model element in MOF. A model is an instance of a metamodel. UML is a language specification (metamodel) from which users can define their own models.

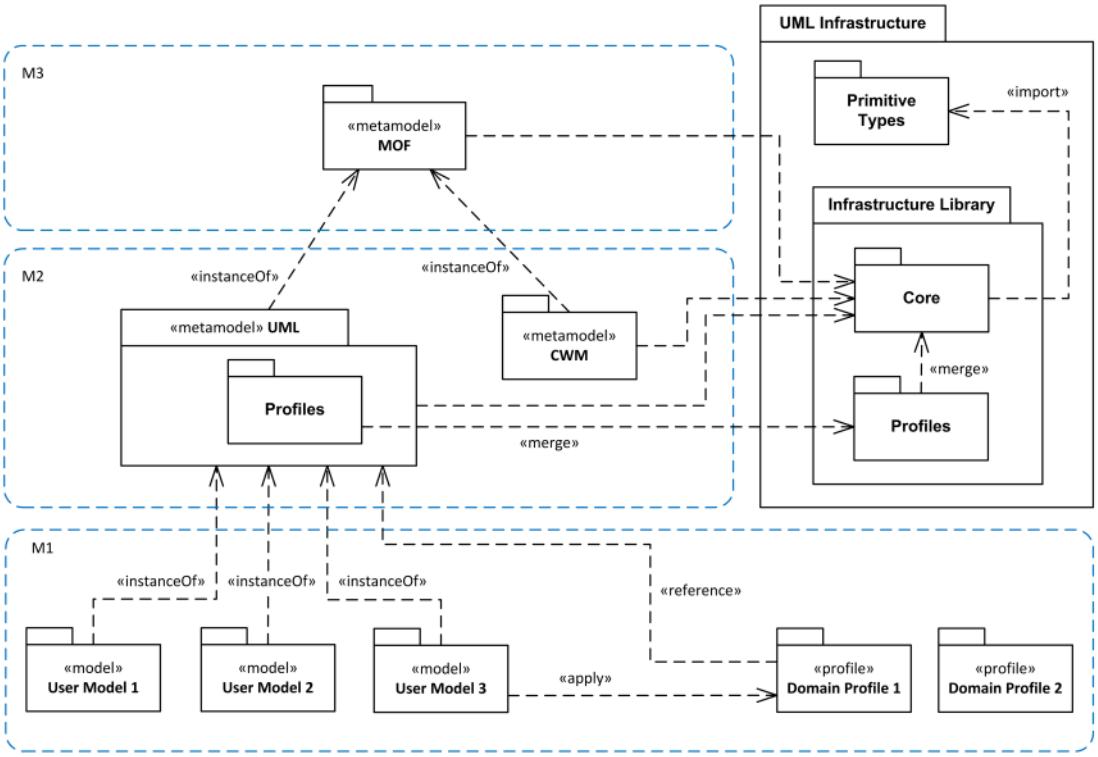


Figure 4.8: Meta Meta Models, UML and Profiles

The *Profiles* package of the *Infrastructure* Library contains mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes, e.g. to adapt the UML metamodel for different platforms (such as JEE or .NET) or domains. As such, it could be considered at the same meta-metalevel as MOF - one level higher than the UML metamodel. The *Profiles* package of the UML Superstructure (from *Auxiliary Constructs*) merges *Profiles* package of the *Infrastructure* library. A *profile* is a restricted form of a metamodel that must always extend some reference metamodel that was created from MOF, such as UML or CWM. So, a *profile* can be defined as a set of *stereotypes* and *tag values* customized for particular domain modeling.

Stereotype is a profile class which defines how an existing metaclass may be extended as part of a profile. It enables the use of a platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass. When a stereotype is

applied to a model element, the values of its properties may be referred to as *tagged values*.

EIS Profile Overview

To define the structure of the proposed profile, we should tabulate the basic tasks identified during EIS architecture design (characterized as viewpoints in section 4.3.1), which are:

i. *Functionality* definition, consisting software architecture description (e.g. a system-oriented view of applications). In practice, Functionality definition consists of the description of functional requirements (e.g. application and data architecture, user behavior and application requirements).

ii. *Topology* definition, consisting of the description of system access points. It facilitates user, application and data allocation to system access points.

iii. *Network Infrastructure* definition, consisting of the description of platform- independent distributed infrastructure (e.g. network architecture and hardware configuration) and the association of software components to network nodes (resource allocation).

iv. *NFR* definition, consisting of the description of non functional requirements, focusing on system performance and availability requirements essential for EIS architecture design.

v. *EIS architecture evaluation*, consisting of these model elements that participate in the performance evaluation process along with the requirements that should be verified.

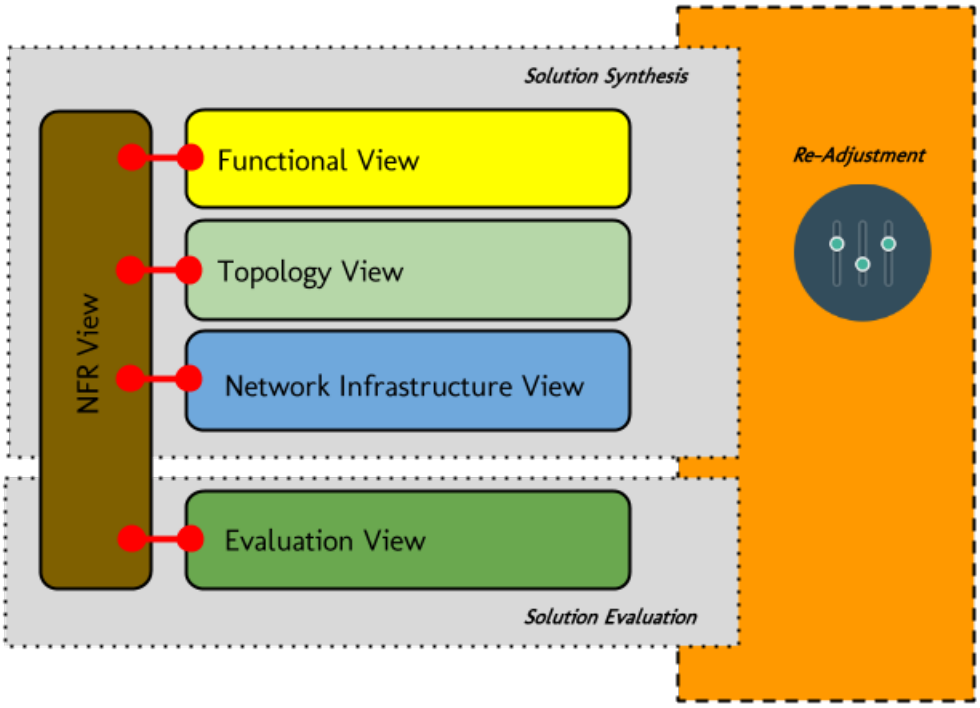


Figure 4.9: EIS Architecture Views and Corresponding Design Tasks

According to the basic processes that are identified during the design activity (presented in Chapter 1.3, Figure 1.2), solution synthesis encompasses *Functionality*, *Topology* and *Network Infrastructure* definitions. *Functionality* definition focuses on software architecture design, *Topology* definition on software allocation process and *Network Infrastructure* definition on hardware architecture design. For each of these concerns, a corresponding view is defined to explore non functional requirements and related design decisions. The system designer, concerned with software architecture design and software allocation, is served by *Functional* and *Topology* views to contribute in the construction of EIS architecture. In a similar fashion, the designer is contributing to hardware configuration using Network Infrastructure View. A complementary view, called *Evaluation* view is proposed to serve system evaluation activity and manage evaluation results and requirements verification. This view incorporates entities from other views and stores all the required attributes for each model element, in order to facilitate the requirements verification process. Specifically, evaluation view facilitates:

i. the definition of the conditions under which the system will be evaluated;
ii. the incorporation of the evaluation results;
iii. the requirement verification, informing the system designers for inconsistencies.

EIS architecture views and corresponding design tasks are presented in Figure 4.9.

It is evident that all aforementioned tasks are interrelated, since none of them can be completed independently, while in most cases tasks are performed in parallel, and often repeatedly by the system architect in order to reach an EIS architecture satisfying both functional (identified during *Functionality* definition and partly during *Topology* and *Network Infrastructure* definition) and NFRs (identified during NFR definition). NFR definition is performed in parallel with Functionality, Topology and Network Infrastructure definition. Developing requirements and architectural artifacts in parallel has already been addressed in section 4.2.2.

After the definition of EIS architecture, an evaluation phase follows, most commonly using simulation, if it is about performance issues. Solution evaluation will determine whether the proposed solution is satisfying all functional and non functional requirements, or the system designer should improve the proposed architecture or readjust requirements by repeating definition tasks. Adopting a model-based approach for EIS architecture design should provide the system architect with a common system model to support all design tasks and enable him/her to perform each design task in an independent fashion taking into account the restrictions imposed by other tasks.

As depicted in Figure 4.10, the aforementioned views are associated with relations such as *satisfy*, *verify*, *allocate* and *evaluate*. These relations are supported by SysML [64]. A *satisfy* relationship is a dependency between a requirement and a model element that fulfills the requirement [64]. *Satisfy* relates system elements in *Functional*, *Topology* and *Network Infrastructure* views with their corresponding requirements. A *verify* relationship is a dependency between a requirement and a test case or other model element that can determine whether a system fulfills the requirement [64]. In EIS profile, *Verify* relates requirements that are verified by elements from evaluation view. SysML also includes an *allocation* relationship to rep-

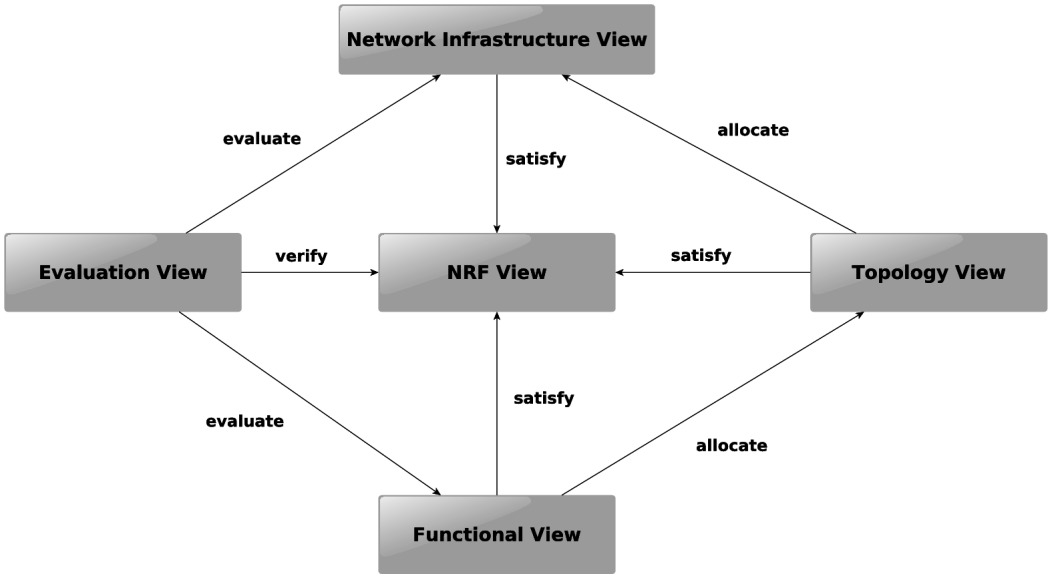


Figure 4.10: EIS architectural model

resent various types of allocation, including allocation of functions to components, logical to physical components, and software to hardware. In our case, *allocate* relates entities from *Functional* or *Topology* views that are allocated to entities from *Network Infrastructure* view, supporting the software to hardware allocation and users allocation (system access points). Finally, *evaluate* relates entities from evaluation view that are evaluating entities from the other views (design views). *Evaluate* relation defined as an extension of SysML relations.

The next two chapters present the defined views, which are categorized as design views (chapter 5), that are functional, topology, network infrastructure and NFR view and evaluation view (chapter 6).

4.4 Summary

This chapter presented an approach based on the concepts of MBSE as defined by INCOSE to explore EIS architecture design. The proposed approach is based on Zachman framework, showing the concepts of views and viewpoints we adopted and the stakeholders identified throughout the design process. To this end, a SysML profile was defined. Next chapter extensively presents two parts of the proposed approach: the design phase and related views and the handling of NFRs.

Chapter 5

Designing EIS Architecture

Contents

5.1 Outline

This and the next chapter present EIS profile views, grouped in three categories:

- *Design* Views. Within these views, system designer defines the software and hardware architectures of the system.
- *NFR* View. Here, non functional requirements are defined for model elements of the design views.
- *Evaluation* View. Since design and NFR views are complete and consistent, evaluation view collects these entities from design views that are participating in the evaluation process, initiates the evaluation process and maintains the results to validate system against NFRs.

Design views provide the appropriate diagrams to define the applications software architecture, to set system access points and to define network architecture.

NFR view, in practice is not an autonomous view. Requirements are defined in design views, each of them satisfying design entities, and are interrelated in many ways:

- requirements are *derived* from other requirements
- requirements *satisfy* design entities
- requirements are *verified* from evaluation entities

Evaluation view encompasses the appropriate design entities along with their requirements in order to clarify which of them could be verified so as to inform the designer about the non-verified ones.

5.2 Design Views

As *design* views we could consider the *Functional*, *Topology* and *Network Infrastructure* views, where system designer defines software and hardware architecture and makes the appropriate allocations (users and software allocations).

Specific extensions, as supported by OMG, provided the desired functionality. A view in EIS profile is depicted as a discrete diagram. The stereotype mechanism provided by UML and SysML is employed to customize SysML functionality to depict EIS Architecture views. *Functional*, *Topology* and *Network Infrastructure* views are described using hierarchical *block-definition diagrams*. SysML *blocks* can be used throughout all phases of system specification and design, and can be applied to many different kinds of systems. These include modeling either the logical or physical decomposition of a system, and the specification of software, hardware, or human elements.

An overview of the synthesis model (i.e., the model elements participating in design views) is presented in Figure 5.1. Their connections with NFRs and the *allocations* between them are presented with dashed lines. The *allocate* relation between design views indicate that entities defined in *Functional* view and more specifically software *modules*, *data entities* and users modeled as *roles* are allocated in system access points, called *sites*, defined in the *Topology* view. The allocation of modules, roles and data entities to sites corresponds to software architecture design. The allocation relation between *Topology* and *Network Infrastructure* views indicates that each *site* defined in *Topology* view is served by a *network* defined in *Network Infrastructure* view. When a *site* is allocated to a network, *Functional* view entities allocated to this site must be specifically allocated to hardware nodes belonging to this network. Figure 5.1 also illustrates the requirements which are referenced by design views model elements. Section 5.3 analyzes the role and the relationships between requirements and model elements belonging to other views.

For each of the aforementioned views, a corresponding table summarizes the elements participating in each view. So the defined stereotype, the SysML/UML extended element, the

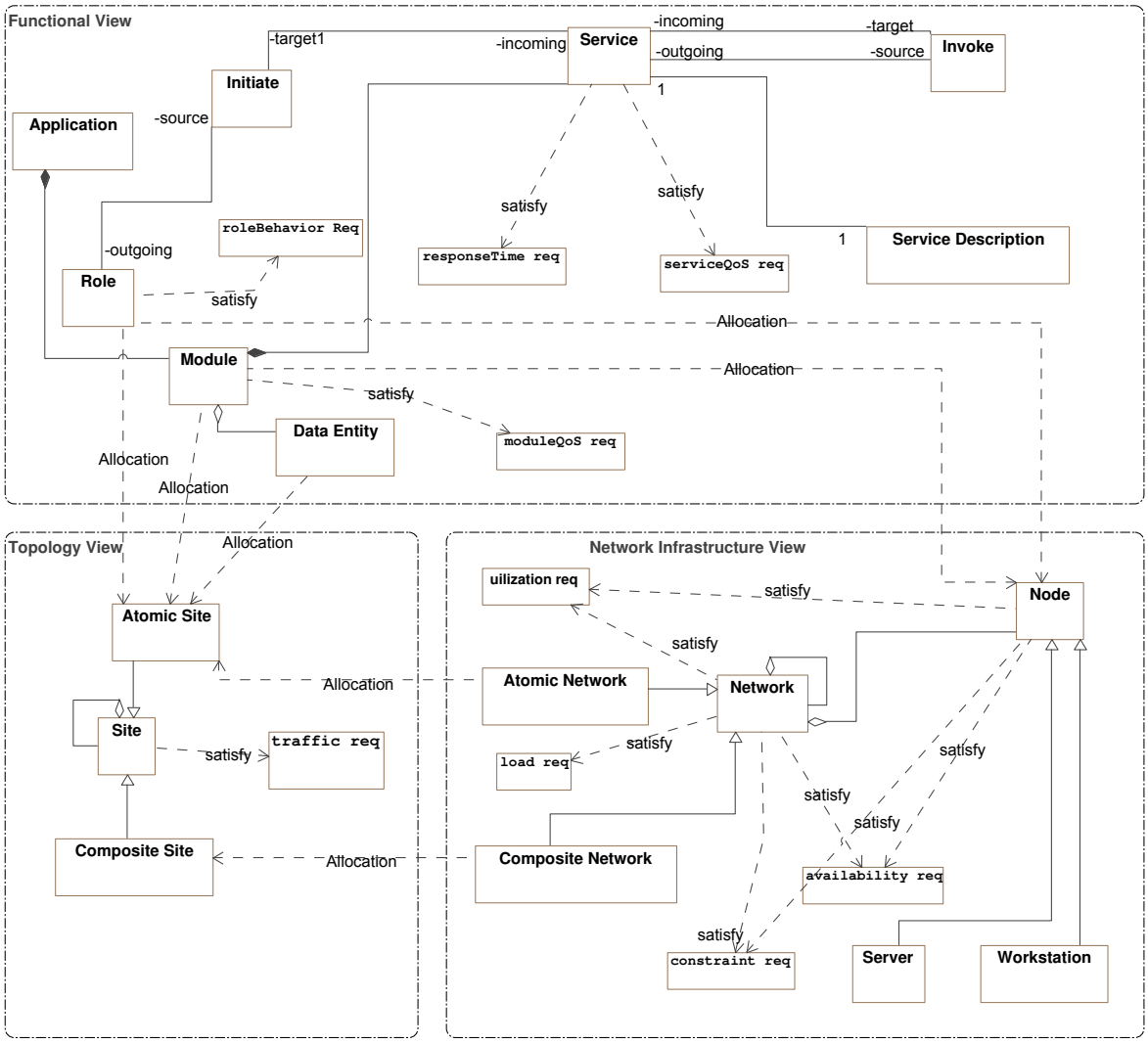


Figure 5.1: EIS synthesis model

tagged values, the *constraints* and the type of the entity are presented at the columns of the table. Each model element, belonging to a view, can have one of the following types:

- *internal*, focusing on the specific view
- *external*, facilitating the integration with entities belonging to other views

5.2.1 Functional View

Functional view depicts functional requirements related to software components and related data, as well as EIS users. It also includes design decisions related to software architecture. Entities that are participating in *Functional* view are briefly described here. *Roles* are used to depict the behavior of different user groups while *modules* (client & server) are application tiers (supporting multi-layered applications) that consist of *services*. Each role *initiates* services that belong to client modules, and each service may *invoke* other services that belong to other modules, depending on the complexity of the application. *Data Entities* are used to represent portions of stored data. Main entities of *Functional* view, along with their type (external or internal, see 4.3.1) and the SysML base class they extend, are presented in Table

5.1. Entities participating in *Functional* view are related to entities participating in all other diagrams to implement the relations depicted in Figure 4.10. These relations are discussed in the following paragraphs as the rest of the views are presented.

Table 5.1 presents the complete definition of the stereotypes belonging to *Functional* view, where the properties (tagged values) of each stereotype and the constraints are depicted.

Table 5.1: Functional View Entities

Stereotype	Base Class	Properties	Constraints	Type
Functional View	Block Def. Diag.		Only FV stereotypes participate in it and all defined constraints must be validated	
Role	Actor	StartTime EndTime NumOfOccurs	Values must be defined for all attributes Initiates at least one service Total of related service initiations percentage must be 100% Satisfies a Behavior Requirement Allocated to an Atomic-Site	External
Service	Block		Satisfies a Response-Time Requirement Satisfies a Service-QoS Requirement Belongs to one Module	External
Server-Module	Block		Includes at least one service Satisfies a Module-QoS Requirement Allocated to an Atomic-Site	External
Client-Module	Block		Includes at least one service Satisfies a Module-QoS Requirement Allocated to an Atomic-Site	External
Data Entity	Block	Size Type Repl. Policy	Invoked by a service Allocated to an Atomic-Site	External
Initiate	Dependency	Percentage	Defined between a Role and a service, belonging in a Client-Module	Internal
Invoke	Dependency		Defined between Services	Internal
Module-Invoke	Dependency		Defined between Modules and created automatically Satisfies a Module-QoS Requirement	External

Let us examine this view from the designer's perspective. Using *Functional* view, system designer has to define the applications and the users of the system. To describe an application, all tiers, starting from the user invocation of a specific operation (through the appropriate interface) to the remote services or database operations should be described. Applications, either web-based or not, are composed of many software components that are communicating and exchanging data. In *Functional* view the architecture of the distributed software components is described.

A user is stated as a role, where specific attributes define his behavior: StartTime, EndTime to declare his daily working hours and NumOfOccurs to declare how many users of that role are existing in the IS. Note that roles are aggregations of users having the same working behavior and dealing with the same applications. The next step is to define the user interfaces (called *client-modules*, stereotypes of *Block* SysML entity), through which users are interacting with applications. Applications are complex, constituting of many tiers (a set of client modules and

server-modules). Each module that is not invoked by a role, is considered as server-module and is a stereotype of SysML *Block* entity.

Roles *initiate* operations of applications (called *services*, which are also stereotypes of *Block*) belonging to *client-modules*. We can imagine services and modules as operations of web-services. The communication between services belonging to different tiers of applications is defined with *invoke* relations, which are stereotypes of UML *Dependency* relationship.

A user can interact with different user interfaces and call a set of operations in the duration of his working time. So, a *percentage* attribute describes the percentage of time a user is interacting with each specific operation. For that reason a constraint defined to this model element is that the sum of percentage attributes of *initiate* relations starting from a specific role, should be 100%. UML defines the constraint as "a restriction or a condition that should be applied". *Data entities* are modeling application's files that store data.

Table 5.1 presents the specific constraints that entities of *Functional* view should satisfy. For example, a role satisfies a *behavior* requirement. This means that a role could behave differently under certain circumstances, e.g. on a heavy load day. Section 5.3 presents the defined requirements and how they relate to other elements.

A conceptual representation of *Functional* view entities is presented in Figure 5.2. Please note that a *Module-Invoke* relation is defined between two *Modules*. This relation is auto-created and is associated with a *module-QoS* requirement, that *holds* the amount of *processing*, *storage* and *traffic* (networking) requirements of the communicating services of these Modules.

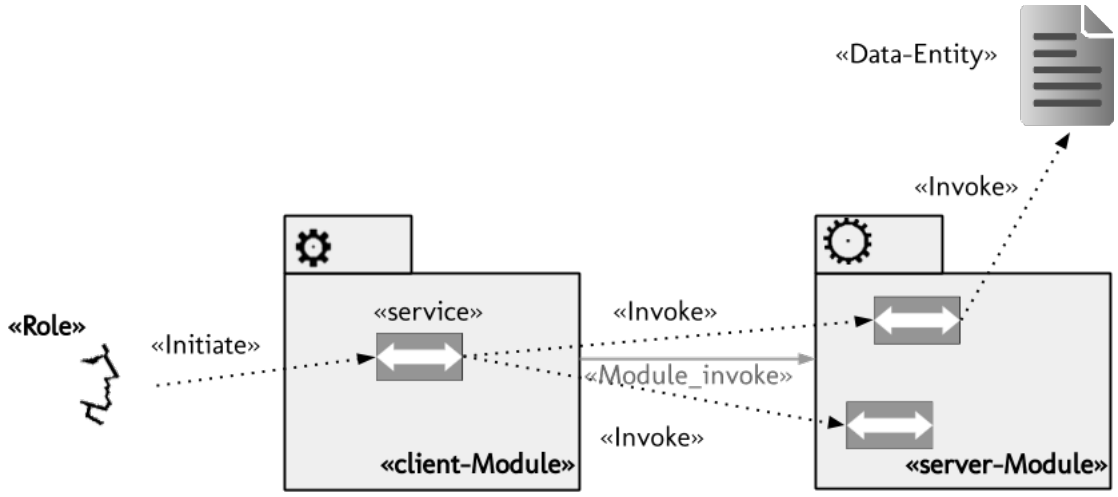


Figure 5.2: Functional view entities

5.2.2 Topology View

Topology view act like a bridge between *Functional* view and *Network Infrastructure* view, facilitating the hierarchical allocation of software entities (such as software modules) and users to hardware elements (such as networks and nodes). It acts as an intermediate step,

that provide information to the designer in order to facilitate him to make the appropriate allocations. Moreover, in *Topology* view, software component *replicas* can be defined, to support distributed architectures. This means that many instances of roles, software components and nodes are supported in EIS profile. The allocation is enhanced with the definition of derived requirements that capture the load that software components and user interactions produce to hardware components.

Topology view facilitates the description of system access points in terms of hierarchically related locations, called *sites*. *Sites* may be *atomic* or *composite* (meaning that are composed of other sites). The *Site* entity is an extension of SysML *Block* entity. *Topology* view is a *Block Definition* diagram that comprises the aforementioned entities. *Topology* view entities are presented in Table 5.2. Entities defined in other views (e.g. *Functional* view) may also participate in this diagram, in order to describe *Topology* view interrelation with other diagrams, as defined in Figure 4.10. *Software Allocation* is used to describe the allocation of software modules (client or server) to *atomic sites*, while *Usage Allocation* refers to roles that are allocated to atomic sites. Sites satisfy *traffic* requirements, indicating the amount of information exchange between the modules allocated to them. A *traffic* requirement is described in terms of traffic coming in, going out and exchanged within each site. Traffic requirements are entities defined in NFR view.

Constraints were used in two ways:

- to constraint SysML functionality, for example only modules may be software-allocated to sites or atomic sites are allocated only to atomic networks and composite sites or networks have to own at least one atomic element and
- to compute values of derived entity attributes, for example, traffic requirements attributes of a specific site are automatically computed from *module-Qos* requirements attributes of modules allocated to this site.

Table 5.2 presents the entities of the *Topology* view, and for each entity the attributes and the constraints are depicted.

Table 5.2: Topology View Entities

Stereotype	Base Class	Properties	Constraints	Type
Topology View	Block Def. Diag.		TV stereotypes and Roles, Client-Modules and Server-Modules from FV participate in it and all defined constraints must be validated	
Site	Block	Range Instances		External
Atomic-Site	Block			External
Composite-Site	Block		Comprised of other Sites	External
Software-Allocation	Allocation	Instances	Defined between Modules/Data Entities and Atomic Sites	Internal
Usage-Allocation	Allocation	Instances	Defined between Roles and Atomic Sites	Internal
Client-Module-Replica	Block		Autocreated from a Client-Module A Client-Module can have one or many Client-Module-Replicas Client-Modules from Topology view are migrating to Topology view, to make replicas of them When a role is allocated to a site, for every service that this role initiates, for the corresponding client module a replica is created and is allocated to the same atomic-site	Internal
Server-Module-Replica	Block		Autocreated from a Server-Module A Server-Module can have one or many Server-Module-Replicas Server-Modules from Topology view are migrating to Topology view, to make replicas of them Allocated with Software-Allocation to Atomic-Sites	Internal
Replica-of	Realization		Defined between Modules and their Replicas	Internal

Two kinds of allocation are defined in *Topology* view: *Software* and *Usage*. The first is defined between software component instances (module-replicas) and atomic-sites and the latter to users allocation to atomic sites, defining the system access points to users. *Roles*, *client-modules* and *server-modules* are derived from *Functional* view and are appeared to *Topology* view, so as to give the ability to the designer to make instances (replicas) of them.

A major constraint defined in *Topology* view is the following: when a role is allocated to an atomic-site, for each client-module that the role initiates services from it, a replica is auto-created and auto-allocated to this atomic-site. Figure 5.3 presents software-allocation of client-module-replicas in gray color, meaning that these allocations are automatically created, when a role is allocated to a site. *Client* and *server modules* are presented in blue, to identify that these entities are coming from *Functional* view. Note that atomic sites may be hierarchically contained in composite sites with the containment relationship of UML.

Section 5.3 presents the requirements defined in *Topology* view. Furthermore, the derivation of the complex requirements is provided there. For example, in *Functional* view, each service is related with a *service-QoS* requirement, indicating the required *processing*, *storage* and *networking* resources that this service needs in order to be executed (i.e., three types of *service-QoS* are defined: *proc-service-QoS*, *stor-service-QoS* and *traffic-service-QoS*). The *module* that contains these *services*, gathers all requirements of services in a requirement called

module-QoS requirement (also three types of *module-QoS* are defined: *proc-module-QoS*, *stor-module-QoS* and *traffic-module-QoS*). In order to estimate the gathered processing and storage requirements for the *module-QoS*, a simple aggregation is enough. For the traffic type of *module-QoS*, this should be defined for each *module-invoke* relation between modules. This derivation is produced in *Topology* view, where module-replicas are defined. The derivation of *module-QoS* requirements is presented in section 5.3.4.

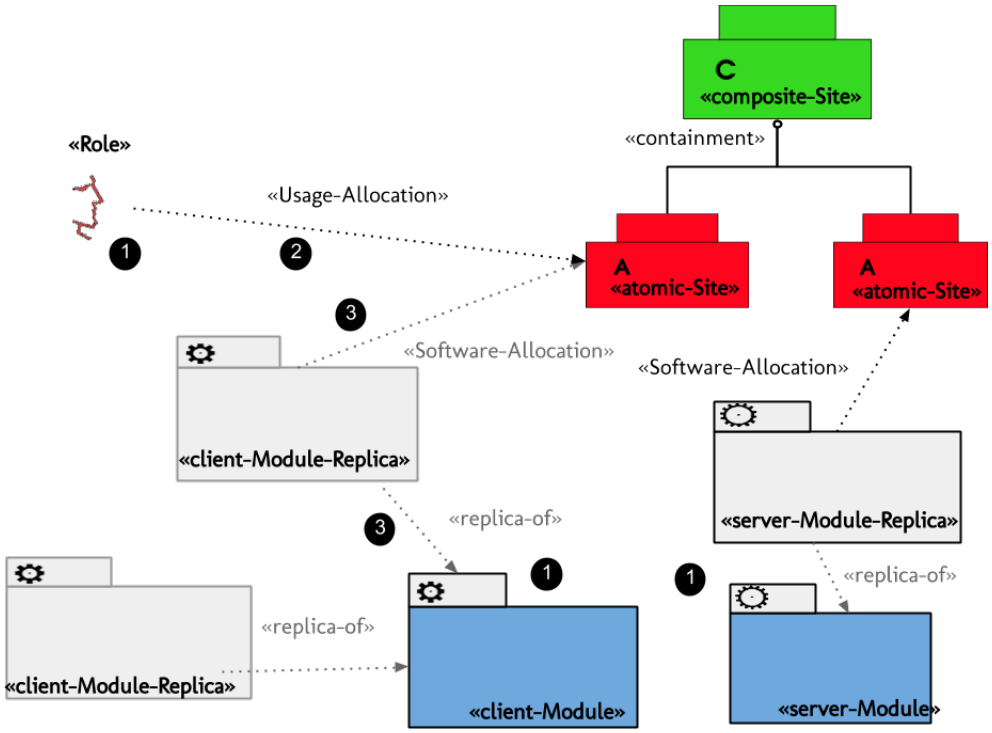


Figure 5.3: Topology view entities

From the system designer's perspective, a usual scenario would be the following. Starting with sites definition, a reasonable hierarchy could be done taking into account the geographical distribution (e.g regions or buildings) of the information system. *Sites* could be considered as *hosts* of users and software components.

As stated, *roles*, *client-modules* and *server-modules* defined in *Functional* view participate also in *Topology* view. Validation rules are applied to system model to ensure this. In a next step, *roles* are allocated to *atomic-sites*. Each *role* in *Functional* view initiates *services*. These services belong to *client-modules*. For that reason, according to the aforementioned constrains, when a *role* is allocated to an *atomic-site* and for each *client-module* that this role *initiates*, a *client-module-replica* is also automatically allocated to the same *atomic-site*. This process is automated to help the designer to define the allocation policy.

Afterwards, the designer decides how many replicas should be defined for each of the *server-modules* and allocates them to *atomic-sites*. Another validation rule checks all *modules* in order to ensure that are allocated to an *atomic-site*. If this is not valid, the system designer is notified about the non-allocated modules. The final step is to estimate the traffic that software allocated modules produce in a site. This helps the designer to define the allocation

policy, based on the load imposed to sites.

A conceptual representation of topology view entities is presented in Figure 5.3. Firstly, *roles*, *client-modules* and *server-modules* are derived from *Functional* view (noted as ❶). Then the *role* has been allocated to an *atomic-site* (noted as ❷), and as a result of the automation, the appropriate *client-module-replicas* are created and allocated to the same atomic-site (noted as ❸).

5.2.3 Network Infrastructure View

Network Infrastructure view refers to the aggregate network, described through a hierarchical structure comprising simple and composite *networks*. It is represented using a hierarchy of *Block Definition* diagrams. Hardware components and configurations are also defined using this view (*servers, workstations and network devices*). Networks could inter-connected with *PTP-connections* (i.e., point-to-point connections, the simplest topology with a permanent link between two endpoints) or could belong to other networks, which is defined with the usage of UML *containment* relationship. Consider a LAN where smaller networks could be defined (e.g. Virtual Local Area Networks (VLANs)) with different configurations for each of them.

Sites are allocated to *networks* using *Structural Allocation* relation. Each *atomic network* is a custom diagram (based also on *Block Definition* diagram), called *atomic network diagram*, which encompasses all hardware elements that belong to that network. *Network Infrastructure* view entities are presented in Table 5.3. Most of them are characterized as *external* entities, since they should be further refined during network implementation by the system constructor. Existing network infrastructure is depicted using constraint requirements defined in NFR view and associated to appropriate network components in *Network Infrastructure* view. Elements of *Functional*, *Topology* and *NFR* views may also participate in *Network Infrastructure* view to represent inter-view relations.

To depict the usage of *Network Infrastructure* view, consider a network architect designing an information system's network using this view. Network architecture is defined taken into account:

i. the system access points, called *sites*, defined in the *Topology* view,
ii. the traffic performance indications for the information exchange within and between sites, and
iii. existing network infrastructure restrictions.

Network architecture is defined in a hierarchical fashion, constituting of *atomic networks*, which depict local networks that connecting hardware elements, such as workstations and servers, where eventually software components are allocated. Atomic networks are interconnected through networks either private or public.

Table 5.3: Network Infrastructure View Entities

Stereotype	Base Class	Properties	Constraints	Type
Network	Block	ProtocolStack Throughput Type NumOfMaxNode		External
Atomic-Network	Block			External
Composite-Network	Block		Composite-Network must contain at least one Atomic or Composite Network	External
Atomic-Network-Diagram	Block Def. Diag.		Atomic Diagram must be associated to an Atomic Network	External
Server	System	Memory OperatingSystem StorageUnit ProcessingUnit		External
Workstation	System	Memory OperatingSystem StorageUnit ProcessingUnit		External
Processing-Unit	System	Cores ProcPower		External
Storage-Unit	System	Capacity StorageSpeed		External
Connection	Association	Usptream Downstream		External
PTP-Connection	Association	ProtocolStack Speed		External
Structural-Allocation	Abstraction	Instances	Atomic and Composite Sites are allocated to Atomic and Composite Networks	Internal
Software-Allocation	Abstraction	Instances	Client and Server Modules are Allocated to Servers and Workstations	Internal
Usage-Allocation	Abstraction	Instances	Roles are Allocated to Workstations and Servers	Internal

Three kinds of allocations are defined in this view:

- *Structural allocation* defines the allocation between sites and networks. Allowed allocations are between the same type of elements: i.e. atomic sites can be allocated to atomic networks.
- *Usage allocation* defines the allocation of users to nodes.
- *Software allocation* defines the allocation of software components to nodes, responsible for their execution.

Servers and *workstations* are comprised of three units: *processing*, *storage* and *network* (in accordance with the three types of requirements in *service-QoS* and *module-QoS* requirements). *Processing* unit has tagged-values to define the processingpower and the cores of the CPU. *Storage* unit has tagged-values that correspond to diskcapacity and speed (defined in rpm). *Network* entity has the following entities: type, which corresponds to network *type* (ethernet, wifi, bluetooth), throughput (depicted in Mbps) , protocolStack which corresponds to the

supported protocols by this network (e.g. TCP, UDP) and numOfMaxNodes, a constraint to define the maximum number of connected devices to this network.

The allocation relation between *Topology* and *Network Infrastructure* views indicates that each *site* defined in the *Topology* view is served by a *network* defined in *Network Infrastructure* view. When a site is allocated to a network, *Functional* view entities allocated to this site must be specifically allocated to network nodes belonging to this network. This task is assigned to system designer/network architect. A validation rule ensures that there are no elements (*roles* and *module-replicas*) non-allocated to networks.

Network Infrastructure view is multi-level. At the first level *atomic* and *composite networks* along with their hierarchy and their connections are defined. The next step is to gather the *atomic* and *composite sites* from the *Topology* view. Each of them has to be *structural-allocated* to a *network*. A constraint applied here is that an *atomic-site* has to be allocated to *atomic network* and a *composite site* to a *composite network*. Of course, many atomic-sites can be allocated to one atomic network. Figure 5.4 presents a sample of a *Network Infrastructure* diagram: *Sites* are allocated to *atomic networks*, and for each *atomic-network*, a corresponding block definition diagram is defined.

In a second level of allocations, for each *atomic-network* a corresponding *atomic-network diagram* is created. To accommodate the designer, in an *atomic-network diagram*, all allocated elements (*roles* and *module-replicas*) to *sites* that are allocated to this *atomic-network*, are automatically created. The remaining task for the designer is to define the *servers* and *workstations* that should *accommodate* them. Afterwards, in the *atomic-network diagram*, the system designer has to allocate *roles* and *module-replicas* to these *workstations* and *servers*. Figure 5.5 presents a simple *atomic-network diagram*, where a *role* and *client-module-replica* are allocated to a *workstation* and a *server-module-replica* is allocated to a *server*. A constraint that is applied here is that since a *role* is allocated to a *node*, the corresponding invoked *client-module-replicas* should be also allocated to the same node. To facilitate the designer, the allocation of *client-module-replicas* is automated. The reader will notice that in an *atomic-network diagram* the connections between the nodes are not presented. We could take into account the following considerations:

- hardware elements inside an atomic-network diagram, are connected through a LAN connection and the properties of the connection are defined in the atomic-network element
- the amount of information that has to be exchanged between nodes is estimated with the help of the *module-QoS* requirements of the module-invoke relations that has been defined in *Topology* view. There are three types of traffic: in, out and inout. Refer to 5.3.4 for the process of the estimation of the derived requirements.

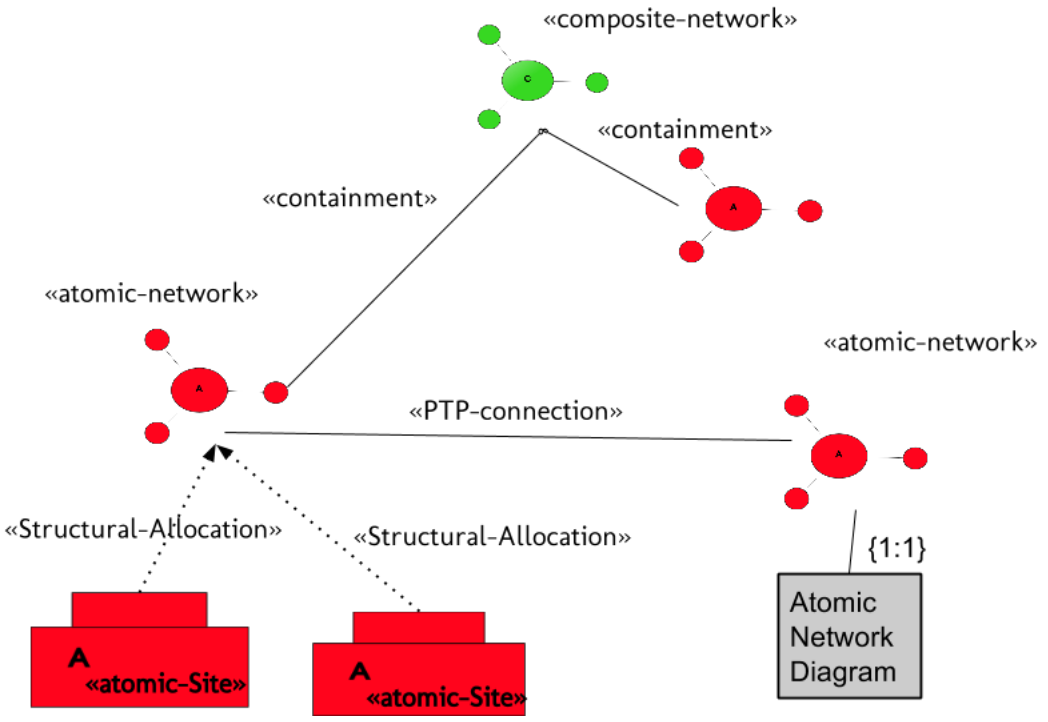


Figure 5.4: Network Infrastructure view entities

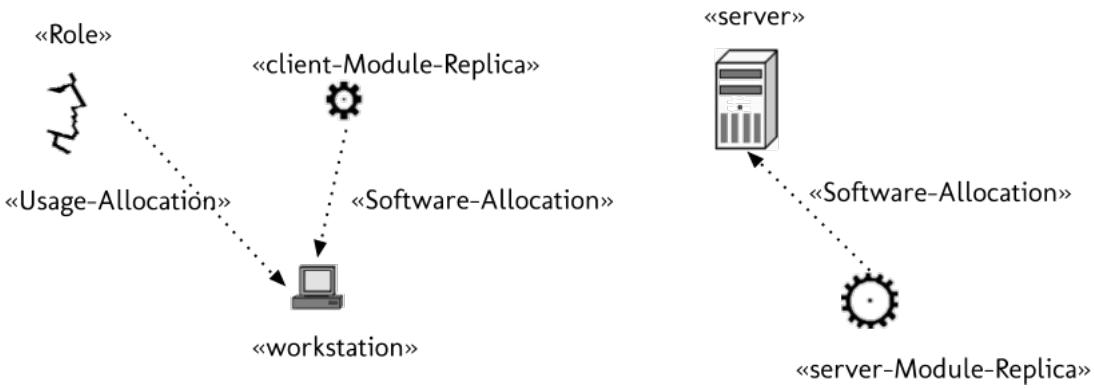


Figure 5.5: Network Infrastructure view: atomic network entities

5.3 Non-Functional Requirements View

NFR view consists of all NFRs that should be satisfied by entities belonging in the three aforementioned -design- views. These requirements are progressively defined during model-based EIS architecture design. Performance requirements are emphasized, since they are essential in EIS architecture design. The utilization of NFR view is not to present all requirements from design views, but relates to the distribution of the requirements to other views. Of course, if a designer wants to see all defined requirements, EIS profile gives him the opportunity to gather all requirements in a single NFR diagram. NFR view bridges the gap between design views and *Evaluation* view, since the NFRs are the mean through which the evaluation can be performed. Thanks to requirements verification, a system model described in design views can be evaluated against the defined requirements.

A classification of NFRs is necessary to help us discover the requirements interrelations.

Two kinds of NFRs are defined: *simple* and *derived*. Simple requirements have attributes that system designer is responsible to provide values. Derived requirements have attributes that their values are depending on values of other requirements. This derivation can be described either with a mathematical expression or can be estimated by a *derivation formula* such as an algorithm. In the latter case an implementation of this algorithm in a programming language is necessary to be incorporated in the modeling tool that the designer uses.

There are three main perspectives (Figure 5.6 presents them conceptually) to help us categorize NFRs, as far as the scope of this thesis defines:

i. *Behavior description*. Requirements belonging at this category are used in order to describe specific user behavior, e.g. user behavior variations under different circumstances. They are usually used as input parameters to evaluation process.

ii. *Performance description*. These requirements are used to dictate specific performance that should be guaranteed by the system components, e.g. response time requirements.

iii. *Load indications*. These kind of requirements are used as specific indicators about the aggregated required resources from the hardware components that the software components impose to them. They help system designer to make allocation policies.

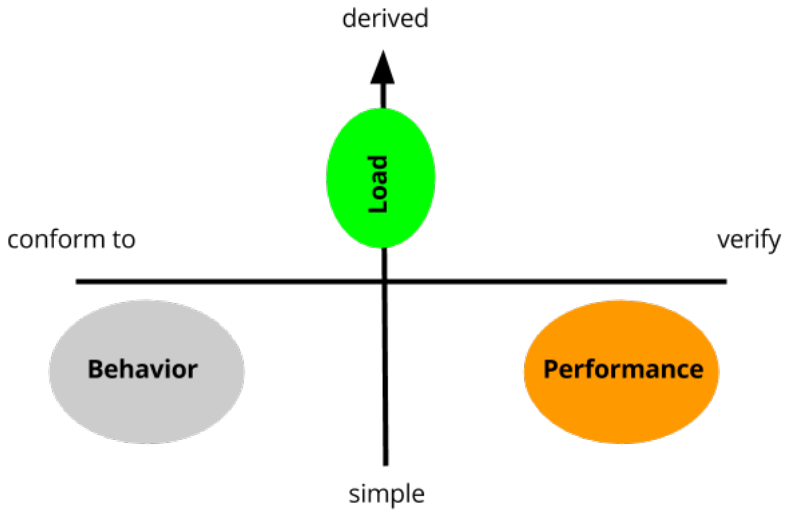


Figure 5.6: Requirements categorization perspectives

5.3.1 Non-functional requirements classification

Requirements could have qualitative and/or quantitative characteristics. Handling them from the performance perspective, any qualitative characteristic should be translated to quantitative, so as to be validated against any formal verification method.

NFR view comprises NFRs relevant to EIS architecture design. They are progressively defined during model-based EIS architecture design tasks. Three main categories are supported:

performance, *physical* and *specific quality* [46]. Performance requirements are emphasized, since they are substantial in EISs architecture design.

Performance requirements are further decomposed to *behavior*, *load* and *utilization* [8]. *Utilization* requirements are associated with *Network Infrastructure* view and regard the proportion of network infrastructure resources used by applications during normal operation or extreme conditions.

Behavior requirements deal with service behavior and are time-related (e.g. response times). They affect *Functional* view. Two of them are defined, namely *responseTime*, indicating the time interval within which a service should complete its execution, and *Behavior*, indicating activation patterns for roles defined within Functional view.

Load requirements concern the load imposed to other EIS resources by EIS components allocated to them. Load requirements are defined in all views. Most of them are derived requirements, calculated using properties of other load requirements. Four different load requirements are defined, namely *service-QoS*, *module-QoS*, *traffic-Load* and *load*.

Regarding *physical* requirements, indicating constraints imposed on design decisions by existing hardware resources, we focus on those concerning *capacity*. Capacity, indicating limitations of the hardware and their impact to the system, is related to *Network Infrastructure* view.

Regarding specific quality requirements, we consider only *availability* requirements. They are associated with *Network Infrastructure* view, where availability deals with hardware aspects. Availability requirements may also be defined for software components within *Functional* View.

NFRs and the way they are interrelated to each other as well as to other entities belonging in *Functional*, *Topology* and *Network Infrastructure* views are depicted in Figure 5.7. In the following, NFRs are analytically presented grouped by EIS architecture view they are satisfied by.

Functional Requirements: Behavior requirement describes alternate user behavior, e.g., when the user is active, or with which probability and how frequent a user initiates services. A *role* initiates *services*, while each service satisfies a *responseTime* requirement. The response time defined here is the accepted time while the user waits for or is informed for the execution of the operation that he is interacting with. The service requires EIS resources for its effective execution, expressed in terms of QoS it should receive from the underlying infrastructure. The *service-QoS* requirement indicates the amount of processed, stored or transferred information a service requires during its execution. Consequently, the *service-QoS* properties are average and maximum estimations of *traffic*, *processing* and *storage* QoS needed for the service execution. The QoS for each service is defined by the system architect, taking into account that it should satisfy corresponding *responseTime* requirement. *Module-QoS* requirement describes the QoS needed for the module execution. It bears the same properties as *service-QoS* and is apparently derived by the *service-QoS* requirements belonging in the same module. Moreover, *module-QoS* requirement properties are calculated as the

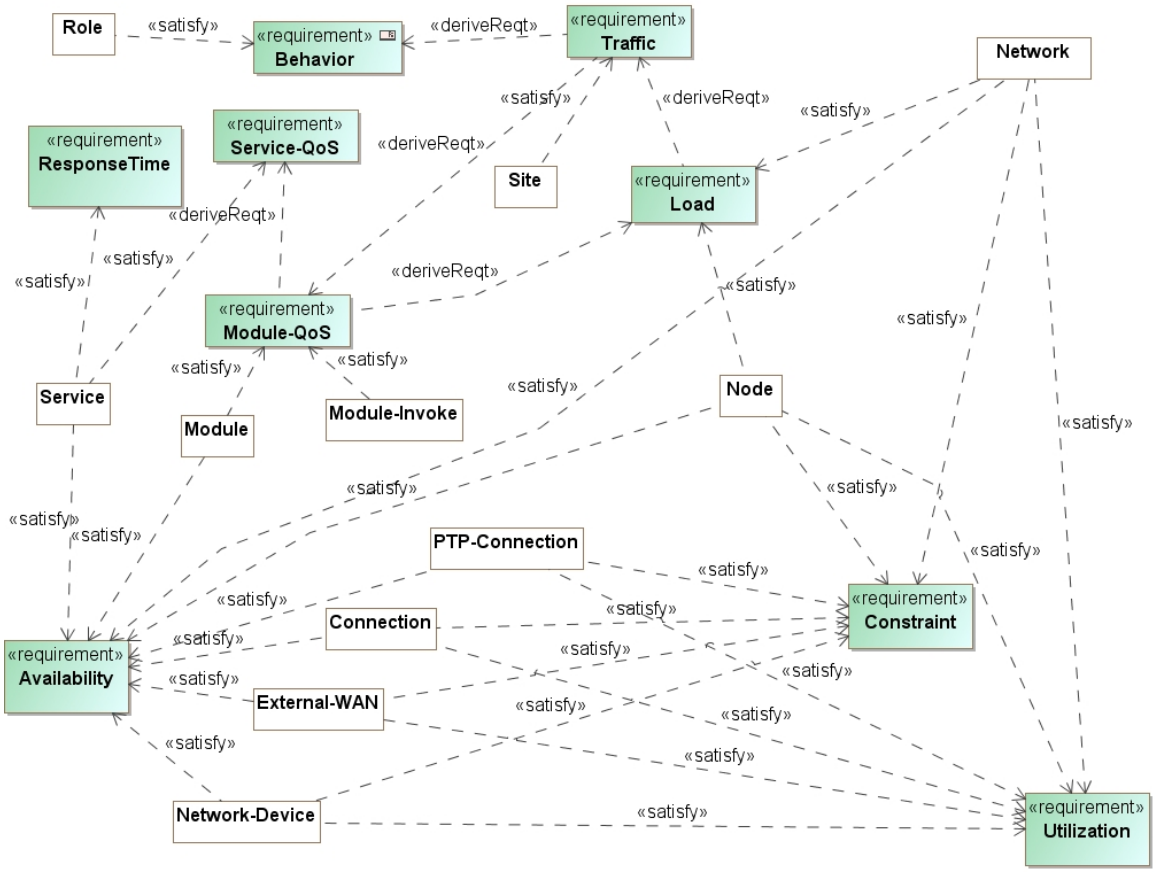


Figure 5.7: Defined NFR Requirements and their relations to other entities

aggregation of the values of the corresponding *service-QoS* requirement properties. Subsection 5.3.4 presents such an estimation.

Topology Requirements: Sites satisfy *traffic* requirements, indicating the amount of information exchange between the allocated modules. *Traffic* requirement is described in terms of *incoming*, *outgoing* and *exchanged* traffic. Maximum and average values are estimated. It is derived from *module-QoS* and *behavior* performance requirements as indicated in Figure 5.7 and it is estimated each time there is a change in allocations performed within *Topology* View. Subsection 5.3.4 presents this computation.

Network Infrastructure Requirements: Networks and network nodes are characterized by capacity indications, for example throughput, storage, speed or processing power. Their definition by the system architect must take into account constraints applied by existing infrastructure, availability, utilization and load requirements, as indicated in Figure 5.7. *Load* requirements are estimated based on *module-QoS* and *traffic* requirement properties satisfied by entities allocated to the specific network infrastructure component (for example modules allocated to a specific network node). Subsection 5.3.4 presents an algorithm to calculate the derived attributes of traffic requirements.

In order to effectively define EIS architecture, the system architect should ensure that all performance requirements are fulfilled. In SysML a *test case* determines whether the system meets specifications placed by requirements. A test case is a set of conditions or variables

which will be tested to ensure requirements are met. Next chapter will explain the equivalent of the SysML test case that we are using to verify the defined requirements.

5.3.2 SysML Extension to support NFRs

Requirements in SysML are described, as stereotypes of *Class*, in an abstract, qualitative manner, since they are specified by two properties, *id* and *text*, corresponding to a simple description. However, SysML specification suggests to use the stereotype mechanism to define additional properties for specific requirement types. Requirements can be grouped in packages based on common characteristics, such as their category (for example functional or non-functional) or the activities they are related to (for example software or hardware requirements) forming a multi-level hierarchy.

SysML includes specific relationships to relate requirements with other requirements (indicating the way they affect each other) or other model elements. The *containment* relationship, defined between requirements, indicates that the composite requirement is realized if and only if all the contained ones are realized. In this way, an abstract requirement may be composed of more specific ones, or a complex requirement may be described in a more detailed fashion. In the case of system design, the notion of composite requirements is essential to indicate the way a requirement defined for the system as a whole may be described in terms of the detailed requirements defined for system components. The SysML *deriveReqt* relationship indicates that a specific requirement is derived by others. However, the way requirements are specified is not depicted.

Requirements should be satisfied by model elements belonging to other diagrams (using SysML *satisfy* relationship). For this purpose, requirements may participate into other diagrams, enabling the exploration of the relationship between requirements and design decisions.

SysML provides the means to describe a set of tests, which should be performed to verify whether a requirement is satisfied by system components. To depict such an activity, the *test case* entity, included in requirement diagrams, is introduced. A test case is related to one or a set of requirements to handle their verification, while it is described through a behavior diagram (for example activity or state machine diagram) corresponding to the activity (as a set of tests) performed to verify related requirements. The way requirements are handled in SysML is summarized in Figure 5.8.

Since NFRs (for example performance requirements) are described using both qualitative and quantitative properties, a quantitative method, such as simulation, should be employed to produce the necessary data for their verification. In the related work (Chapter 3), tools and methods about simulating SysML models were discussed. The concept of the test case is not supported by any of them. In such case, the way system evaluation is performed, conforms to the corresponding simulation method. Thus, the definition of test cases is of less importance, since they could only be used to specify the conditions under which the system should be

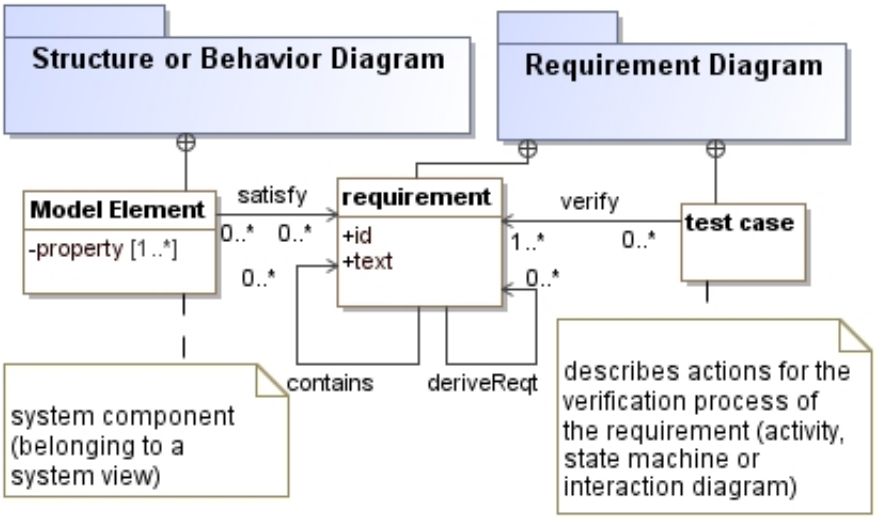


Figure 5.8: SysML Requirement representation

evaluated and not the evaluation method itself [95]. Furthermore, the results of the tests performed either by a test case or using simulation to verify requirement satisfaction are not included in SysML models. Such information is crucial for the system engineer to adjust system design or relax imposed requirements.

When SysML is utilized for system design, as for example in EIS architecture model-based design, NFRs are emphasized. To be accurately defined, NFRs should be described using quantitative properties, in a similar fashion as the non-functional properties defined in MARTE profile [81]. Since, NFRs may not always be described in an exact fashion, value deviation of quantitative properties should be allowed, to indicate for example that the response time for a specific service should be 4 to 5 seconds. In the same rational, maximum, minimum or average values should be described. Thus, more than one properties should be available for their description.

Furthermore, derived quantitative properties of NFRs should be automatically estimated. The *deriveRqt* relationship indicates only the fact that the derived requirement is related to one or more others. It does not provide any information about the way the requirement may be derived. The derivation may be depicted by indicating the way its quantitative properties are estimated, and this estimation is based on properties of the corresponding requirements. Thus, a *computation formula* property should be defined. The computation formulas may involve heuristics and become complicated. SysML requirement entity must be extended to:

i. effectively represent the quantitative aspects of requirements and
ii. define the way derived requirements should be computed.

Constraints, specific purpose languages as VSL [82] or scripts can be applied to derived requirements to enforce the automatic computation of derived properties, while computation algorithms must also be integrated in the SysML model. It should be noted that the specification of computation formulas is meaningful only if it is actually executed and corresponding quantitative properties are calculated.

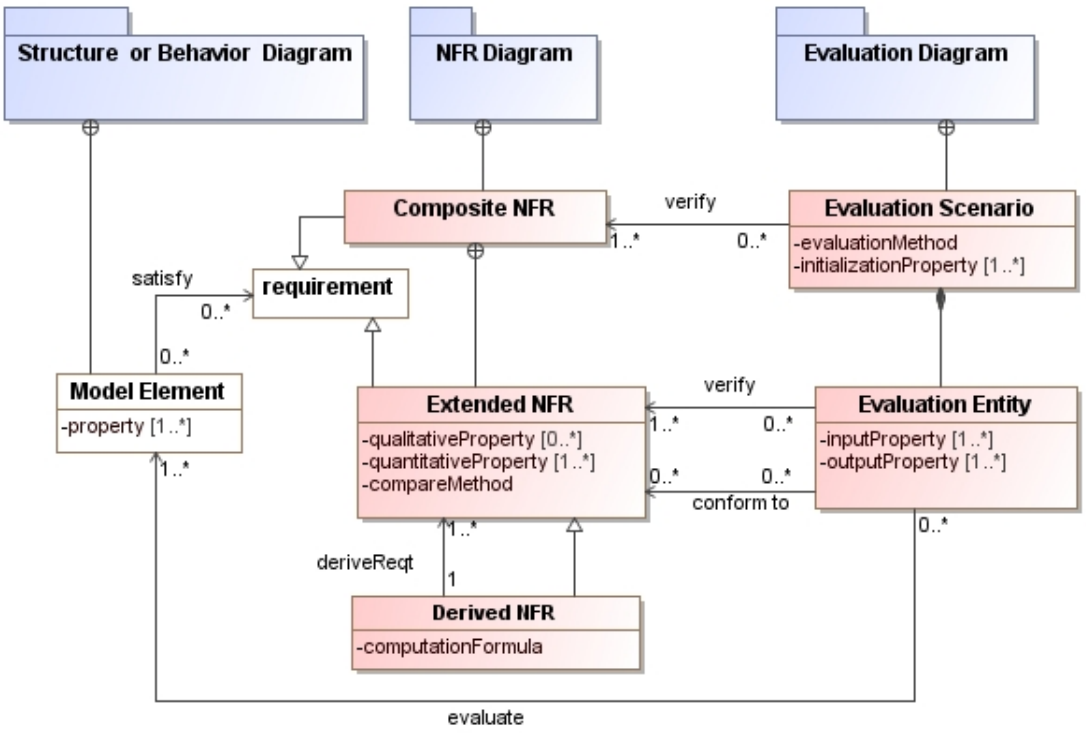


Figure 5.9: Extending SysML to explore NFRs

NFRs must be satisfied by system components included in any of the system design diagrams. In such case, in order to decide whether a NFR is verified, the designer may have to explore if the value of a quantitative property is satisfied by the related system components. To perform such a task, the comparison of specific evaluation results for each system component and related requirements properties should be performed, leading to the necessity of integrating evaluation data into the system model.

The SysML test case, as a concept, is focused on depicting how to evaluate model element satisfying a specific requirement, while integrating evaluation results into the system model is not considered. In the case of system design, NFRs are verified in a quantitative fashion by evaluation scenarios instead of test cases. An *evaluation scenario* should facilitate both:

- the definition of the conditions under which the system will be evaluated (probably using simulation) and
- the depiction of the evaluation results, so that the system engineer may be directly informed of requirement verification.

An *evaluation scenario* comprises of evaluation entities, used to *evaluate* model elements, to *verify* the corresponding requirement or requirements and can be described with block definition diagrams. Since an evaluation scenario is introduced to specify the conditions under which the system design should be explored, it involves the evaluation of all model elements, thus it is used to verify a composite, abstract NFR (for example the system performance must be high), constituting of specific ones (e.g., service time should be between 3 to 5 seconds). When an evaluation scenario verifies a composite requirement or a set of requirements, it should be used to verify all the included requirements.

Regardless of the method used to perform system evaluation, evaluation elements have *input* properties, related to evaluated model elements, and *output* properties, depicting evaluation data. Based on the value of output properties, requirements are verified. In the case of NFRs described in a quantitative fashion, an appropriate comparison method should be defined for the specific requirement, based on the output properties of all related evaluation entities. Such a method could be defined for example using a SysML *Parametric diagram* or executable scripts, associating requirement quantitative properties to evaluation entity output properties.

As already mentioned, to simulate a SysML model using a specific simulation method, simulation-specific characteristics should be included in the model. Such properties may be incorporated into evaluation entities, thus evaluation specific information does not have to be included in a system model designed by the system engineer, promoting discrete activity independence.

During system design, NFRs may also used to depict specific behavior forced on system components (for example the way a traffic generator may behave under heavy traffic conditions). In such a case, there is no point in verifying the requirements. The corresponding evaluation entity may *conform* to them, since they specify conditions under which the system design should be evaluated. The same requirement or requirements may be verified more than once, by evolving evaluation scenarios, as the system design is re-adjusted. Evaluation data and conditions included in them should be integrated in the SysML model. Thus, evaluation scenarios should be grouped into a distinct diagram, named *Evaluation Diagram*. The way basic SysML concepts are extended to handled NFRs for system design is summarized in Figure 5.9.

5.3.3 NFR Representation

As seen in Figure 4.10, NFR view comprises requirements that are satisfied by entities of the three aforementioned views (called design views) and are verified by elements of the *Evaluation* View. Table 5.4 presents these requirements and the related entities that satisfy them. All requirements are defined as stereotypes of SysML *requirement* entity, while additional stereotype attributes are defined to accommodate specific requirement properties. Requirements may be derived from other requirements, while all of them are treated as internal entities, since they are defined on the context of EIS architecture design. As stated earlier, requirements in SysML are described in an abstract, qualitative manner, since they are defined using a name and a description. In the case of EIS architecture design, NFRs should be more accurately descriptive using quantitative properties. Furthermore, derived requirement properties should be automatically computed by combining specific attributes of requirements and allocation entities. Though, SysML supports NFRs description, SysML requirement entity was heavily extended to effectively represent the quantitative aspects of requirements and the way they derive from each other.

Table 5.4: Requirements and their relationship with other model elements

EIS Profile entity	Associated Elements	Derived from
Load-Req (derived)	Network, Server, Workstation	Traffic-Req
Availability-Req	Server, Workstation	
Traffic-Load-Req (derived)	Site	Module-QoS-Req
Utilization-Req	Network, Server, Workstation	
Service-QoS-Req	Service	
Module-QoS-Req (derived)	Module	Service-QoS-Req
Response-Time-Req	Service	
Behaviour-Req	Role	
Constraint-Req	Network, Node, Connection, External-WAN, PTP-Connection	

Table 5.5 presents the entities that extend the SysML requirement entity with their attributes and the corresponding constraints.

Table 5.5: Non-Functional Requirements View Entities

Stereotype	Base Class	Properties	Satisfied by	Constraints
NFR View	Reqts. Diag.			Only NFRV stereotypes participate in it and all defined constraints must be validated
Service-QoS	Reqt	Type Value	Service	Types are: Processing, Storage, Traffic Value must be defined for all types
Response-Time	Reqt	Value Deviation	Service	Value and deviation must be defined
Module-QoS	Reqt	type max-value avg-value deviation comp. form	Module Module-Invoke	Types are: Processing, Storage, Traffic Derived by corresponding Service-QoS requirements
Behavior	Reqt	ActDistrFunc Mean StdDeviation	Role	Attributes are obligatory One Behaviour requirement per Role
Traffic-Load	Reqt	type max-value avg-value deviation comp. form.	Site	Types are: in,out,within Derived by allocated modules Module-QoS requirements
Load	Reqt	type max-value avg-value deviation comp. form.	Network	Types are: in,out,within Derived by allocated sites

5.3.4 NFR Derivation

SysML includes specific relationships to associate requirements with other requirements (indicating the way they affect each other) or other model elements. The *containment* relationship, defined between requirements, indicates that the composite requirement is realized if and only if all the contained ones are realized. In UML, *realization* is defined as a specialized abstraction relationship between two sets of model elements, one representing a specification (the supplier) and the other represents an implementation of the latter (the client). In this way, an abstract requirement may be composed of more specific ones, or a complex requirement may be described in a more detailed fashion. In the case of system design, the notion of composite requirements is essential to indicate the way a requirement defined for the system as a whole may be described in terms of the detailed requirements defined for system components. The *deriveReqt* relationship indicates that a specific requirement is derived by others. Since relationships do not have properties, the way derived requirements are specified is not depicted.

Derived requirements have attributes that their values are related to values of other, interrelated requirements. A derive relationship between a derived requirement and a source requirement is based on analysis. A derive relationship often shows relationships between requirements at different levels of the specification hierarchy. There are many ways to define the derivation. OCL could be used to describe the derivation. If the derivation formula is complex, an algorithm (or a heuristic method) could be used to define the derived attributes. The implementation of the algorithm could be done in any programming language that a design tool could support, even with a call to external program.

Derived requirements are used in order to provide indications to system designer about the required resources (storage, processing and traffic). They could be considered as estimations where specific QoS requirements are calculated to help the designer to define allocation policies. In the case of EIS profile, the implementation language is the Java language, as it is supported by the MagicDraw [3] modeling tool, that was used for the profile definition and implementation.

The following subsection present the derivation algorithms for the following derived requirements:

- **Processing Module-QoS** requirement: the derived attributes are the $\mathrm{avg}-\mathrm{value}$ and $\mathrm{max}-\mathrm{value}$.
- **Traffic Module-QoS** requirement: the derived attributes are $\mathrm{avg}-\mathrm{value}$ and $\mathrm{max}-\mathrm{value}$.
- **Traffic-Load** requirement: the derived attributes are $\mathrm{avg}-\mathrm{value}$ and $\mathrm{max}-\mathrm{value}$.

Derivation of *processing* & *storage* module-QoS requirement

Processing and *storage module-QoS* requirements are defined for each *module*. These requirement capture the average and maximum values of the processing and storage resources

that are required from the services that are belonging to a module.

1: **for** i = 1 to m (where m is the number of modules in $\{mod\}$) **do**
2: **for** j = 1 to s (where s is the number of services in $\{srv\}_i$) **do**
3: **for** k = 1 to r (where r is the number of roles in $\{role\}_j$) **do**
4: **for** t = 0 to 23 (where t is an hour of a day) **do**
5: **if** $StartTime_k \leq t \leq EndTime_k$ **then**
6: $SR_j[k,t] = numberOfOccurences_k$
7: $SRavg_j[k,t] = numberOfOccurences_k * percentage_{kj}$
8: **else**
9: $SR_j[k,t] = 0$
10: $SRavg_j[k,t] = 0$
11: **for** t = 1 to 23 **do**
12: $Smax_j[t] = \sum_{k=1}^{r} SR_j[k,t]$
13: **for** t = 1 to 23 **do**
14: $Savg_j[t] = \sum_{k=1}^{r} SRavg_j[k,t]$
15: **for** t = 1 to 23 **do**
16: $Sproc_j[t] = Smax_j[t] * proc_s$
17: $Sstor_j[t] = Smax_j[t] * stor_s$
18: $Savgproc_j[t] = Savg_j[t] * proc_s$
19: $Savgstor_j[t] = Savg_j[t] * stor_s$
20: **for** t = 1 to 23 **do**
21: $Mproc_i[t] = \sum_{j=1}^{s} Sproc_j[t]$
22: $Mstor_i[t] = \sum_{j=1}^{s} Sstor_j[t]$
23: $Mavgproc_i[t] = \sum_{j=1}^{s} Savgproc_j[t]$
24: $Mavgstor_i[t] = \sum_{j=1}^{s} Savgstor_j[t]$
25: $proc_i = \max_{t=1}^{23} Mproc_i[t]$
26: $stor_i = \max_{t=1}^{23} Mstor_i[t]$
27: $avgproc_i = \frac{\sum_{t=1}^{23} Mavgproc_i[t]}{x}$ where x the number of $Mavgproc_i[t] \neq 0$
28: $avgstor_i = \frac{\sum_{t=1}^{23} Mavgstor_i[t]}{x}$ where x the number of $Mavgstorc_i[t] \neq 0$

Algorithm 1: Calculating the max-value and avg-value attributes of the processing and storage Module-QoS-requirement

A *module* is comprised of *services*. Each service satisfies a *Service-QoS* requirement, which is composed of three types: *processing*, *storage* and *traffic*. A *module-QoS* requirement also is composed of the same three types. To estimate the corresponding processing *module-QoS* requirement, we have to estimate the average and the maximum values of the required processing power and the storage capacity. The maximun value is estimated if all the services concurrently require processing power and storage for their active time.

Module-QoS requirement is defined for each *Module-Replica* in *Topology* view. The derived attributes are the maximum and the average processing power and storage that the services belonging to this module require.

For each module replica participating in *Topology* view which is replica of a corresponding module defined in *Functional* view ($\{mod\}$), maximum and average processing power and storage requirements are estimated as described in the following steps:

i. For each service belonging to the module $\{srv\}_m$, the roles initiating it, either directly or indirectly, are gathered in $\{role\}_s$ list.

ii. Maximun concurrent instances of each role initiating this service, called SR, are esti-

mated, based upon StartTime, EndTime and numberOfOccurences properties of this role on a daily basis (using 24 time-intervals lasting an hour).

iii. Average concurrent instances of each role initiating this service , called SRavg, are estimated on a daily basis (using 24 time-intervals lasting an hour), based upon StartTime, EndTime and numberOfOccurences properties of this role and the percentage property of initiation entities associating the role (either directly or indirectly) to the service .

iv. Maximum concurrent role instances initiating the service, called $\mathrm{Smax_i}$, are estimated for each time-interval.

v. Average concurrent role instances initiating the service, called $\mathrm{Savg_i}$, are estimated for each time-interval.

vi. Maximum processing power requirements imposed by the invocation of each specific service, called $\mathrm{Sproc_s}$, are estimated for each time-interval, based upon corresponding *service-QoS* requirement properties and maximum concurrent role instances initiating the service.

vii. Maximum storage requirements imposed by the invocation of each specific service, called $\mathrm{Sstorc_s}$, are estimated for each time-interval, based upon corresponding *service-QoS* requirement properties and maximum concurrent role instances initiating the service.

viii. Maximum processing *module-QoS* requirement, called $\mathrm{proc_m}$, is estimated as the maximum value of the sums of the maximum processing power requirements imposed by the invocation of each specific service it belongs to, computed for each time-interval (called Mproc).

ix. Maximum storage *module-QoS* requirement, called $\mathrm{stor_m}$, is estimated as the maximum value of the sums of the maximum storage requirements imposed by the invocation of each specific service it belongs to, computed for each time-interval (called Mstor).

x. Average processing power requirements imposed by the invocation of each specific service, called $\mathrm{Savgproc_s}$, are estimated for each time-interval, based upon corresponding *service-QoS* requirement properties and average concurrent role instances initiating the service.

xi. Average storage requirements imposed by the invocation of each specific service, called $\mathrm{Savgstor_s}$, are estimated for each time-interval, based upon corresponding *service-QoS* requirement properties and average concurrent role instances initiating the service.

xii. Average processing *module-QoS* requirement, called $\mathrm{avgproc_m}$, is estimated as the average value of the sums of the average processing power requirements imposed by the invocation of each specific service it belongs to, computed for each time-interval (called Mavgproc).

xiii. Average storage *module-QoS* requirement, called avgstor_m, is estimated as the average value of the sums of the average storage requirements imposed by the invocation of each specific service it belongs to, computed for each time-interval (called Mavgstor)

Algorithm 1 describes this computation process. The same algorithm calculates and the *storage-module-QoS* requirement.

Derivation of Traffic Module-QoS requirement

Traffic-module-QoS requirement is defined for each *module-invoke* relationship defined in *Topology* view, which is also derived from the services relations defined in *Functional* view. This requirement captures the average and maximum values of the networking resources that are required from the communicating services of these two module-replicas.

In order to find the maximum traffic between a module A and a module B, the following steps are performed [1]:

i. for each service belonging to module A, called srv_A,

ii. for each service belonging to module B, called srv_B,

iii. if a service of the module A (srv_A) invokes a service of module B (srv_B), we add this traffic to the $\text{total\_traffic}(\text{A} \rightarrow \text{B})$

iv. in order to find the maximum traffic that is exchanged between srv_A and srv_B (srv_A *invokes* srv_B) hourly , we multiply the value of the *traffic service-QoS* requirement that is satisfied by the srv_B ($\text{trafficQoSreq}_{\text{srv}_\text{B}}$) with the maximum instances of the roles that initiate this srv_A for each specific hour

v. the maximum traffic for the module invoke between module A and B ($\text{max}(\text{traffic} - \text{module} - \text{QoS}_{\text{A}\rightarrow\text{B}})$), is the maximum value of $\text{total\_traffic}(\text{A} \rightarrow \text{B})$ matrix

vi. in order to find the average traffic that is exchanged between srv_A and srv_B in a hourly basis, we multiply the value of the *traffic service-QoS* requirement that is satisfied by the srv_B with the average concurrent instances of the roles that initiate srv_B for each hour

vii. to find the average traffic requirement for the exchange between modules A and B, we get the average traffic requirement from all hours

The way that the derived *traffic module-QoS* requirement is estimated, is presented in Algorithm 3. The input, output and temporary data for this estimation are presented in Algorithm 2.

Load requirements concern the load imposed to EIS resources by EIS components allocated to them. They provide an indication for the system designer so as to have an estimation

[1] We have adopted the following assumption: a role ia active by calling services at specific hours a day, which are defined in startTime and endTime attributes

Input: $\{mod\}$: a list of Client and Server Modules of Functional View

Input: $\{smod\}$: a list of Server Modules of Functional View

Input: S_i: a list of the services that belong to a specific Module i

Data: $Smax$: a Table created for each service s holding the maximun number of all role instances responsible for the invocation of the service a specific hour of the day. It is assumed that all the associated roles invoke the service at the same time.

Data: $Savg$: a Table created for each service s holding the average number of all role instances responsible for the invocation of the service a specific hour of the day.

Data: $invokes(S_i, S_j)$: returns true if service S_i invokes service S_j otherwise returns false

Data: $Mtmax_{m_i \to m_j}$: a Table created for each pair of modules $m1, m2$ holding the maximum traffic that module $m1$ invokes to module $m2$ each hour of the day

Data: $Mtavg_{m_i \to m_j}$: a Table created for each pair of modules $m1, m2$ holding the average traffic that module $m1$ invokes to module $m2$ each hour of the day

Output: $maxtraff_{m_i \to m_j}$: the value of maximun traffic requirement between modules $_i1$ and m_j. It is stored in the max-value attribute of the traffic Module-QoS-requirement associated to invokation $m_i \to m_j$

Output: $avgtraff_{m_i \to m_j}$: the value of average traffic requirement between modules $_i1$ and m_j. It is stored in the avg-value attribute of the traffic Module-QoS-requirement associated to invokation $m_i \to m_j$

Algorithm 2: Input, Output and Temporary Data Structures

1: **for** $i = 1$ to m (where m is the number of modules in $\{mod\}$) **do**
2: **for** $j = 1$ to ms (where ms is the number of server modules in $\{smod\}$) **do**
3: **if** $i \neq j$ **then**
4: **for** $k = 1$ to S_i, $l = 1$ to S_j
5: (where S_i is the number of services in $\{srv\}_i$ and S_j the number of services to $\{srv\}_j$) **do**
6: **if** $invokes(S_{ik}, S_{jl})$ **then**
7: $Mtmax_{m_i \to m_j} += Smax(S_{ik}) * traffic(S_{jl})$
8: $Mtavg_{m_i \to m_j} += Savg(S_{ik}) * traffic(S_{jl})$
9: $maxtraff_{m_i \to m_j} = \max_{t=1}^{23} Mtmax_{m_i \to m_j}[t]$
10: $avgtraff_{m_i \to m_j} = \frac{\sum_{t=1}^{23} Mtavg_{m_i \to m_j}[t]}{x}$, where x the number of $Mtavg_{m_i \to m_j}[t] \neq 0$

Algorithm 3: Calculating traffic Module-QoS-requirements

of the software components that require resources. Similarly to *service-QoS* and *module-QoS* requirements, three types of load requirements are defined: *processing*, *storage* and *traffic*. The first two types are derived straightforward from the corresponding types of *module-QoS* requirements. *Traffic-load* requirements derivation is more complex, since it is based on the network communication of software components, which are allocated to nodes across the network.

Derivation of processing and storage Load requirement

Processing load requirements are defined for each node that is defined in atomic-network diagrams. In nodes there are allocated module-replicas, which have specific processing requirements that are derived from the services that are composed of. Two values are calculated for each processing load requirement: *average* and *maximum* value. Average value is calculated as the sum of the average values of the module-replicas that are allocated to each specific node. Maximum value is the sum of the maximum values of the module-replicas, accordingly.

Derivation of *traffic*-Load and Load requirements

Performance requirements are defined as extended NFRs. Besides *id* and *text* properties, quantitative properties are also defined. Maximum and average values are defined for *utilization* requirement and minimum and average values are defined for *availability* requirement, while the accepted deviation of values is also defined. The *load* requirement is described by *maximum value*, *average value*, *deviation* and *measurement unit* quantitative properties. *Availability* and *utilization* requirements are defined by the network designer. The *load* requirement, though, is a derived one, as shown by the corresponding stereotype. Thus, it is described by *computational formula* additional property. In practice, it is an estimation of the aggregated QoS parameters of the module replicas they serve as system resources. There are three types of load requirement: *processing*, *storage*, satisfied by nodes, and *traffic* satisfied by networks. *Traffic-load* requirement is related to *site-to-network* allocation decisions, and is derived by corresponding *traffic* requirements, which in turn are derived by *traffic-Module-QoS* requirement of *module-replicas* allocated to *sites*. More than one sites may be allocated to a network. When a site is allocated to a network, the *module-replicas* allocated to it, must be specifically allocated to network nodes belonging to this network. When changes are made to site allocation or network architecture (e.g., new sites are allocated/removed to/from a network), module-replica allocation to network nodes is also adjusted and the re-estimation load requirements is needed, to ensure that hardware elements composing the network architecture provide the requested quality of service to the software components.

The maximum and average value properties of *traffic-load* requirement are derived from corresponding *traffic* requirements properties, defined for each site that is allocated to the network satisfying this requirement. *Traffic* requirements, which are defined for each site, in-

dicate throughput requirements concerning the information flowing from this site to others and within the site. *Traffic-load* requirement is described by similar quantitative properties as the *load* requirement. The *destination* property is defined to indicate the site the information is flowing to.

To determine the traffic between two sites, named *source* and *target*, the following process is proposed:

i. get all modules of *source* site
ii. get all modules of *target* site
iii. for each module belonging to *source* site, if there is a *module-replica-invoke* relationship and the *target* module belongs to *target* site, then add the $\max$ and avg *traffic values* of the *traffic* requirement of this *module-invoke* relation to the corresponding values of the *traffic* requirement.

The computation formula of the maximum and average value of the *traffic-load* requirement, is a complex process defined by a heuristic algorithm presented in algorithm 4. In practice, it was integrated in corresponding MagicDraw plugin. The algorithm 4 estimates *traffic-Load* values in a recursive fashion, starting with *traffic-load* computation for atomic networks, based on atomic site traffic requirement values allocated to the specific network. To this end, matrix A is estimated based on traffic requirements between all atomic sites ($\text{traffic}_{s_k \to s_l}$ in STEP1). STEP 2 consists of *traffic-load* computation for all atomic networks. Since composite networks consist of other networks, a *traffic-load* requirement is estimated for a composite network, if and only if the *traffic-load* requirement has been estimated for all the networks belonging in the composite network. Thus, in STEP 3 *traffic-load* is estimated for all composite networks consisting only of atomic ones, and in STEP 4 *traffic-load* is estimated for all the rest composite networks. In STEP 5, *traffic-load* is estimated for all network connections defined in the *Network Infrastructure* view. As shown in algorithm 4, the traffic traveling through a network consists of internal traffic, exchanged between the sites allocated to it, and external traffic, propagated to other networks. In the case of a composite network, the traffic imposed to it may be estimated based on the external traffic of all the networks belonging to it. Since the computation of a network's internal traffic can be performed more efficiently, the network traffic is estimated using the total and internal traffic of its components.

5.3.5 NFR Verification

System designer uses design views to define software and hardware architecture and NFR view to impose performance requirements to model elements. The intension is to evaluate the proposed *system synthesis* and this is possible if an appropriate evaluation method is chosen and if all the defined requirements are finally verified by their corresponding system elements. There are two sets of requirements: the first set is providing input information for evaluation process, for example the *service-QoS* requirement defining the required resources

for its execution and the second set is used as the validation rule, to check if a specific evaluation process output is between a range defined in this requirement (Figure 5.10). In order to facilitate the evaluation process, a specific view that matches the input and output properties of each model entity and defines the evaluation configuration data is introduced.

Since this thesis emphasizes on performance NFRs, which are described using both qualitative and quantitative properties, a simulation, as a quantitative method, is employed to produce the necessary data for their verification. In such case, the way system evaluation is performed, conforms to the corresponding simulation method. A specific view that serve the evaluation process is introduced, to separate the system model from the verification process.

Evaluation view serves the following aspects:

1: let $S = \{s_0, s_1, \cdots, s_n\}$ be the set of all atomic-sites
2: **STEP 1:**
3: create a $(n^2 \times 3)$ matrix **A** = A[i, j] , where $n = |S|$, $|S|$ stands for the number of elements of the set S, as follows:
4:

$$A[i,j] = \begin{cases} s_k, & k = \lfloor i/n \rfloor,\ j = 0 \\ s_l, & l = i \bmod n,\ j = 1 \\ \text{traffic}_{s_k \to s_l}, & k = \lfloor i/n \rfloor, l = i \bmod n, j = 2. \end{cases}$$

where $s_k, s_l \in S$,
$\text{traffic}_{s_k \to s_l}$ is the aggregated traffic between sites S_k and S_l
5: **STEP 2:**
6: let P be the set of all atomic-networks
7: $\forall p \in P$: $S_p, S_p \subset S$ be the set of sites allocated to atomic-network p
8: $\forall p \in P$:
$\text{totalTraffic}_p = \sum A[x,2], \{x : A[x,0] \vee A[x,1] \in S_p\}$
$\text{inTraffic}_p = \sum A[y,2], \{y : A[y,0] \wedge A[y,1] \in S_p\}$
$\text{remove}(A[y,*]\{y : A[y,0] \wedge A[y,1] \in S_p\})$
9: **STEP 3:**
10: let C be the set of all composite-networks that are comprised only of atomic-networks
11: $\forall c, c \in C$, let Q_c be the set of atomic-networks belonging to c, $n = |Q_c|$
12: $\forall c, c \in C$, let S_c be the set of sites allocated to atomic-networks belonging in Q_c
$\forall c, c \in C : \text{totalTraffic}_c = \sum_{q=1}^{n} \text{totalTraffic}_q - \text{inTraffic}_q,\ q \in Q_c - \text{inTraffic}_c$
$\text{inTraffic}_c = \sum A[y,2], \{y : A[y,0] \wedge A[y,1] \in S_c\}$
$\text{remove}(A[y,*]\{y : A[y,0] \wedge A[y,1] \in S_c\})$
13: **STEP 4:**
14: let R be the set of all composite-networks that are comprised of networks for which traffic is calculated
15: $\forall r, r \in R$, let Q_r be the set of networks belonging to r, $n = |Q_r|$
16: $\forall r, r \in R$, let S_r be the set of sites allocated to networks belonging in Q_r
$\forall r, r \in R :$
$\text{totalTraffic}_r = \sum_{q=1}^{n} \text{totalTraffic}_q - \text{inTraffic}_q,\ q \in Q_r - \text{inTraffic}_r$
$\text{inTraffic}_r = \sum A[y,2], \{y : A[y,0] \wedge A[y,1] \in S_r\}$
$\text{remove}(A[y,*]\{y : A[y,0] \wedge A[y,1] \in S_r\})$
repeat step 4 until $R' = \emptyset$
17: **STEP 5:**
18: let T be the set of all network-connections
19: $\forall t, t \in T$,let n_1, n_2 be the two connected networks and S_1, S_2 the sets of sites allocated to n_1, n_2 respectively
$\forall t, t \in T :$
20: $\text{traffic}_t = \sum A[y,2], \{y : A[y,0] \wedge A[y,1] \in S_1 \cup S_2\}$

Algorithm 4: Estimating the load requirement for networks

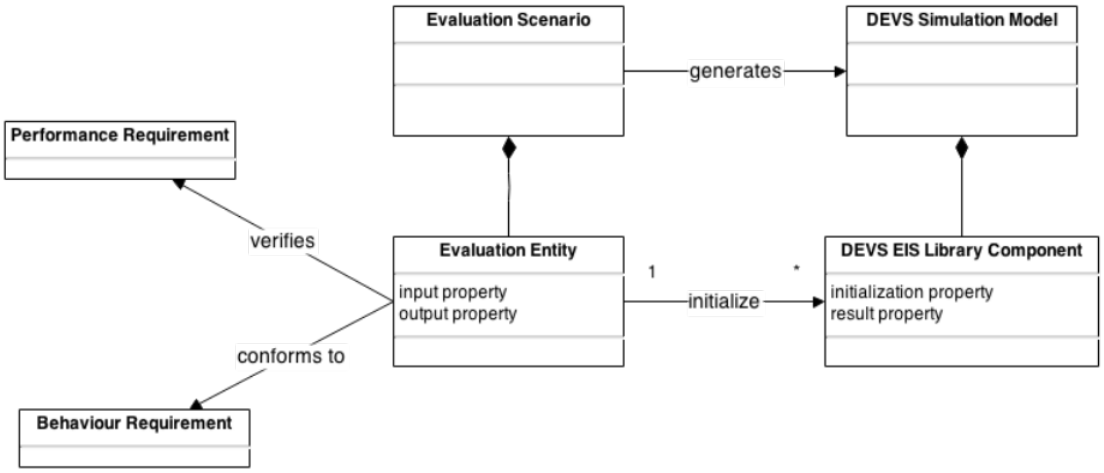


Figure 5.10: Two kinds of requirements: *performance* and *behavior*

- evaluation view is automatically created when an evaluation method is applied to system model;
- design data are separated from evaluation data;
- each evaluation method require specific configuration data, which can are differentiated per simulation scenario;
- a history of the evaluation scenarios facilitates the designer to explore alternative solutions;
- each evaluation scenario is a snapshot of the architecture design.

Algorithm 5 presents the process of requirements verification when the simulation results are incorporated into the system model.

1: **STEP 1:**
2: Gather simulation results and populate *output* attributes of *evaluation* entities
3: **STEP 2:**
4: If an evaluation entity satisfies a requirement, a validation rule is applied. Values of outpout attributes are checked against the defined values for the corresponding NFR. For all NFRs, a satisfying value is required, but there are cases where a range of values is more appropriate. Let s be a value of an output attribute of an evaluation entity, r be the value of the corresponding requirement and d be the deviation of this value. Then the following equation should be valid:

$$s \in [r - d, r + d]$$

5: **STEP 3:**
6: If $s \notin [r - d, r + d]$, this means that the validation rules failed, so the specific requirement is not verified. The user is notified for this event, by annotating the evaluation and the design entity.

Algorithm 5: NFR Verification

5.4 Summary

In this chapter the views concerning the design phase, where software and hardware architecture are specified, were analytically presented. Furthermore, the organization of NFRs was discussed, as well as, the estimation of derived requirements was presented with specific algorithms. Next section presents the *Evaluation* view in detail.

Chapter 6

Evaluating EIS Architecture

Contents

6.1 Outline

This chapter describes the *Evaluation* view. This view is introduced to enhance the definition specific EIS architecture configurations, which should be evaluated, as well as, to store the evaluation results of different configuration scenarios. Using these results, NFRs verification is done. Finally, performance evaluation is enabled via requirements verification.

6.2 Evaluation View

In order to effectively define EIS architecture, the system architect should ensure that all NFRs are fulfilled. *Evaluation* view is used to verify these requirements. System performance varies because of the ability to define different EIS architecture configurations in *Functional* and *Network Infrastructure* views. In practice, Evaluation view determines whether the proposed (current) architecture meets specifications placed by NFRs. Since EIS architecture de-

sign process may require to evaluate and properly adjust the proposed architecture more than once, *Evaluation* view consists of multiple *evaluation scenarios*, in order to evaluate alternative solutions. Since simulation is employed to evaluate the architecture design, these scenarios are specific simulation experiments.

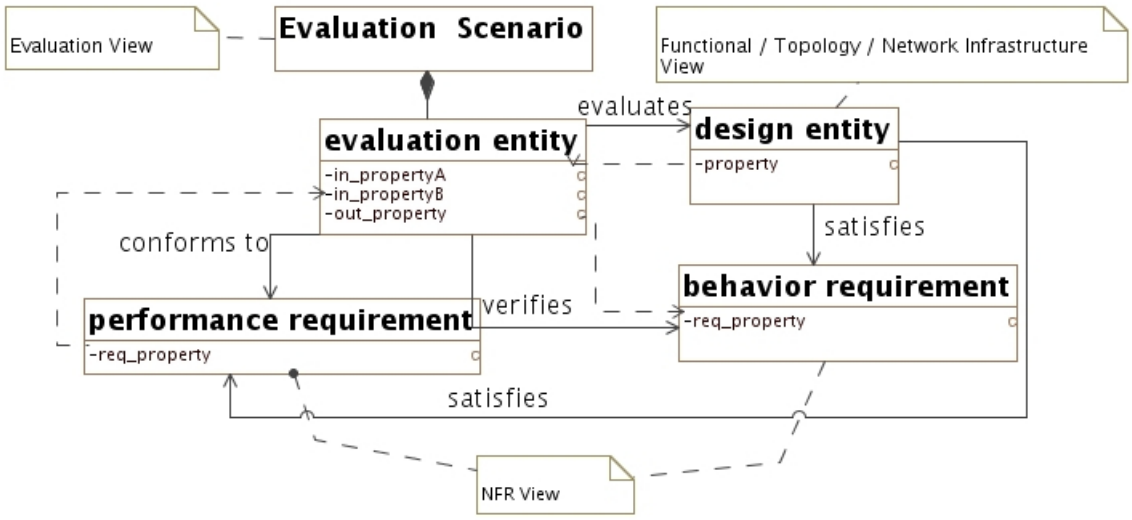


Figure 6.1: Interrelating EIS Performance Requirements, Design Entities and Evaluation Entities

An *evaluation scenario* is a set of conditions or variables which will be tested (simulated) to ensure that requirements are met. As indicated in Figure 6.1, an *evaluation scenario* is conducted to evaluate design decisions depicted in *Functional*, *Topology* and *Network Infrastructure* views, while its results are used to verify requirements defined in *NFR* view. When conflicts are discovered, changes are made to the system configuration by the system architect (e.g. *Functional*, *Topology*, *Network Infrastructure* or even *NFR* view) and a new *evaluation scenario* is initiated by the system architect until a satisfiable solution is reached.

The entities participating in a *evaluation scenario* and the way they are interrelated to with entities belonging to design views are depicted in Figure 5.1. An *evaluation scenario* evaluates network topology and network elements. For each design entity that will be evaluated, a corresponding evaluation entity is created (noted with *Eval* prefix). As such, *Eval-Node* entity evaluates workstation and server elements from *Network Infrastructure* view. An *atomic network* is represented as *Eval-Atomic-Network* and a *composite network* as *Eval-Composite-Network*. Interrelations between network evaluation entities have a direct mapping to interrelations of corresponding entities in *Network Infrastructure* view. *Roles* and *modules* defined in *Functional* view are allocated to nodes in *Network Infrastructure* view. Since allocation decisions are part of the EIS architecture, these entities should also be represented within a evaluation scenario by *Eval-Role* and *Eval-Module* entities. Furthermore, the *Eval-Service* entity is included corresponding to a *service* defined in *Functional* view, since it contains the necessary information for the execution of specific services grouped within a module.

Each entity in *Evaluation* view is created in order to evaluate a specific EIS architecture

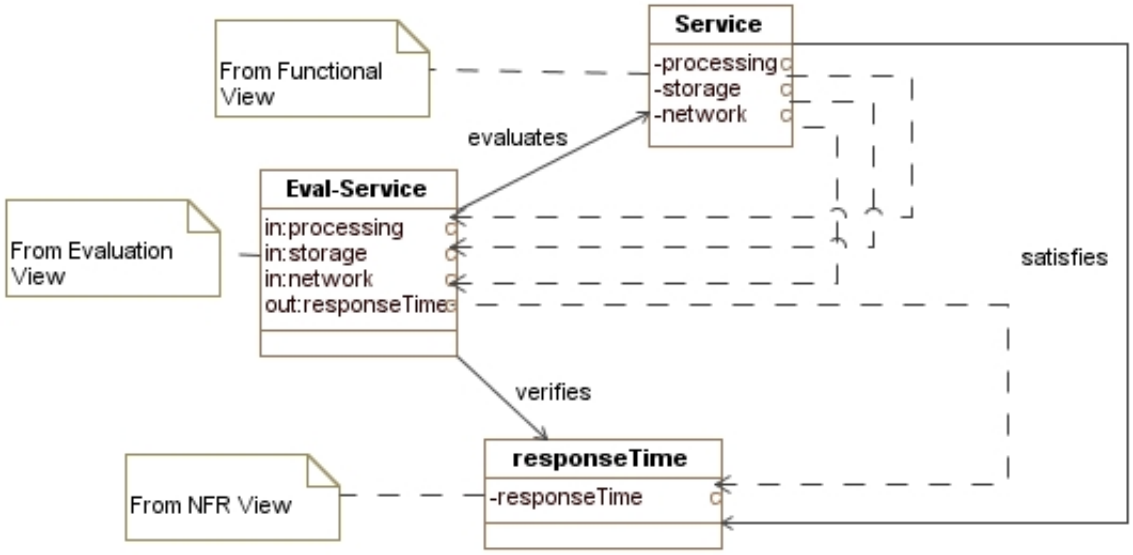


Figure 6.2: Eval-Service entity description

entity. Thus, NFRs related to the entity that the system designer wishes to verify, should also be related to the evaluation entity. An *Evaluation* view entity can only be related to requirements that the corresponding design entity should satisfy. For example, a *service satisfies* a *responseTime* requirement indicating maximum execution time. This requirement is verified by *Eval-Service* entity. Figure 6.2 illustrates this example.

Evaluation entities have *input* and *output* attributes. Input attributes correspond to attributes describing corresponding design entity. Output attributes indicate simulation results. To *verify* a requirement, the system designer (better yet, the design environment) should compare output attributes to corresponding requirement attributes, to check if there is a conflict. As indicated in Figure 6.2 for example, *Eval-Service* has as input attributes the amounts of processed, stored or transferred information that a service requires during its execution. These attributes are inherited from *Service* entity belonging to *Functional* view. Moreover, *Eval-Service* has as output attribute the average *responseTime*, which is computed when the evaluation scenario is executed. *ResponseTime* attribute of *Eval-Service* is compared to *responseTime* requirement that this *Service* has to satisfy. If a conflict has been identified, the system designer should alter the system design (e.g. modify the network architecture or the requirement itself) using *Functional* and *NFR* views and conduct a new evaluation scenario. The *Evaluation* view entities are presented in Table 6.1, where the specific evaluation diagram that they are participating is defined, while Table 6.2 illustrates them along with the defined constraints.

It is important to enable the system designer to maintain all performed evaluation scenarios in order to reach an acceptable solution, since they are part of the information used to make design decisions. Even if a acceptable solution is reached, information contained within evaluation scenarios may be used to pursue alternative solutions. This is facilitated by the fact that output attributes are directly compared to corresponding NFR attributes. There

Table 6.1: Evaluation View Entities in Diagrams

EIS Profile entity	SysML Entity	Part Of
Evaluation View	Block Definition diagram	-
Evaluation Scenario	Class	-
Software Architecture Diagram	Block Definition diagram	-
Hardware Architecture Diagram	Block Definition diagram	-
Eval-Service	Block	Software Architecture Diagram
Eval-Service-Replica	Block	Hardware Architecture Diagram
Eval-Server-Module	Block	Software Architecture Diagram
Eval-Client-Module	Block	Software Architecture Diagram
Eval-Server-Module-Replica	Block	Hardware Architecture Diagram
Eval-Client-Module-Replica	Block	Hardware Architecture Diagram
Eval-Role	Block	Software & Hardware Architecture Diagram
Eval-Workstation	System	Hardware Architecture Diagram
Eval-Server	System	Hardware Architecture Diagram
Eval-Node	System	Hardware Architecture Diagram
Eval-Connection	System	Hardware Architecture Diagram
Eval-PTP-Connection	System	Hardware Architecture Diagram
Eval-Network (Atomic & Composite)	Block Definition diagram	Hardware Architecture Diagram
Eval-Usage-Allocation	Allocation	Hardware Architecture Diagram

are two kinds of requirements: *qualitative* and *quantitative*. In the case of quantitative requirements, the exact comparison between arithmetic values is not always appropriate. Thus, an appropriate comparison method should be defined for a specific requirement.

The system designer may choose to evaluate the whole EIS architecture or a part of it. Conditions, under which the EIS architecture is evaluated, are defined by *behavior* requirements associated to *eval-Roles*, since they are used to represent different behavior of the same role, e.g. when a user initiate services, with *what* probability and *how* frequent.

Each evaluation entity is created in order to evaluate a specific EIS architecture entity and verify corresponding requirements. During system design, NFRs may also be used to depict specific behavior forced on system components (e.g., the way a traffic generator may behave under heavy traffic conditions). In this case, the corresponding evaluation entity should conform to them, providing input that could be used for the generation of the simulation model. The relation between design and evaluation entities, as well as corresponding requirements is depicted in Figure 6.1. To generalize this, it could be stated that a design entity satisfies two NFRs: performance requirement (depicting system performance restrictions) and behavior requirement (depicting system behavior). Only the first requirement must be verified by an eval-entity, since the second provides input properties to the evaluation entity, indicating the conditions under which the evaluation should be done.

Each evaluation scenario consists of two sub-views (diagrams), focusing on software and hardware architecture design respectively.

Table 6.2: Evaluation View Entities

Stereotype	Base Class	Properties	Constraints	Diagram
Evaluation View	Block Def. Diag.		contains one or more evaluation scenarios	-
Evaluation Scenario	Block Def. Diag.		contains all evaluation entities	-
Eval-Role	Block	in:activDistrFunct in:mean in:numberOfOccurences in:startTime in:endtime	evaluates Role entity and conforms to Behavior requirement for each role in an evaluation scenario only one behavior req is defined	H.A. & S.A.
Eval-Service	Block	in:QoS-traffic in:QoS-processing in:QoS-storage out:max-ResponseTime out:avg-ResponseTime	evaluates service entity and verifies ResponseTime & conforms to Service-QoS reqts	S.A.
Eval-Module	Block	in:numberOfOccurences in:list-of-services out:avg-QoS-processing out:max-QoS-processing out:avg-QoS-storag out:max-QoS-storage	evaluates module entity and verifies Module-QoS req. avg-QoS-processing is computed as the average of corresponding module replicas attribute max-QoS-processing is computed as the maximum of corresponding module replicas attribute avg-QoS-storage is computed as the average of corresponding module replicas attribute max-QoS-storage is computed as the maximum of corresponding module replicas attribute	S.A.
Eval-Module-Invoke	Dependency	in:invoking-Eval-Module in:invoking-Eval-Module out:avg-QoS-traffic out:max-QoS-traffic	defined between Eval-Modules evaluates Module-Invoke entity and verifies traffic Module-QoS requirement avg-QoS-traffic is computed as the average of corresponding module replicas attribute max-QoS-traffic is computed as the maximum of corresponding module replicas attribute	S.A.
Eval-Initiate	Dependency	in:percentage	defined between an Eval-Role and an Eval-Service	S.A.
Eval-Module-Replica	Block	in:instances in:list-of-services out:avg-QoS-processing out:max-QoS-processing out:avg-QoS-storage out:max-QoS-storage	evaluates Module-Replica entity and verifies Module-QoS req.	H.A.
Eval-Usage-Allocation	Allocation	in:instances	defined between an Eval-Role and an Eval-Workstation	H.A.
Eval-Software-Allocation	Allocation	in:instances	defined between an Eval-Module-Replica and an Eval-Node	H.A.

Software Architecture Diagram The entities participating in *software architecture* diagram correspond to entities from *Functional* view and are used to define the behavior of the software components during the evaluation of the proposed EIS architecture design; evaluate the corresponding functional view entity and verify the requirements that should be satisfied. Technically, it is a simple block definition diagram, resembling the *Functional* View.

Hardware Architecture Diagram *Hardware architecture* diagram entities correspond to entities from *Topology* and *Network Infrastructure* views, and are used to initialize an appro-

priate simulation model instance and evaluate the design entity and verify the corresponding requirements. *Hardware architecture* diagram consists of a network diagram, presenting the evaluation equivalent entities of *Network Infrastructure* view, and for each atomic network, a block definition diagram exists, having the evaluation entities of the corresponding atomic-network diagram of *Network Infrastructure* view.

Software architecture diagram defines the software architecture of replicas defined and distributed in *hardware architecture* diagram. Their combination is able to build an executable simulation model.

During the automated construction of software and hardware diagrams, validation rules are applied to model elements to ensure that the appropriate simulation input data exist and are valid. As soon as the simulation is performed and the results are incorporated, specific validation rules are performed to check the verification of all requirements, as stated in Section 5.3.5.

6.3 The Big Image: Views Interrelation

This section describes the interconnections of model elements: how a design entity is related to an evaluation entity and how is connected with requirements, in order to *verify*, *satisfy* or *conform* to them. Furthermore, requirements derivations and allocations are also depicted. All defined views are depicted in an abstract fashion, to show how they are interconnected through their containing elements connections.

To be able to track the interrelations of the model elements, we should place all model elements in a figure. This would explain how the *allocations*, *replications*, *evaluations* and *verification* are defined between design entities and evaluation entities, as they were presented in Figure. 4.10.

Focusing on Network Infrastructure view, in Figure 6.3, networks *A:atomic-network* and *B:atomic-network* are interconnected through a point-to-point connection. Network architecture design decisions are aiming to the allocation of sites to networks supporting them in order to minimize network traffic. In Figure 6.3, sites *S1:site* and *S2:site* are allocated to network *A:atomic-network*. To take such a decision, the network architect is based on the *Traffic* requirements of the information exchange within and between sites. For that reason, a *Load* requirement, satisfied by networks, is defined. In practice this requirement represents a performance QoS indication imposed to the network system resource. It depends on site allocation decisions and it is derived by corresponding sites *Traffic* requirements. It is recomputed each time a new network is added or deleted or a site allocation decision is made, while its computation is not trivial, since a heuristic computation algorithm should be applied.

Network configuration is performed taking into account, besides *load*, *utilization* and *availability* requirements defined in *Network Infrastructure* view, as well as existing network infrastructure restrictions, represented as *constraint* requirements.

Focusing on the *Topology* view, *traffic* requirements, associated to *sites*, depend on software allocation decisions. Software components, called *modules*, are defined in *Functional* view. Each of them contains *services* initiated by *roles*, or invoked by other services. Each service satisfies a *responseTime* requirement and impose specific requirements about the needed QoS from *processing*, *storage* and *networking* resources for its execution.

Based on *service-QoS* requirements, *module-QoS* requirements associated to each *module* are derived. Modules communication is described by a *Module_Invoke* relation, as shown in Figure 6.3 between modules *M1* and *M2*.

A *traffic-module-QoS* requirement indicates the QoS needed from the network resources to effectively perform data-exchange within acceptable time, e.g. so as all corresponding services have an acceptable response time. Based on this, the software designer decides on alternative software allocation polices, by

i. allocating one or multiple *module-replicas* for each *module* in different *sites*
ii. deciding which *module-replica* to use in module communication (depicted by the *module-replica-inkove* relation).

For example, in Figure 6.3, two replicas of module *M1* ($\mathrm{M1R1}$ and $\mathrm{M2R2}$) are allocated to site *S1:site*. Both of them are invoking the single replica of module *M2*, called $\mathrm{M2R1}$ allocated to site *S2:site*. *Traffic* performance requirement depends on module replica allocation decisions and is derived by *module-Qos* requirements associated to *module-replica-inkove* relation.

Module-replicas are allocated to nodes (either workstation or server) belonging in an atomic network. In Figure 6.3, the module replica $\mathrm{M1R1}$ is allocated to *W1:workstation*, which satisfies the *W1: Pro Load* processing-Load requirement. This requirement is derived by the corresponding *Proc Module-QoS* requirements (e.g., $\mathrm{M1R1 : ProcModule - QoS}$) of all the module replicas running on that workstation (the software components that are executing on that node).

As shown in Figure 6.3, the *load* requirement for a network (A:Load), defined by the network designer in *Network Infrastructure* view, is derived by a set of *traffic-Module-QoS* requirements defined by the software designer in *Functional* view ($\mathrm{M1M2 : TraffModule - QoS}$). This means that *A:Load* is derived by the communication of software components that are allocated to nodes that are belonging to that network. Paragraph Derivation of *traffic*-Load and Load requirements analyzes this derivation. Analyzing this dependency is not evident, thus a number of discrete views and an enhance performance requirement description mechanisms, emphasizing requirement derivation and their association with specific design decision is needed to effectively design the system architecture.

In order to evaluate the proposed architecture design, *Evaluation* view is utilized. *Evaluation* view consists of two sub-views, focusing on software and hardware design respectively (as shown in the right part of Figure 6.3). The entities participating in software architecture diagram view correspond to *Functional* view entities and are used:

i. to define the behavior of the software components during the simulation of the proposed architecture design and

ii. to evaluate the corresponding Functional View entities (for example the *Eval-Module M2* evaluates the *M2* module and verifies the requirements that are satisfied by *M2*, based on the behavior of all of its replicas).

Hardware architecture diagram entities correspond to *Topology* (module replicas) and *Network Infrastructure* view entities, are used:

i. to initialize a corresponding simulation model instance and
ii. to evaluate the corresponding design view entities.

For example, the properties of *W1: Eval-Workstation* entity are used to verify the processingLoad requirement that the corresponding design entity *W1: workstation* satisfies *(W1: Proc Load)*.

When the evaluation process is completed, evaluation results (simulation results) are incorporated into evaluation entities. In that way, evaluation entities are checked against defined validation rules. If incorporated results are not in accordance with all related requirements that this entity verifies, this means that the defined requirements are not verified. System designer has to change the architecture so as to re-evaluate the new system architecture until all requirements to be verified.

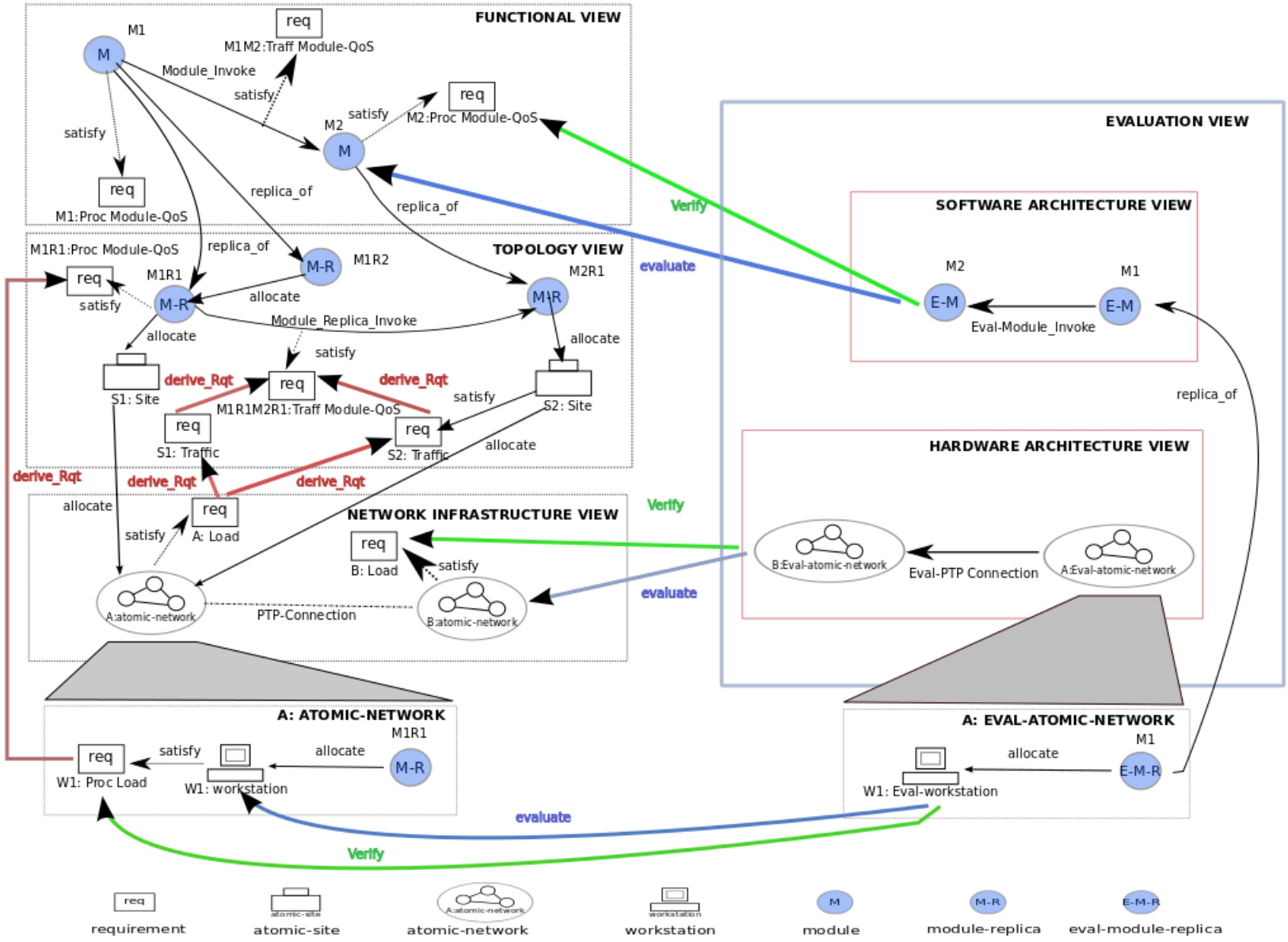


Figure 6.3: Performance Requirement Derivation and Verification in IS Architecture Design: An example

6.4 Automating the verification Process

To evaluate the performance of an EIS architectural design, system designer would like to have an integrated design environment that would support performance evaluation. This means that ideally the designer would not like to be involved in the evaluation process, for example he/she could perform evaluation without knowing the background processes such as model transformations, simulation framework initialization and simulation execution. To manage this, an automated evaluation process is presented in this section.

The steps required to accomplish the automation of the verification process, are:

- simulation framework selection
- simulation code generation
- simulation execution
- simulation results incorporation

6.4.1 Simulation framework

In the case of EIS, simulation frameworks supporting discrete time simulation are well suited, as users and software services generate requests on other services. The requests generate specific traffic on the network and processing and I/O load on the site of service. Stochastic functions may be used to define random behavior of specific parts of the system, like time handling and type of requests made by users.

Regarding the requirement for MDA compliance, the selected simulation framework must provide a standards-based specification for input simulation models, i.e. a MOF meta-model. This enables the definition and execution of standards-based transformations of EIS models (UML meta-model) to simulation models. Modelica and DEVS simulation frameworks both provide MOF meta-models for simulation models specification, in the notion described above. DEVS is appropriate for discrete time simulation by definition. Modelica is better suited for simulating systems with continuous behavior, but it can also be used for discrete time simulation. As stated in 2.4.2, an extension of DEVS was introduced, making possible to define simulation capabilities in a SysML model. From this work, the DEVS metamodel is utilized, as we have already defined a SysML profile for our model.

DEVS is a formalism allowing a hierarchical and modular description of the models. In the classic DEVS formalism, *atomic* DEVS captures the system behavior, while *coupled* DEVS describes the structure of system. DEVS coupled elements enable the definition of composite models in a way similar to SysML components and contain ports (input and output), other DEVS (Atomic or Coupled) elements and *couplings* (e.g., port inter-connections).

In this thesis, focus was given on defining behavior of DEVS atomic models and combining them in DEVS coupled models, rather than using existing simulation components. Therefore, the DEVS meta-model, proposed there, did not feature provisioning for this case. However, in the case of EIS, models are composed of large amounts of interconnected components of specific types (e.g. nodes, services, etc.). As simplification of the verification activity is one of

our main goals, there is no need for EIS engineers to define each component's behavior, using yet another (than the EIS) simulation specific profile. Therefore, the required simulation components where analyzed and implemented during the application of the approach for EIS, while initialization, composition and interconnection of the components emerge from the EIS model during the verification execution. The missing feature was introduced in the DEVS meta-model by extending it, as illustrated in Figure 6.4. In the previous version of the DEVS meta-model, component references of DEVS coupled components could only be references to other DEVS components, defined in the simulation model. In the revised version of the meta-model, they may also be references to existing simulation library components, in which case, required initialization parameters must be specified. The parameters may be single values, multiple values or other simulation library components.

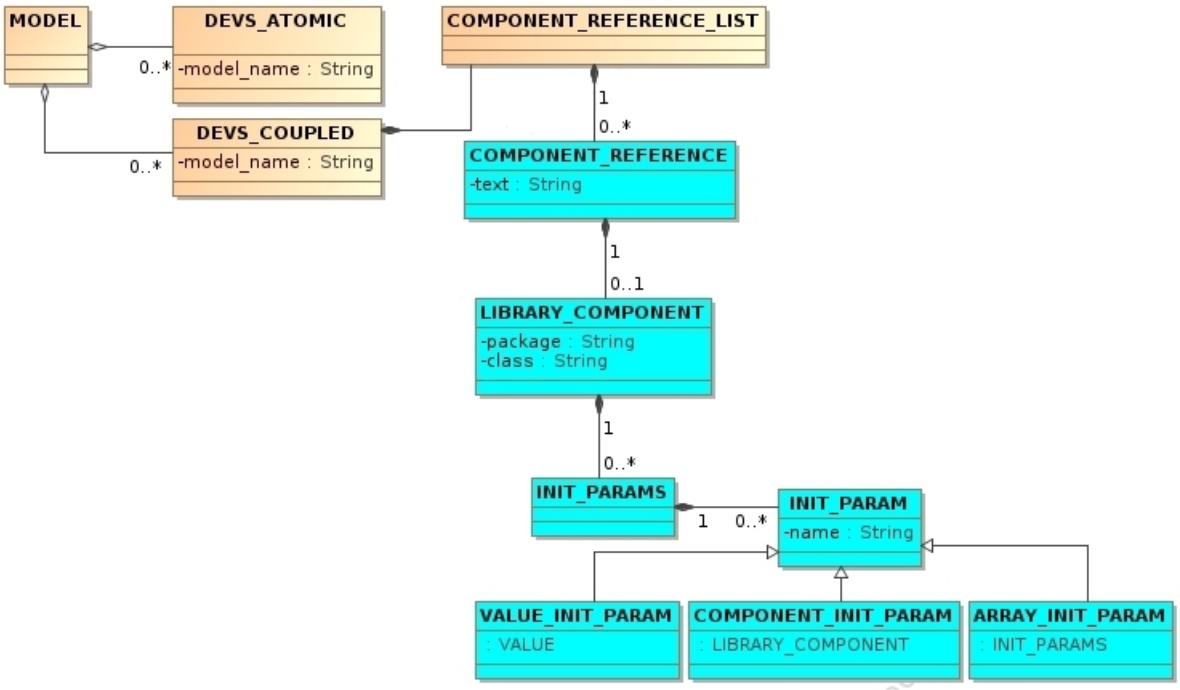


Figure 6.4: DEVS Meta-model extension

6.4.2 Generate executable simulation model

A valid executable simulation model should conform to the DEVS meta-model, mentioned in Figure 6.4 [74]. Thus, executable simulation model generation is performed with the transformation of the EIS model to the respective DEVS model.

This requires that a DEVS execution environment supporting the DEVS meta-model is selected. If no such environment is available, existing environments must be configured or extended, so that they can handle DEVS models as input. In our case, DEVSJava [121] was selected and extended with a transformation layer that translates complex DEVS models, that conform to the aforementioned meta-model, to DEVSJava executable code.

Regarding the generation of DEVS models from EIS models, the structure and relation-

ships of the latter were analyzed. The main model entities that affect overall performance were identified in both *hardware* and *software architecture* diagrams (that are included in the *Evaluation* view), like *Eval-Network*, *Eval-Node* etc.

Although, for EIS design purposes, it is better to define different aspects of EIS elements using distinct diagrams/views, the simulation model should capture the structure and attributes of a given concrete system each time. Additionally, verification and effectiveness of the simulation largely depends on the simplicity of the simulation model. Apparently, model structure and model element attributes and dependencies exist and can be used in the transformation independently of the diagrams they appear in, which are more useful for organization/presentation purposes. Therefore, a set of simulation components were defined as an equivalent to EIS software and hardware elements, as they could be combined in the context of a given evaluation scenario. This is coarsely depicted in Figure 6.5, where EIS model elements reside on the left side of the Figure and the respective DEVS model elements on the right.

Moreover, Figure 6.5 illustrates the high-level patterns of the conceptual mapping between EIS model elements and DEVS model elements. However, these patterns can be applied on large and complex configurations of EIS, with several hundreds of nodes.

It is clearly illustrated that simulation elements usually derive from the combination of more than one EIS model elements, with the exception of *DEVS_Simulation_Controller*. The latter was decided to be defined independently from the *DEVS_Scenario*, in order to separate simulation model structure from simulation execution context, like simulation duration or number of simulation executions.

The *EIS* part of Figure 6.5 includes the main entities of the *hardware architecture* diagram (Eval-Network, Eval-Node, Eval-Module-Replica-Invoke), the main entities of the *software architecture* diagram (Eval-Role, Eval-Module, Eval-Service) and their associations and dependencies. Also, the *Eval-Scenario* block provides information regarding the context of the evaluation that is used for the generation of the respective, top-most *DEVS_Scenario* and the *DEVS_Simulation_Controller* elements. These EIS entities are utilized for the generation of the respective DEVS elements (*DEVS_Network*, *DEVS_Node*, etc.), while associations and dependencies mostly indicate how DEVS model elements should be interconnected. In EIS, elements are organized under diagrams, which are combined with a scenario. On the contrary, DEVS elements are organized in a more strict hierarchy. The *DEVS_Scenario* is the root of the model containing the *DEVS_Simulation_Controller*, a set of interconnected *DEVS_Network* elements and a set of *DEVS_Nodes*. Each *DEVS_Node* is composed of a *DEVS_Processor*, a *DEVS_Storage*, a *DEVS_Network_Interface* and a set of *DEVS_Modules*, containing sets of *DEVS_Services*. Additionaly, the *DEVS_Roles* using each node are specified, as well as the *DEVS_Network*, where each node belongs.

In a lower level of this transformation scheme, EIS entities were further analyzed to identify their key attributes that determine the performance of the system. Equivalent attributes were defined in the respective simulation elements. The transformation handles their proper

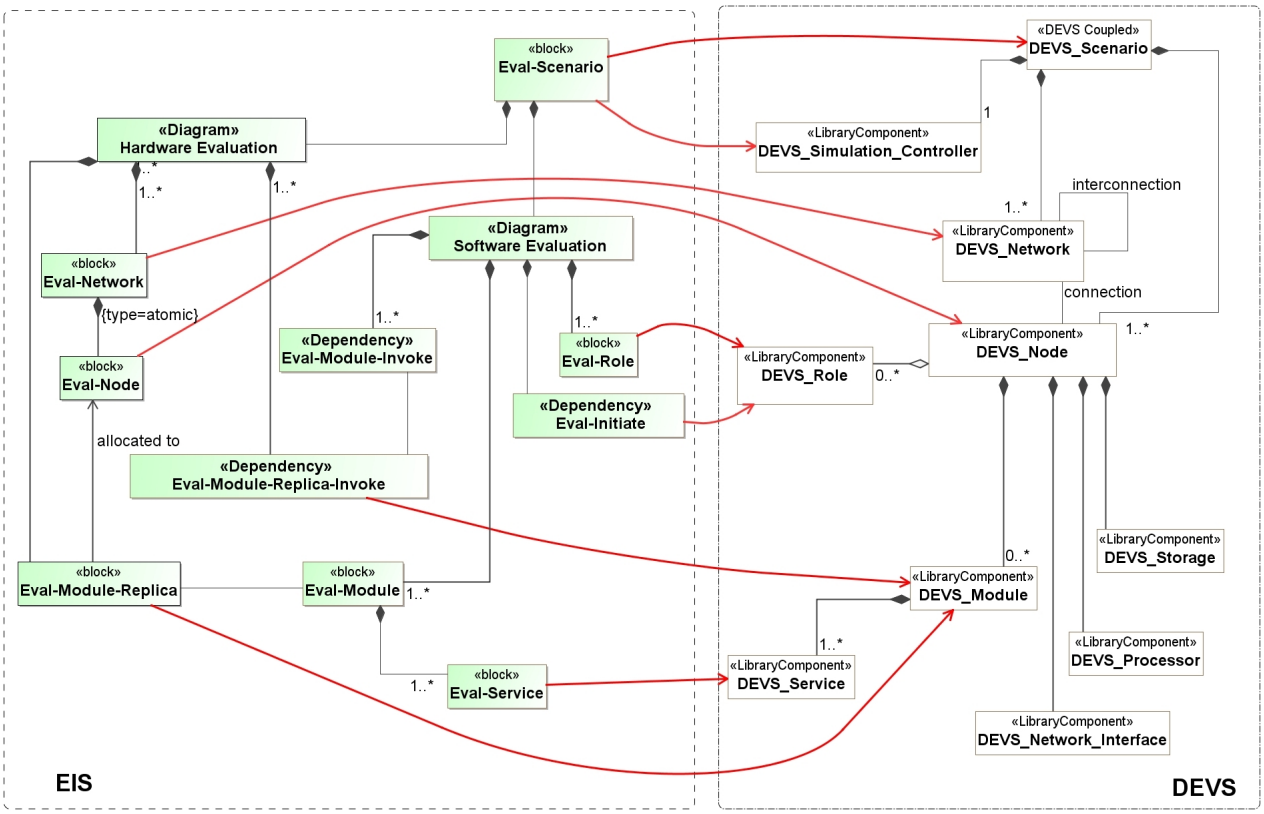


Figure 6.5: Outline of the EIS to DEVS model transformation

initialization, based on the values of the respective attributes of EIS elements. The possible entity interconnection schemes were also examined. Additional information derived from the combination with other EIS model elements, like the Eval-Initiate dependencies that indicate which services are initiated by each user role. Actually, a large set of attributes and associations of the EIS model, that cannot be depicted in the high level representation of Figure 6.5, are utilized in the implemented transformation.

DEVS_Scenario, *DEVS_Node* and *DEVS_Module* were implemented as DEVS *Coupled* components that are composed of other components. All other leaf elements were implemented as DEVS *Atomic* components with the expected behavior. The interested reader may refer to the source code of the aforementioned library components, in [122]. Each library component is a Java class that extends DEVSJava basic classes *ViewableAtomic* or *ViewableDigraph*. Simulation execution can be inspected via SimView, a free GUI framework for DEVSJava [121]. In *ViewableAtomic* classes, the behavior of the simulation component is defined as determined by the component state as affected by the reaction of the external events, with internal functionality. In *ViewableDigraph* classes, composite simulation components are defined, containing other interconnecting components. The behavior of *ViewableDigraph* components is defined by the composition of the behaviors of the contained components.

The fact that both meta-models (SysML and DEVS) are MOF-based, enables the use of standard transformation languages, like QVT. Therefore, the appropriate QVT relations were defined for the generation and interconnection of DEVS model elements from the respective

EIS model elements. The first relation of the EIS to DEVS QVT transformation is displayed in listing 6.1.

Listing 6.1: EIS to DEVS QVT transformation

```
transformation eis2devsMM(eis:uml, devs:Devs) {
    top relation eisScenario2DevsModel {
        scenarioName: String;
        checkonly domain eis scenario : uml::Class { name = scenarioName };
        enforce domain devs model : Devs::MODEL {
            DEVS_COUPLED = devsCoupled : Devs::DEVS_COUPLED {
                MODEL_NAME = modelName : Devs::T_Model_Name { text = scenarioName },
                COMPONENT_REFERENCE_LIST =
                    componentReferenceList : Devs::T_Component_Reference_List { },
                INTERNAL_COUPLING = internalCoupling : Devs::T_Internal_Coupling { } } };
        when { scenario.getAppliedStereotype('EvaluationView::Evaluation Scenario')->notEmpty(); }
        where {
            eisSimulationAttributes2ComponentReference(scenario,  componentReferenceList,
                    internalCoupling);
            eisEvalNetwork2ComponentReference(scenario,componentReferenceList,internalCoupling);
            eisIncludes2InternalCoupling(scenario.getModel(),internalCoupling);
            eisIncludesRev2InternalCoupling(scenario.getModel(),internalCoupling);
            eisEvalPTPConnection2InternalCoupling(scenario.getModel(),internalCoupling); } }
    ...
}
```

Executable simulation models contain all the information required, indicating the importance of their automated generation from system models, without the need for human interference. A part of a generated DEVS simulation model is presented in the listing 6.2. In this listing a DEVS Coupled scenario contains a SimController DEVS library component and a Network DEVS library component. Initialization parameters, derived from the EIS SysML model, are in the simulation model.

Listing 6.2: DEVS simulation model

```
<?xml version="1.0" encoding="UTF-8"?>
<Devs:MODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:Devs="urn:DEVS_MM.ecore"
xsi:schemaLocation="urn:DEVS_MM.ecore platform:/resource/Eis2DevsMM/metamodel/DEVS_MM.ecore">
  <DEVS_COUPLED>
    <MODEL_NAME text="aScenario"/>
    <COMPONENT_REFERENCE_LIST>
      <COMPONENT_REFERENCE xsi:type="Devs:T_Component_Reference" text="SimulationController">
        <LIBRARY_COMPONENT class="SimController" package="eis.library">
          <INIT_PARAMS>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="simulationTime">
              <VALUE type="Real" value="3600"/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="simulationRuns">
              <VALUE type="Integer" value="1"/>
            </INIT_PARAM>
          </INIT_PARAMS>
        </LIBRARY_COMPONENT>
      </COMPONENT_REFERENCE>
      <COMPONENT_REFERENCE xsi:type="Devs:T_Component_Reference"
      text="Composite-Network regional office 1 net evaluation">
        <LIBRARY_COMPONENT class="Network" package="eis">
          <INIT_PARAMS>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="throughput">
              <VALUE type="Real" value="100"/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Array_Init_Param" name="nodes">
              <INIT_PARAMS/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Array_Init_Param" name="networks">
              <INIT_PARAMS>
                <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="network">
```

```
                <VALUE type="String" value="Atomic-Network registry office 1 net evaluation"/>
              </INIT_PARAM>
              <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="network">
                <VALUE type="String" value="Atomic-Network datatacenter 1 net evaluation"/>
              </INIT_PARAM>
            </INIT_PARAMS>
          </INIT_PARAM>
        </INIT_PARAMS>
      </LIBRARY_COMPONENT>
    </COMPONENT_REFERENCE>
    ....
  </DEVS_COUPLED>
</Devs:MODEL>
```

Given that the required simulation components are already implemented in DEVSJava, the essential information contained in the generated DEVS model is the initialization, interconnection and configuration of such components. Therefore, as far as the DEVS meta-model compatible environment is concerned, we decided to implement a transformation of DEVS models (XMI) to the respective DEVSJava configuration class that instantiates and configures all EIS-related DEVSJava components, forming, this way, the executable DEVSJava code. This transformation was defined using EXtensible Stylesheet Language Transformations (XSLT), as it is basically a syntactic transformation that exploits initialization information in the DEVS model and creates the respective Java declarations and statements. Therefore, the generated DEVS simulation models are executed in the DEVSJava environment after an automated transformation.

To be able to execute the simulation, the corresponding library components were developed [122]. Table 6.3 presents them along with the corresponding EIS entities that they implement.

Table 6.3: DEVS library components

Library	Type	EIS model element
Module	Coupled	Client-Module, Server-Module
Network	Atomic	Composite Network
NetworkInt	Atomic	Atomic-Network
Node	Coupled	Server, Workstation
Processor	Atomic	Processing Unit
Role	Atomic	Role
Service	Atomic	Service
Storage	Atomic	Storage Unit
SimController	Atomic	

6.4.3 Simulation Execution

Having produced the DEVSJava simulation code, the simulation is executed and anyone can watch the progress through the DEVSJava simulation viewer (Figure 6.6).

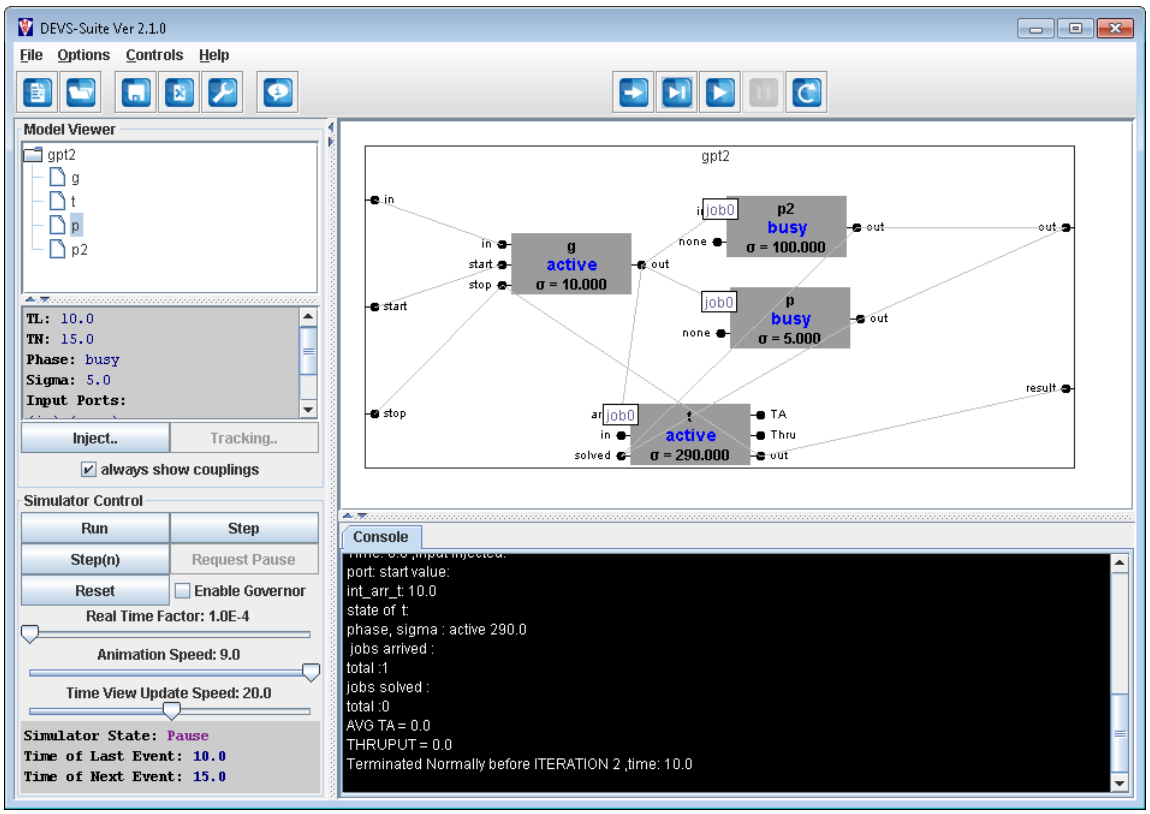


Figure 6.6: The DEVS suite simulation viewer

Simulation results are stored in XML format, so as to be able to be incorporated into the design environment.

6.4.4 Simulation results incorporation

Having generated the executable simulation model, simulation may be executed and the simulation results must be incorporated in the EIS model. Thus, requirement verification through the profile, independent from the simulation environment is possible.

To exploit simulation results and import them in the EIS model via standards-based model manipulation approaches, they should be provided in a standard representation, i.e. according to a MOF meta-model. A simple meta-model for the representation of performance estimation, was defined for that purpose, as illustrated in Figure 6.7.

For each EIS model element there are two recorded properties: one holding information related with the identification of the model element, such as the *name*, the *stereotype name* and a *unique identifier* and the second holding information related with the simulation results for that model element, such as a key-value pair for each attribute of the model element that will be imported to the system model. Having such a representation, the incorporation renders to one step procedure.

Having the simulation results imported in the system model, the only thing that the system designer should do is to run *validation rules*. Validation rules are utilized to indicate the evaluation entities associated to non-verified requirements. Evaluation entities not verifying

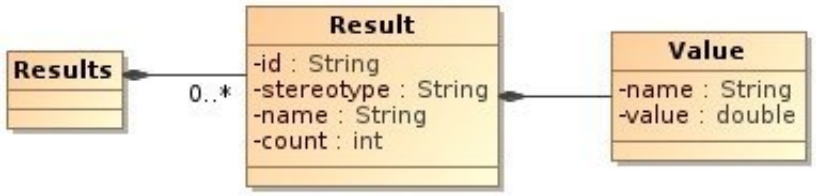


Figure 6.7: Simulation results meta-model

a model constraint are marked with red color, and the designer is able, by clicking on them, to identify the non-verified requirement. The verification process have been described in detail in Section 5.3.5

All these tasks require expertise in several technologies and standards, as SysML, DEVS, QVT, Java and the EIS domain. However, once implemented, the benefits from their combined use are available for multiple uses by numerous, interested EIS engineers.

6.5 Implementation

The following steps were accomplished in order to implement the proposed approach:

i. EIS metamodel definition
ii. Implementation of EIS profile in MagicDraw [3] modeling tool, which was accomplished in two steps:
    - profile definition, extending SysML entities [123]. EIS profile is a set of diagrams, each of them holding the corresponding stereotypes with their tagged-values
    - validation rules and constraints implementation, using the plugin extension mechanism of MagicDraw [124], writing Java code
iii. definition of the DEVS metamodel
iv. definition of the QVT transformation to produce the DEVS model from the system model
v. generation of DEVSJava simulation code from the DEVS model
vi. implementation of the EIS library components [122] using the DEVSJava [125] simulator
vii. incorporation of the simulation results into the system model
viii. verification of the requirements, which is based on the simulation results

Many software tools were exploited throughout this thesis. The selection criteria were to use open source software when possible or widely accepted commercial ones. A key issue was to select a modeling environment that would support the SysML, as officially specified my OMG. The complete list of software that was used, follows.

- MagicDraw [3] and SysML plugin [123] for MagicDraw, to define the EIS profile which is supported by the EIS plugin
- Netbeans [126], as IDE for the development of EIS plugin
- Medini QVT [127] eclipse plugin, for the transformation of EIS model to DEVS model
- Java Architecture for XML Binding (JAXB), for the incorporation of simulation results

- DEVSJava [121], to execute the simulation

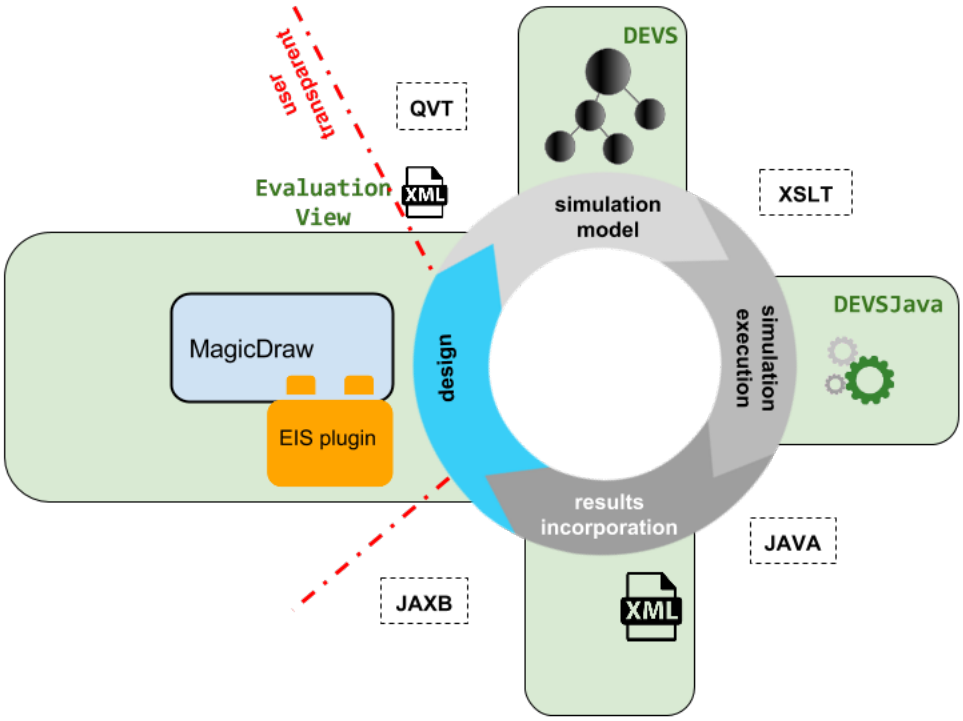


Figure 6.8: Implementation overview

Effort was given in order to provide an automated environment to system designer (see Figure 6.8). The design environment is based on a well-known commercial product, MagicDraw (Academic Standalone Edition) with SysML plugin, which provides an open API [124] to encourage programmers to implement custom plugins for their specific domain. From the designer's perspective, the evaluation process is transparent (he is not involved in the simulation code generation, execution and incorporation of results). Especially, the EIS plugin, incorporated the following functionality:

- Custom diagrams to support EIS profile views. Each diagram has a toolbar with the corresponding view elements (enhanced with diagram constraints). These diagrams are:
  - Functional View
  - Topology View
  - Network Infrastructure View
  - Atomic Network Diagram
  - NFR View
  - Evaluation View
  - Software Architecture Diagram

  - Hardware Infrastructure Diagram
  - Evaluation View Atomic Network Diagram
- Validation rules to ensure model consistency
- Constraints implementation in EIS plugin.
- Auto-generation of evaluation scenarios
- Invocation of shell scripts
- Invocation of model transformations
- Incorporation of simulation results
- Requirements verification

6.6 Summary

This chapter presented the evaluation phase of the proposed approach. *Evaluation* view was analytically described. An abstract overview of the approach, involving all defined views with their interrelations was presented, to help the reader understand how we manage to produce simulation executable code from SysML models. Finally, implementation issues were discussed, to prove that the approach was based on the integration of standards and tools, in accordance in INCOSE. The next chapter presents an extensive case study, showing the design and evaluation process from the perspective of the system environment, using the implemented design tool.

Chapter 7

A Case Study

Contents

7.1 Outline

This chapter presents a case study, to prove the feasibility of the proposed approach. Here, we will show how the system designer could use the proposed views to define the architecture design of an EIS. Furthermore, the evaluation process will be described step-by-step and the potential re-adjustments are discussed.

7.2 Description

In the following we discuss the case of renovating a legacy information system supporting a large-scale public organization. The organization supports more than 350 interconnected regional offices and its main purpose is to provide services to the public. Regional offices are technologically supported by a central IT Center responsible for IT diffusion and management. More than 15.000 employees work in the organization having on-line access to the legacy system, while there are more than 300 different services provided to the public. Regional offices are divided into three categories according to their size, structure and personnel (large, medium and small). Each category is treaded differently in terms of network infrastructure requirements. All of them have the same structure consisting of seven different departments reflecting independent operation, while all departments provide services to the citizens.

Existing system architecture is based on client-server model. All application logic is programmed within the client platform, while data are distributed in local database servers located in each regional office. A central database is supported in the IT Center for data synchronization and lookup purposes. Client programs access the local database to store data, while they access the central database mostly for lookup purposes. Local data are asynchronously replicated in the central database using a transaction management system (TMS). The IT Center and all regional offices participate in a private TCP/IP network to facilitate efficient data replication.

To enhance the level of service provided by the organization, over the last decade an e-government portal was established. The main target of the portal is to provide easy access to citizens twenty four hours per day, seven days per week and to minimize the need for citizen's presence in regional offices. The portal facilitates on-line transactional services and ensures on-line access to the databases of the legacy information system, serving almost one third of requests processed by the legacy system on a daily basis.

Since hardware supporting the legacy system was obsolete, the IT Center obtained the necessary funds to replace it. Though, since almost one third of the citizens request are serviced through the portal, it was decided to explore the renovation of the legacy information system by adopting modern technological trends, such server-based computing and thin clients to minimize maintenance cost. Hardware consolidation in the IT Center was considered instead of supporting local servers in regional offices, as well as changes in the database architecture by supporting one central database to avoid synchronization. Legacy system architecture modification should be considered without any changes to existing application code. The model-based EIS Architecture design approach presented in Chapter 4 was applied to explore alternative architectures and their implications to the network infrastructure. One of the main objectives of legacy system architecture re-design was to enhance application performance without rewriting the applications themselves.

One of the main objectives of legacy system architecture re-design was to enhance ap-

plication performance, without major rewriting of the applications themselves. Alternative software architectures and their implications to hardware/network infrastructure were evaluated. Since performance plays a significant role, the application of the EIS profile explored related design decisions and evaluated them. The scenario to be explored is to support the existing distributed database architecture and try to consolidate the hardware in order to improve the overall performance. A set of different architectural designs were proposed forming the corresponding evaluation scenarios. To measure the overall performance, specific non-functional requirements, such as response times were defined.

7.3 Challenges

To measure the performance of a system, specific quantitative parameters are required. Our first challenge was to define these parameters. The identification of the requirements for thr software components was not a straightforward procedure, since NFRs were not recorded, indicating the lack of Enterprise Architecture perception in the organization.

Another challenging issue was the fact that neither the software maintenance or administration personnel were able to provide accurate NFR information regarding response time or other performance-related requirements for the software and hardware components.

To overcome these challenges, the response time requirements were finally defined by software designers of the organization, while the QoS requirements were obtained via manual auditing application functionally in the current version of the system. To obtain the required resources (processing, storage and networking) of a software component, the Constructive Cost Model (COCOMO) II [128], as a procedural software cost estimation model, was utilized. Using Functional Point Analysis technique it is possible to quantify the functions contained within software in terms that are meaningful to the software users. Function points are a standard unit of measure that represent the functional size of a software application. Using these methods we were able to calculate the function points of each software component (service) and transform them to countable processing instructions.

As a result, the accurate definition of service-QoS was essential for the effective exploration of application performance based on alternative architecture scenarios.

7.4 Design Mode

7.4.1 Functional View

Functional View describes software architecture of legacy system. Seven independent applications are supported, each one perceived as a different module, while a total of 300 on-line services are provided by them. According to legacy application design, each application reflects the operation of a specific department of regional office. Since application

functionality is well-known, the identification of software architecture and performance requirements was perceived as a trivial task. To obtain this information, the system designer had to communicate with application maintenance personnel in the corresponding department of the IT Center. RUP methodology ([103], [75]) was used for software development, thus application description models were developed within Rational Rose platform. Application description (e.g. applications, modules and services) as well as data structures were manually extracted from corresponding Rational Rose [129] files. Though the process was not automated, the provision of Functional View meta-model, enabled the system architect to easily obtain the necessary information. Unfortunately, the identification of service performance requirements was not a straightforward procedure, since software maintenance personnel was not able to accurate provide either response time or service QoS information. Response time requirements were finally defined by system architects, while service QoS information were obtain after monitoring application functionally during working hours by system administration personnel in the current version of the system. Service QoS requirement accurate definition was essential for the effective exploration of application performance based on alternative architecture scenarios.

A snapshot of the *Functional* view corresponding to a distributed architecture scenario of the system under consideration is depicted in Figure 7.1. All application logic is programmed within clients running on users' workstations, while data are distributed in local database servers located in each regional office.

A central database is supported in the IT Center for data synchronization and lookup purposes. Client programs access the local database to store data, which are asynchronously replicated in the central database using a transaction management system (TMS). The IT Center and all regional offices participate in a private TCP/IP network to facilitate efficient data replication.

As an example, an excerpt of *Functional* view focusing on manage citizens application is depicted in Figure 7.1. This application, named *registry application*, is composed of four services:

- add taxation registry
- update taxation entity
- deactivate taxation entity
- check data

The application is used by two different user roles, the *registry staff* and the *registry manager*. *Registry staff* perform all these operations (services) while *registry manager* cannot update registry data. These interactions are shown as special relations between role and service, called *initiate*, indicating the fact that a user perform a specific operation in the system.

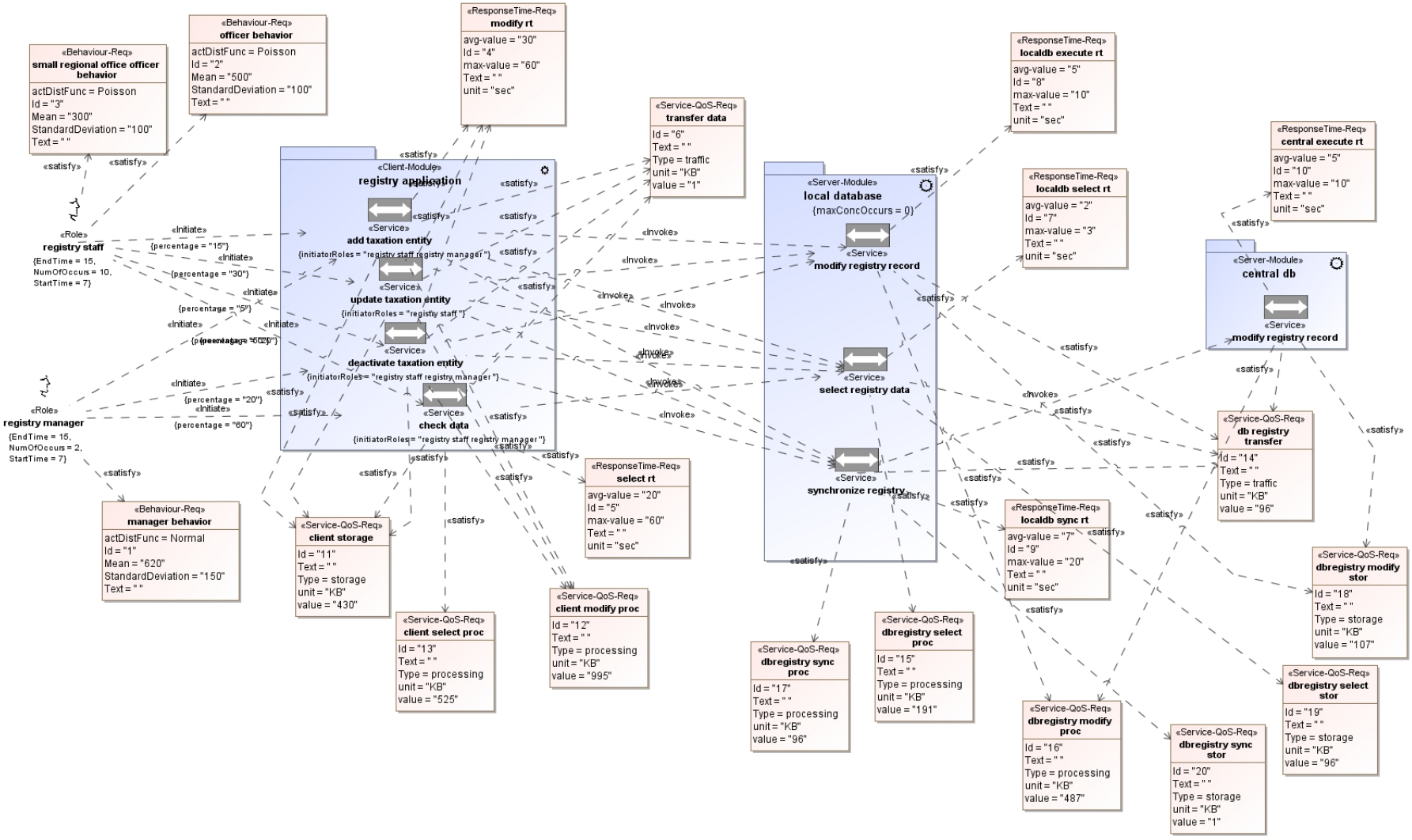


Figure 7.1: Functional View

This kind of relation contains additional data: a percentage variable. To explain the role of *percentage*, let us consider a role in his everyday interaction with the system and the application performing some operations. The percentage indicates how often the role uses these specific services. We consider that a registry staff calls operation *add taxation entity* by 15% , while he/she makes updates to existing entities by 30% , deactivates entities by 5% and the rest 50% is about checking specific records of the entities (e.g., readonly database operations).

Each service (belonging to a client-module) that a user initiates, in order to be executed requires the invocation of other services, either belonging to this regional office or to a remote datacenter, constituting a distributed environment. For example, the service *update taxation entity* invokes *modify registry record*, *select registry data* and *synchronize registry* of a local database server module. Afterwards, the *synchronize registry* invokes *modify registry record* of a remote server module belonging to the datacenter of the organization.

The software designer may use a *behavior requirements* to model role behavior. Here, the registry staff role may satisfy two different behavior requirements corresponding to normal and heavy workload. Later, on Evaluation view, two different evaluation scenarios are enabled for execution, each of them according to these behavior aspects.

Each service could be related to specific requirements: e.g., one kind is *service-QoS* requirement, storing the required resources for execution. This is roughly an estimation about the processing power, storage and networking resources that this service requires in order to execute. Consequently, there are three QoS requirements for each service. For each of them, a max-value and an average value is estimated. These values were acquired using the COCOMO II [128] methodology. According to our proposal, the relation between a service and a *service-QoS* requirement is *satisfy*. For a specific module, all *service-QoS* requirements constitute a *module-QoS* requirement, which is a derived requirement, that gathers the required resources of all services belonging in this module.

Another requirement related to services is the *responseTime* requirement. This defines the maximum execution time that a user could wait for this operation to accomplish. In the case that this service would invoke other services, this particular time includes the responses of all the invoked services. During the evaluation process, the response time of a service is estimated and the estimated value should *verify* this predefined time. From the system designer perspective, the following steps are required to work on *Functional* view:

- User roles creation. A role groups users interacting with the same applications and have similar behavior.
- application access points definition. Usually this is the front-end environment of the applications, providing specific operations to users.
- Definition of the identical operations that are performed for each application. These operations, called services, are grouped into modules.
- Service invocations architecture definition; for each service the calling services should be defined, in order to provide the complete software architecture of the application

under consideration.

In parallel, a set of validation rules are performed to ensure that the model is in accordance with the defined EIS profile, as depicted in Table 5.1. Consider the percentage of all services initiations from a role summing up to 100%. When a violation in a validation rule occurs, the designer is notified.

Figure 7.2 presents the application of validation rules to model. A specific role produces several violations: In our case, one of them, the summing up to 100%, for the percentage attribute of services initiations, of this role, is not valid. The design environment identifies possible problems and suggests alternatives(Figure 7.3).

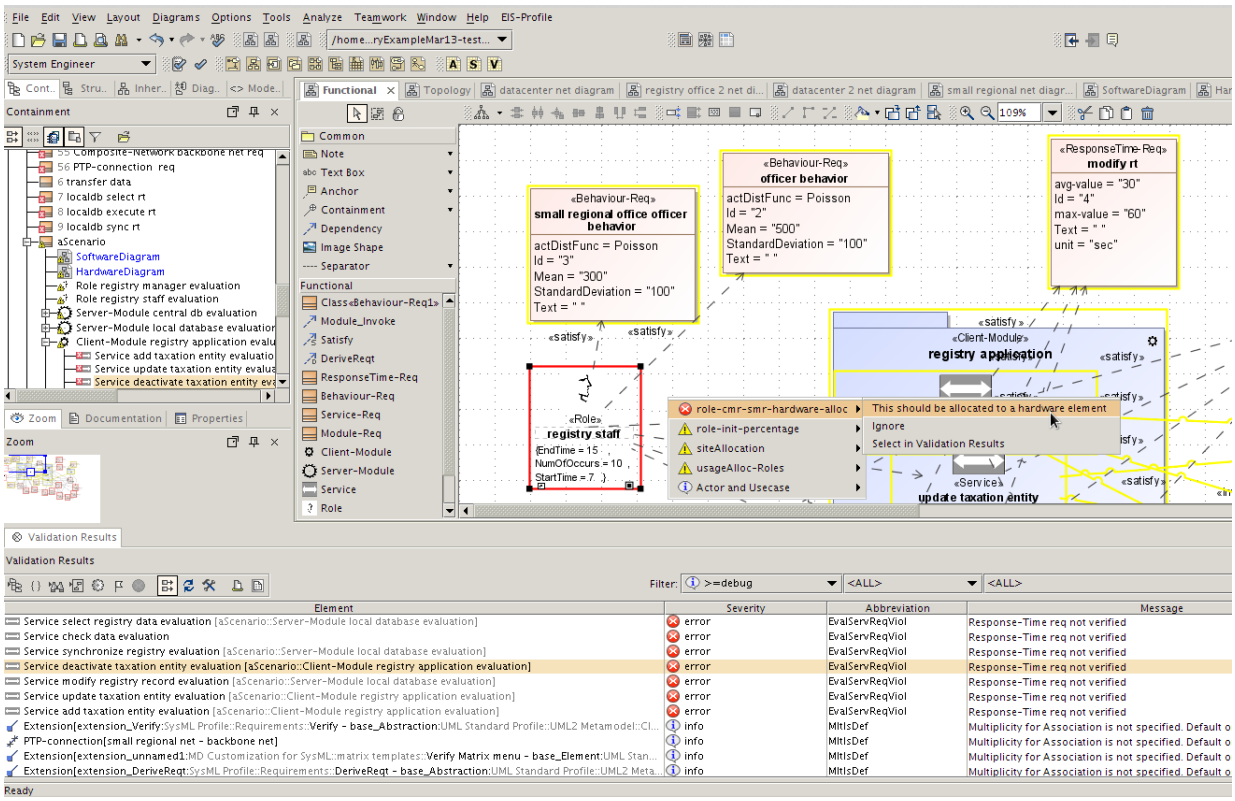


Figure 7.2: Functional view: Validation rules applied

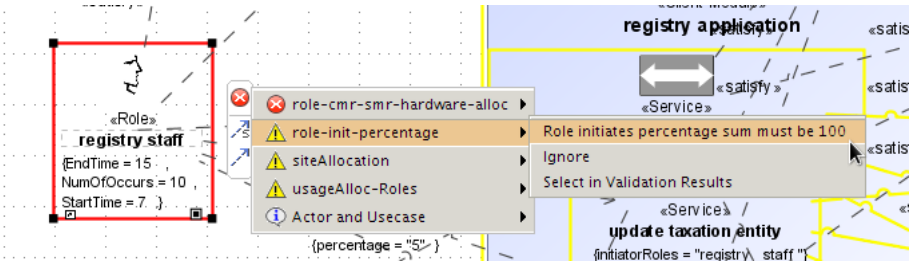


Figure 7.3: Functional view: Validation handling

7.4.2 Topology View

Topology view is exploited to specify the system access points and define the software replicas. Three model elements are participating in this view:

- Roles (derived from *Functional* view) are automatically imported.
- Client and Server modules (according to *Functional* view again) are also auto-imported.
- Sites (system access points). There are two kind of sites, atomic and composite [1]. Sites are defined by system designer to gather geographically or conceptually the roles and modules.

As stated, roles, client and server modules are auto-derived from *Functional* view. To the designer remains he allocation of roles and modules to sites and then, via *Network Infrastructure* view, to workstations and servers. Since EIS are complex and distributed information systems, there are many replica software components. For that reason, the system designer should make replicas of he server modules and allocate them to sites, e.g. local or remote datacenters. This is achieve by right-clicking on a server module, and the new created model element is related with *replica of* relationship to source element.

Next step deals with the allocation of roles to sites. This is accomplished by defining a *usage allocation* relationship between the role and the site. When roles are allocated to sites, then an automation could be applied to allocate the initiating client-modules to the same site from the application menu. The logic is simple; when a role is allocated to a site, the client modules that contain services initiated by this role are auto-allocated with *software allocation* relationship to the same site. In practice, a client-module replica is created and allocated to the same site. This has to do about our ability to define many users sharing a specific role (defined with the attribute NumberOfOccurs of role entity). This step has been implemented as part of the EIS plugin, facilitating the system designer and ensuring model consistency, according to EIS profile. Thus, we cannot have a role allocated to a site without the called services of this role to do not belong to the same site.

There are some additionally constraints that our model should obey to. *Usage allocations* and *software allocations* should be applied to atomic sites, which are the leafs of the sites hierarchy. Atomic sites are contained in composite sites using the *containement* relationship of UML. Validation rules ensure the compliance of our system model according to constraints defined in Table 5.2. For example, a validation rule checks if there is a role defined in *Functional* view that has not been allocated to a site in *Topology* view. The same applies to client and software modules; all modules have replicas allocated to atomic sites.

As shown earlier, *service-QoS* requirements have been defined to services in *Functional* view. Two of them (processing and storage) are estimated for the modules that the services belong to. The other one, traffic *service-QoS* is meaningless to be defined in relation to a module. What makes sense is to de defined in the case of communicating services between modules. For that reason, *traffic service-QoS* are automatically created and their values are estimated according to the Algorithm 3 presented in subsection 5.3.4. The *module-QoS* derived requirements are related to the *Module Replica Invoke* relations between the communicating module replicas.

[1] comprised of other sites to formulate a site hierarchy

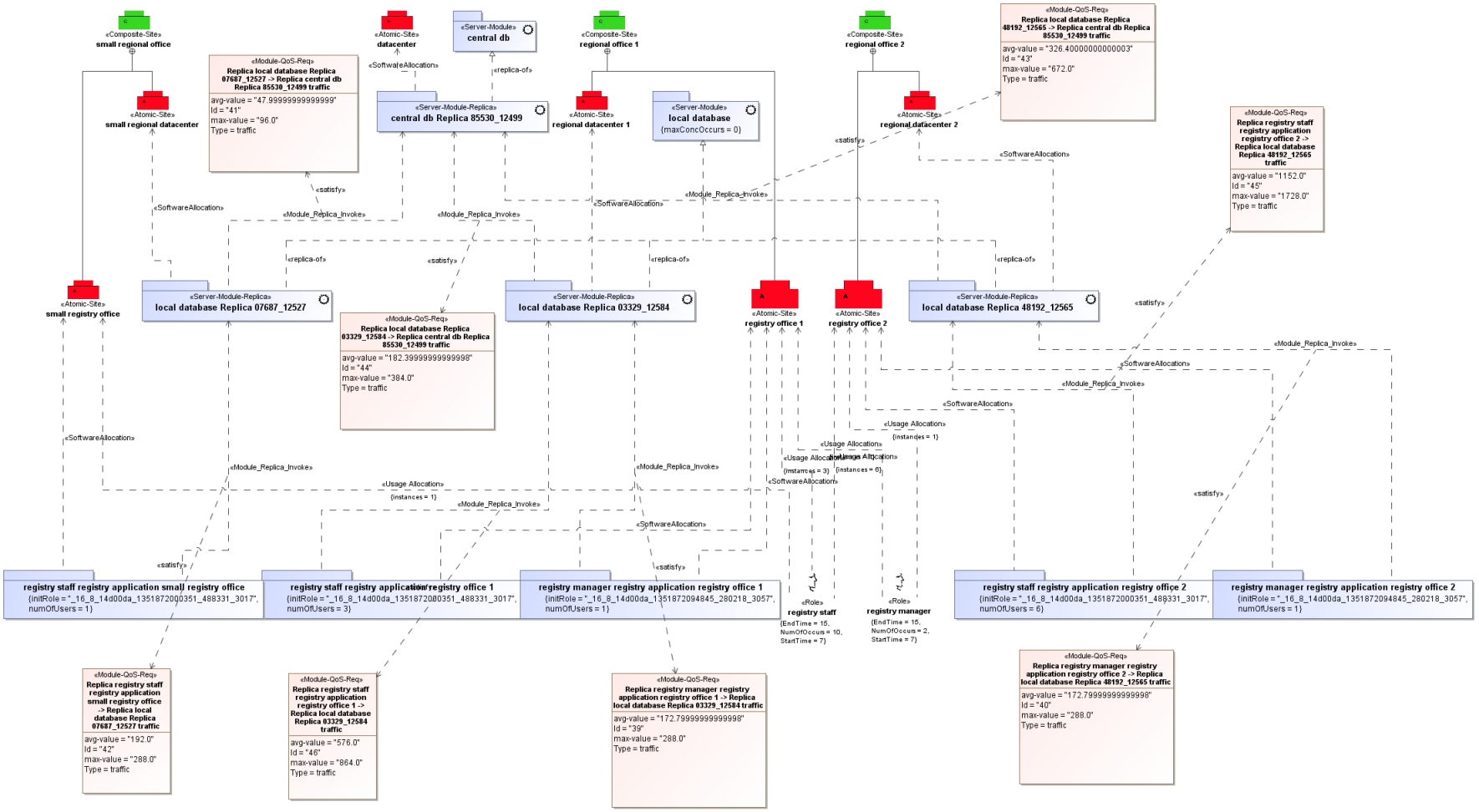


Figure 7.4: Topology View

In *Topology* view diagram (Figure 7.4) and in accordance with *Functional* view diagram, the *registry staff* role is allocated to three different sites: *small regional office*, *registry office 1* and *registry office 2* atomic sites. To these sites are also allocated module replicas [1] of the registry application (namely *registry staff registry application small registry office*, *registry manage registry application registry office 1* and *registry manage registry application registry office 2*). Ten discrete role instances are allocated to sites with the following occurrences: one of them is allocated to *small regional office*, four to *regional office 1* and the remaining six to *regional office 2*.

7.4.3 Network Infrastructure View

Network Infrastructure view is utilized to design the network architecture of the EIS. Each network is either an atomic or composite [2] network. For atomic networks, the specific hardware elements that belonging to that network are presented in a special atomic network diagram. In that way, Network Infrastructure View is composed of:

- an overall network diagram where network hierarchy and interconnections to other external networks are presented
- a complementary diagram associated with each atomic network presenting the roles, the software components and the hardware elements connected to that specific network. Specifically:
  - Roles and software components that have been defined in *Functional* and *Topology* views, and have been allocated to sites.
  - Hardware components that are defined in atomic networks.
  - Roles and software allocations that are also take place in atomic network diagrams.

To help the designer with allocations, in network infrastructure main diagram, sites are auto imported from the *Topology* view, and what remains to the designer is to allocate them to networks. Validation rules ensure the consistence with the EIS profile: e.g., every site should be allocated to a coresponding network [3] (see Table 5.3 for the set of the defined constraints).

Figure 7.5 presents the network architecture of the EIS under consideration. In most cases sites are allocated to networks with *structural allocation* relationships in a one-to-one fashion. Depending on the *load* requirements (see subsection 5.3.1) the system designer may decide to allocate multiple sites to a network or vice-versa. Certainly, if a composite network is allocated to a network, all the contained sites should also be allocated to the same network's or subnetworks' hierarchy. Validation rules ensure this compliance.

[1] the names are auto-produced by the combination of the role name, the module name and the site name
[2] if it interconnects other networks
[3] an atomic site has to be allocated to an atomic network

Requirements are also presented in this diagram. For each network a *load* requirement is estimated as the result of the demanding networking resources of all software components that are allocated to this network and is produced by specific roles' behavior. The way these requirements are derived is described in the Algorithm 5.3.4 of the subsection 5.3.4. *Load* requirements provides a quantity indicator of the networking load imposed to networks. System designer can reclaim site allocation policy to increase/reduce load to networks.

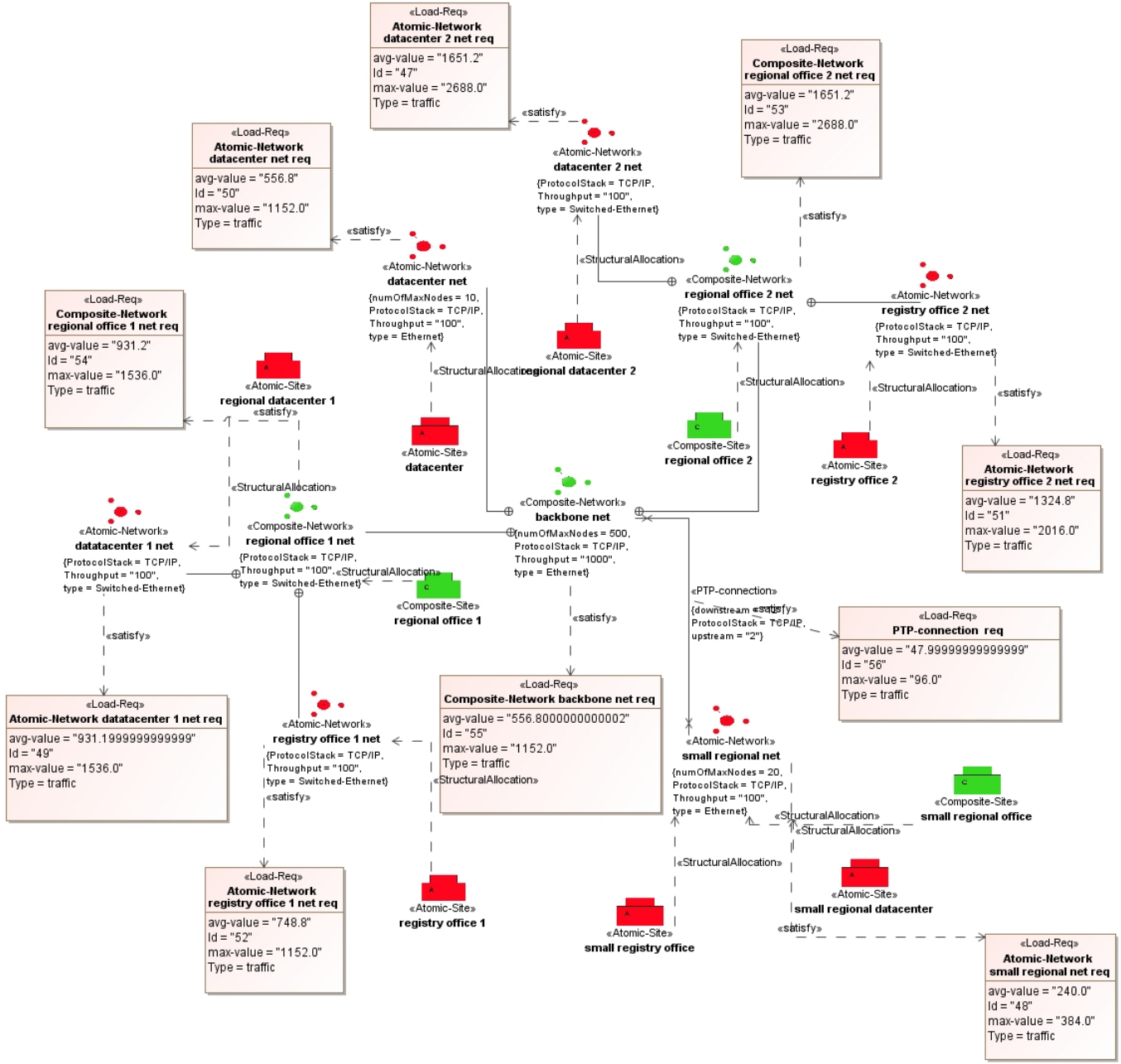


Figure 7.5: Network Infrastructure view

Figure 7.6 presents such an atomic network diagram for the *regional office 2* network, as presented at Figure 7.5. Roles and software components are auto imported from *Functional* and *Topology* views. The designer should only add the hardware nodes where the roles and the software will be allocated to. For each of them, a number of instances could be defined, describing many users allocated to workstations with similar hardware and software capabilities. For each node, a number of attributes could be defined (see Table 5.3), categorized as follows:

- Quantity
- Operating System
- Memory
- Processing Unit
  - Processing Power
  - Cores
  - Number of processors
- Storage Unit
  - Number of disks
  - Storage speed
  - Capacity (per disk)

We consider that in *regional office 2* network, there are 2 registry managers ir front of a workstation and 10 registry staff, eh one having one workstation For each of them the working hours are providing, as defined in *Functional* view.

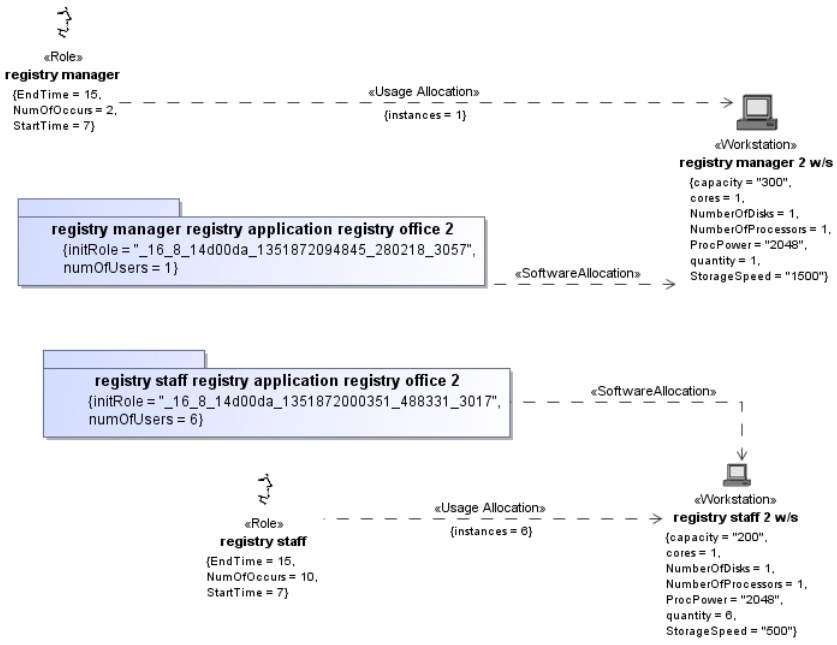


Figure 7.6: Network Infrastructure View, Atomic Network

7.4.4 NFR View

NFR view is a complementary view where all the requirements from other views are gathered. Figure 7.7 presents an excerpt of the NFR view, where the requirements defined in all design views are illustrated. Requirement's expert or a system designer are able to see or define the requirements relationships and derivations. Starting from generic, qualitative text-based requirements, we could end up to specific quantitative requirements, in order to be used as simulation input or evaluation conditions. From the perspective of the system designer, this view is not mandatory for the completion of his work. It is more useful in the early stage of system analysis.

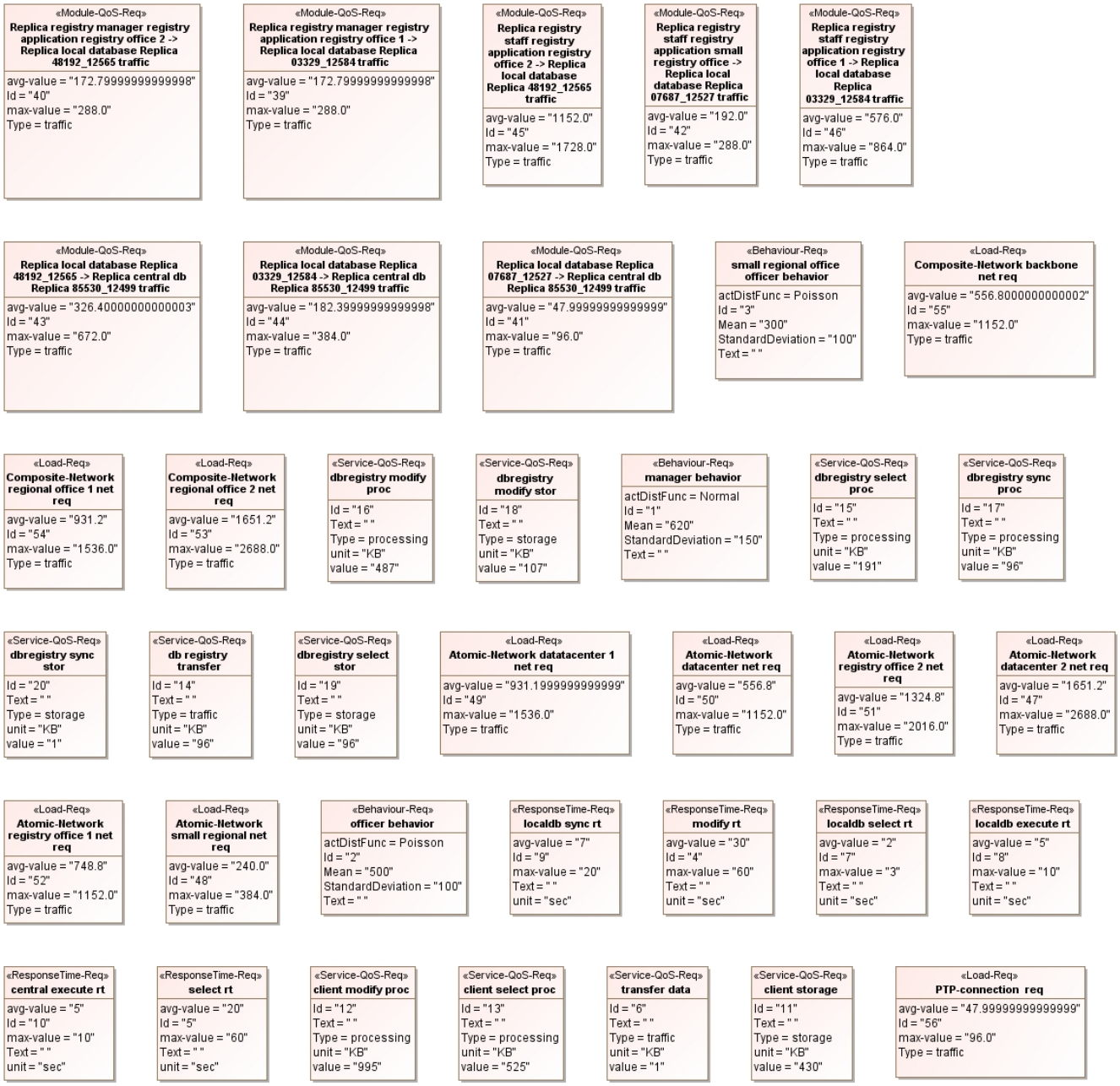


Figure 7.7: Non Functional Requirements

7.5 Producing Evaluation View and Inflating Simulation Parameters

As the design phase is performed and after the validation rules have been applied to system model in order to check its consistency and completeness, we could proceed to evaluation phase. The first step is to create an evaluation diagram and an a corresponding scenario. An evaluation scenario is comprised of two specific diagrams:

- a software architecture diagram: a replica of *Functional* view where only evaluation specific elements are presented
- a hardware architecture diagram: a multi-level diagram in accordance with *Network Infrastructure* view, where also only evaluation specific elements participating.

The reason that we have different evaluation diagrams is that we want to record the different scenarios under consideration. Evaluating a snapshot of the architecture enables the ability to make comparisons and peruse the influence of changes in the overall performance of the system.

During the evaluation process, requirements act as behavior descriptors or performance ones (see section 5.3 and Figures 6.2 and 6.1). Behavior requirements are used as simulation input while among performance ones are *responseTime*, *availability* and *utilization* requirements.

7.5.1 Evaluation scenario

As already stated, an evaluation scenario is a snapshot of the system. Evaluation scenarios comprise of evaluation entities used to evaluate design entities and verify corresponding requirements (Figures 4.10 and 6.3). It is important to notice that evaluation scenarios are auto-created from the design phase and this process does not require human interaction. Of course, if the design phase diagrams are inconsistent or not completed, notification messages inform the designer about the imposed problem.

Software Architecture Diagram

The entities participating in a software architecture diagram correspond to *Functional* view entities and are used to:

- define the behavior of the software components during the evaluation of the proposed EIS architecture design
- verify the appropriate requirements of software components (e.g. services response-time)

As we can see in Figure 7.8, roles and modules are presented. For each role, there are *Eval-Initiate* relationships to each module, so many times as the number of the calling services. Notice that each element has an attribute named *simid*, namely a unique identifier used in order to handle the corresponding simulation results.

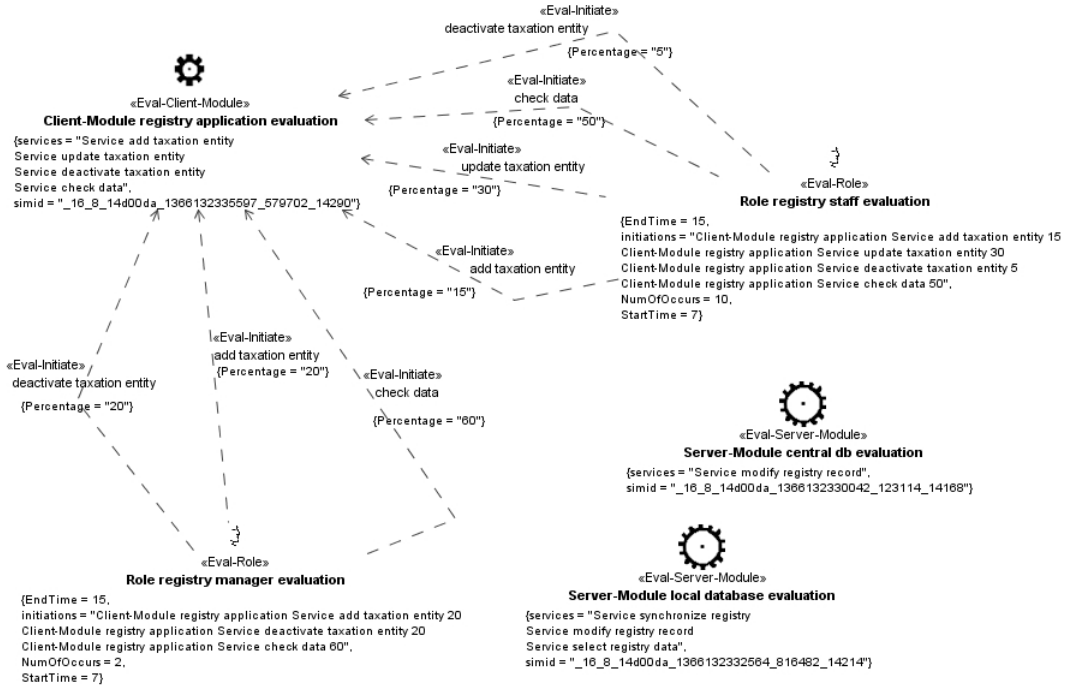


Figure 7.8: Software Architecture Evaluation Diagram

Hardware Architecture Diagram

Hardware architecture diagram entities correspond to Topology (module replicas) and Network Infrastructure view entities, and are used to:

- to initialize a corresponding simulation model instance
- to evaluate the design entity and
- verify the corresponding requirements.

Hardware architecture diagram (Figure 7.9) is like a network infrastructure diagram where no *load* requirements are presented. *Load* requirements are used in the early stages of the design, to help the network architect allocate sites to networks. Each network has input and output attributes, where input attributes (Figure 7.10) are used as simulation parameters (passed as initialization parameters to a constructor of a network class). Output parameters will be filled with values when the simulation results will be incorporated to system model.

Network hierarchy is built using include relations between networks. For each atomic network a specific diagram presents roles, software components and nodes. Figure 7.11 is produced from the corresponding atomic network diagram from the design phase, presented in Figure 7.6. The obvious difference is that in the evaluation phase diagram, behavior requirements are presented. These requirements are derived from *Functional* view (see Figure 7.1) and help the designer to select specific behavior for the roles (in case there are many behavior requirements specified).

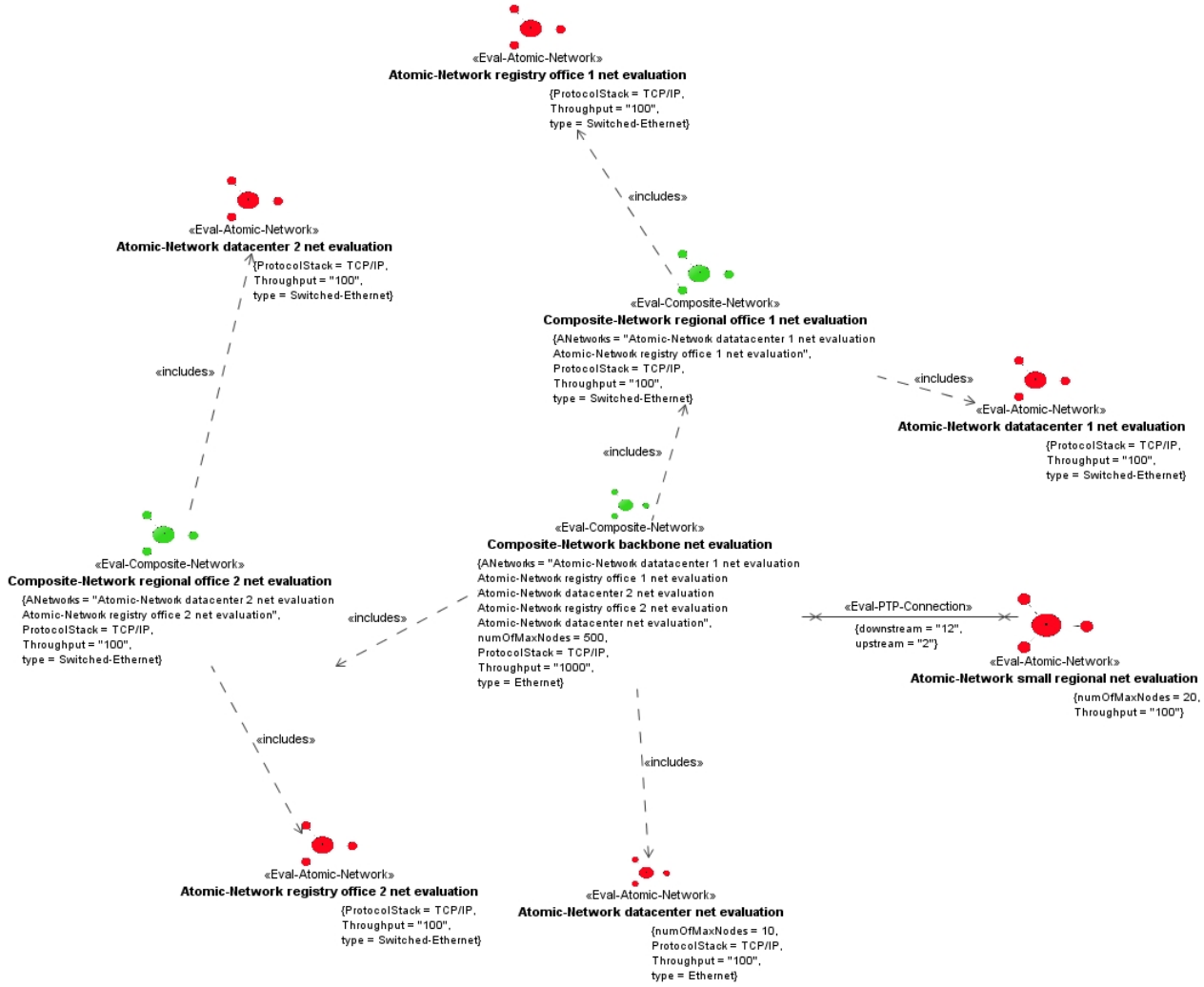


Figure 7.9: Hardware Architecture Evaluation Diagram

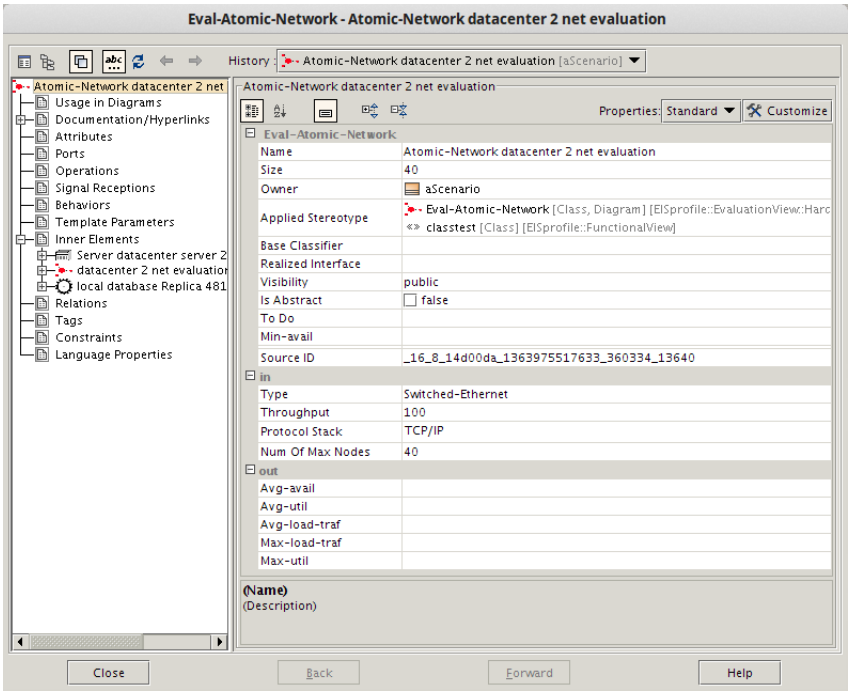


Figure 7.10: Hardware Architecture Evaluation Diagram, Atomic Network properties

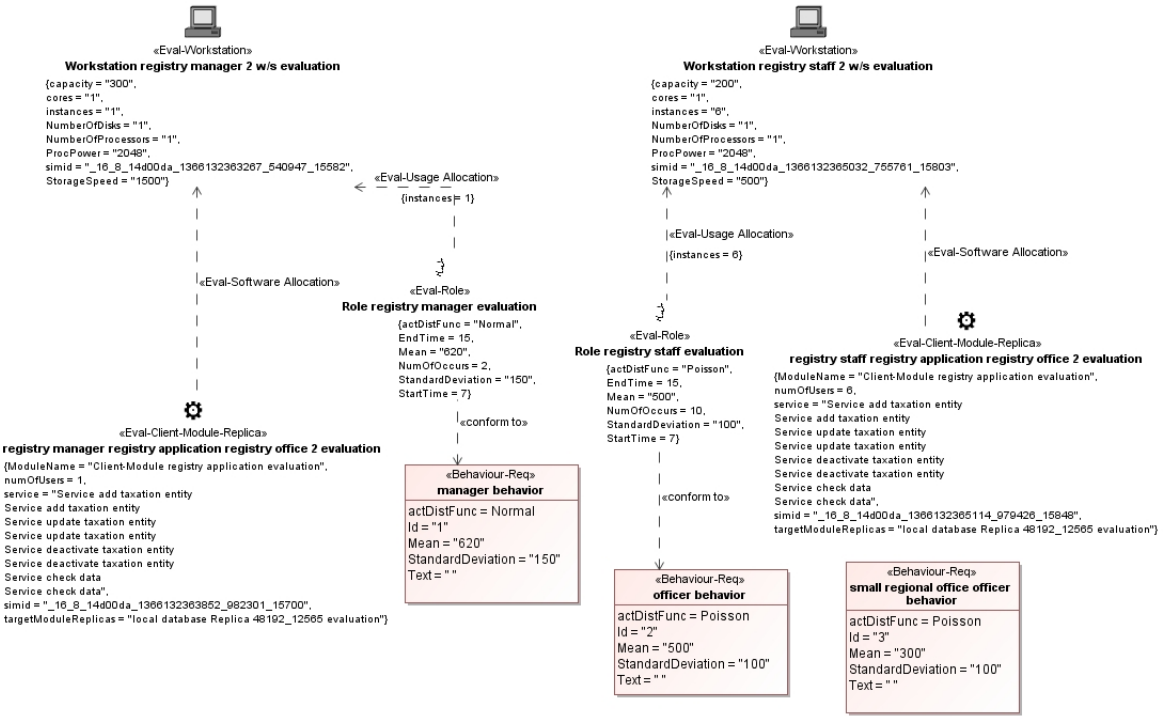


Figure 7.11: Hardware Architecture Evaluation Diagram, Atomic Network

7.6 Transformation to simulation code

Simulation model should represent a concrete system, which is described in many views. For that reason model transformation is inevitable. The transformation formula has been described in Figure 6.5. In an EIS model, elements are presented in different views, which are described in an evaluation scenario. Here, the selected simulation framework is DEVS. In a DEVS model there is a strict model hierarchy, due to the necessity of executable simulation code. To transform the system model to a corresponding DEVS simulation model, the following steps are required:

i. export the system model in XML format, using XMI specification [130]. Magicdraw modeling tool supports the export of a model in XMI format (Eclipse UML2 v2.x XMI file).

ii. import the XMI file in Medini QVT [127] tool, and apply the QVT transformation (the qvt file in repository [122]). The resulting model is also in XML format corresponding to a DEVS model.

iii. apply an XSLT transformation to obtain DEVSJava executable code from the xported XML file.

In [25] the interested reader can find details about the model transformations in order to produce the simulation code.

7.7 Simulation execution and results incorporation

DEVSJava simulation environment has to be extended with the appropriate simulation library components due to the lack of the description of the behavior (explained in section 6.4.1) . The source code of library components is presented in [122]. Simulation time and the number of simulation runs should be defined, in order to have reliable results. After the simulation execution, an XML file is produced containing simulation results, according to Figure 6.7. Listing 7.1 presents an excerpt of these results.

Listing 7.1: Simulation results

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<results>
    ...
    <result
        stereotype="Eval-Atomic-Network::Eval-Service-Replica"
        name="synchronize registry"
        id="_16_8_14d00da_1366132358008_45445_15160"
        count="1">
        <value
            value="0.00008364964640774856"
            name="avg-ResponseTime"/>
        <value
            value="0.0003955021723850222770124673843383789062 5"
            name="max-ResponseTime"/>
    </result>
    <result
```

```
        stereotype="Eval-Atomic-Network::Eval-Service-Replica"
        name="synchronize registry"
        id="_16_8_14d00da_1366132358005_661866_15146"
        count="1">
        <value
            value="-0.00000860113645457735"
            name="avg-ResponseTime"/>
        <value
            value="0.000577387292614730540663003921508789062 50"
            name="max-ResponseTime"/>
    </result>
    ...
</results>
```

The incorporation of the simulation results is a simple task, since it has been implemented in EIS plugin. Figure 7.12 presents how this could be achieved.

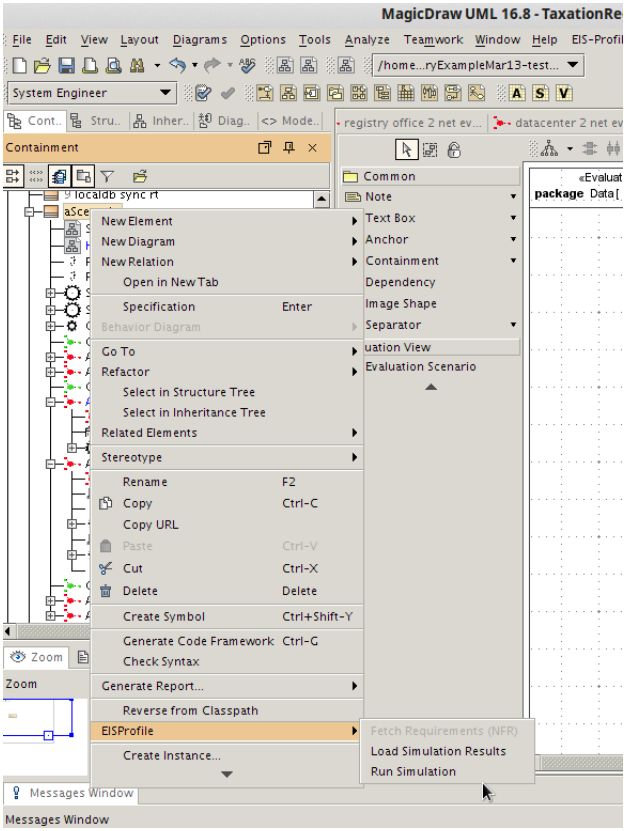


Figure 7.12: Importing Simulation results

7.8 Verifying Requirements

Having the results incorporated to our system model, the *out* tagged values of all evaluation entities (e.g. out values of Figure 7.10) are filled. Afterwards, a validation rule that checks the *out* values with the corresponding requirements values is applied. Figure 7.13 presents the case where an atomic network fails to verify a *load* requirement: the requirement has an average value of 2.7Mbps with 0.2 deviation and the simulation reported an average load of 3.1Mbps. Since this value does not belong to the interval $[2.5, 2.9]$, the corresponding validation rule fails and the designer is being notified with a red annotation (see Figure 7.13).

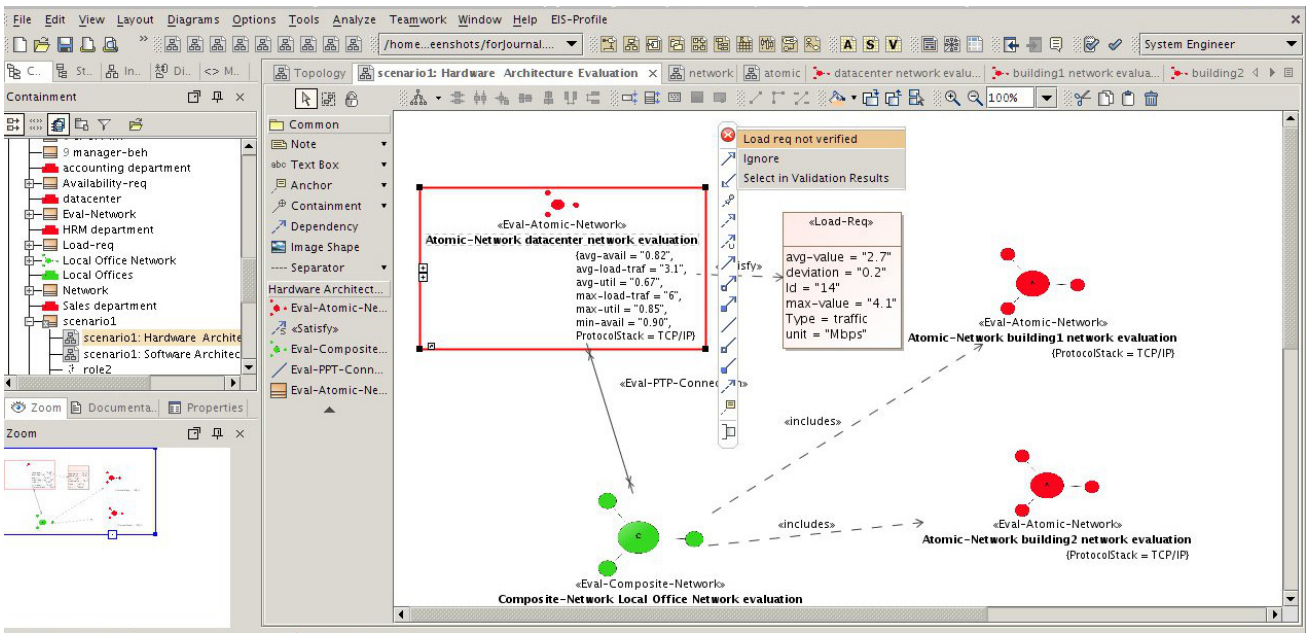


Figure 7.13: Verifying a load requirement

7.9 Re-design System Model

Depending on the nature of the non-verified NFR, system designer has to adapt the system model. This can be done with one of the following:

- in case of a *load* NFR, a solution is to alter the software to hardware allocations so as to increase/decrease the imposed load
- in case of a *response-time* NFR, either reducing the complexity of software architecture (e.g., decreasing software tiers) or providing hardware with additional capabilities (e.g. processing power or memory)
- in case of *availability* and *utilization* NFRs via improving either the hardware and/or the network.

7.10 Experience Obtained

System designers that tested the tool in the case study, appreciated the fact that all the information related to requirement verification was presented in a single view. The Load requirement derivation mechanism, enable the designer to accurately define network architecture. For example a suggestion was that more DB servers were required in Large Offices. They also found useful that all different experiment results were maintained, as evaluation scenarios, and could be used when making modification in architecture design. In fact, they ranked it as the most important feature of the proposed SysML extensions, since it enabled them to keep track of all redesign decisions and the reasons leading to them. Derived requirement computation was considered useful, but also a bit confusing for some of the experts

using the tool. Furthermore, most of them also suggested that the tool could propose alternatives on system architecture modifications to satisfy imposed requirements. Using the tool, it was estimated that software performance was improved almost by one third by the second scenario suggesting a Central Database Architecture, while the network architecture, interconnecting regional offices and the IT Center, remained the same. Thus, the communication cost was not much bigger.

The outcome of this experience proved that it was feasible to achieve desired performance adopting alternative system designs without any application re-writing. The proposed approach enabled the suggestion of alternative designs for local databases that result in improved performance. Initial service response time requirements were not verified, because of limited network bandwidth in the initial private network design. Requirements derivation enhanced the private network design. Load requirements were estimated and they revealed that in heavy load circumstances, network bandwidth requirements was not adequately defined.

7.11 Summary

This chapter presented a detailed case study concerning the application of the proposed approach as this is conceived and realized from the designer's perceptive. Design mode of the EIS plugin has been explained by exposing its facilities in a sequential manner. Each one of the provided views and diagrams has been depicted and explained to help the reader understand their capabilities. Next chapter presents a critical view about the contribution as well as the shortcomings of the proposed approach.

Chapter 8

Discussion

Contents

8.1 Overview

A SysML-based approach about model-based EIS architecture design was presented. The related background was reviewed in chapter 2. To reveal the research challenges that influenced the elaboration of this thesis, the related approaches were discussed in chapter 3. During this thesis, two main research areas were exploited: the *model-based systems engineering* and the *requirements engineering*. The literature review revealed the following shortcomings:

- regarding system architecture design, there are many stakeholders that each one wants his own perspective. A model-based approach, based on the literature, ensuring model consistency, supports this feature.
- requirements play a significant role in systems design and their verification is vital when talking about system performance. Regarding requirements verification, simulation has been identified as an appropriate technique for the estimation of system models' performance.
- simulation results should be incorporated within the original system model and a comparison against the predefined, performance-related, requirements should be performed within the SysML modeling environment.
- automated requirements verification within the SysML model could be enabled, once system performance estimation has been added in the model.

Afterwards, the contribution of this thesis was presented in three parts: the proposed approach described and documented by related work, the design phase of the proposed approach along with the definition and the handling of NFRs and finally the evaluation process through the NFR verification.

This thesis had two complementary objectives that concern an EIS: to provide auto-derived NFRs in order to act as indicators about the load of the software to hardware allocations and to propose an automation process of NFRs verification. To this end, focus was given on EIS architecture design. The same concepts described here, could be applied to other domains equally, with the corresponding customizations. In addition, EIS profile can be extended to include modern aspects of the web based applications development, such as cloud computing issues. In spite of the fact that the modules and services of *Functional* view resemble Software as a Service (SaaS), the concepts are more general and could be applied to Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) as well.

8.2 Contribution

As the application domain of this thesis was the EIS, the contribution could be considered as domain-specific. Though, during this research, some aspects were proposed that could be applied in general for the design of systems, and not necessarily only IS. Hence, as domain specific contribution, we consider the following:

- a domain specific profile for the design of EIS based on SysML and
- a design environment with:
  - enhanced model validation rules
  - auto-derived requirements
  - automated evaluation process

At the same time, the following aspects are domain agnostic and could be applied in systems design process:

- exploitation of requirements to define the behavior and the performance of the system and a way to be verified
- NFRs categorization focused on performance requirements
- an additional view dealing with the evaluation, enabling the existence of an evaluation scenario history

A key contribution of this thesis was the proposition of a way to measure the performance of a designed EIS, based on defined NFRs and constraints. As a result, system designer became evaluation-agnostic. After the definition of software and hardware architectures (at design phase views), the evaluation phase is auto-generated and a snapshot of the system model is able to be simulated, following specific guidelines. In any stage of the system design, validation rules ensure the conformance to the EIS metamodel.

As NFRs are critical in systems design process, this approach provides a systematic re-

view and a categorization of NFRs. Emphasis was given on performance issues and related requirements. Issues such as auto-derivation of requirements were resolved. To this end, quantitative characteristics have to be defined as attributes to NFRs and a derivation formula is applied. The integration of the derivation process inside the modeling environment supports the decision making process.

In early stages of software to hardware allocations, specific requirements provide the required information about the imposed load to hardware elements, helping the designer to define his allocation policy. To handle the user behavior, a new kind of requirement was introduced: behavioral requirements. These requirements can be associated with the user roles to describe their behavior, in terms of traffic generators, providing distributions and their parameters. Throughout this thesis, OMG standards were exploited in order to define a SysML profile and the corresponding QVT relations to transform system model to simulation model. These standards facilitated the interoperability between methods and tools of the proposed approach.

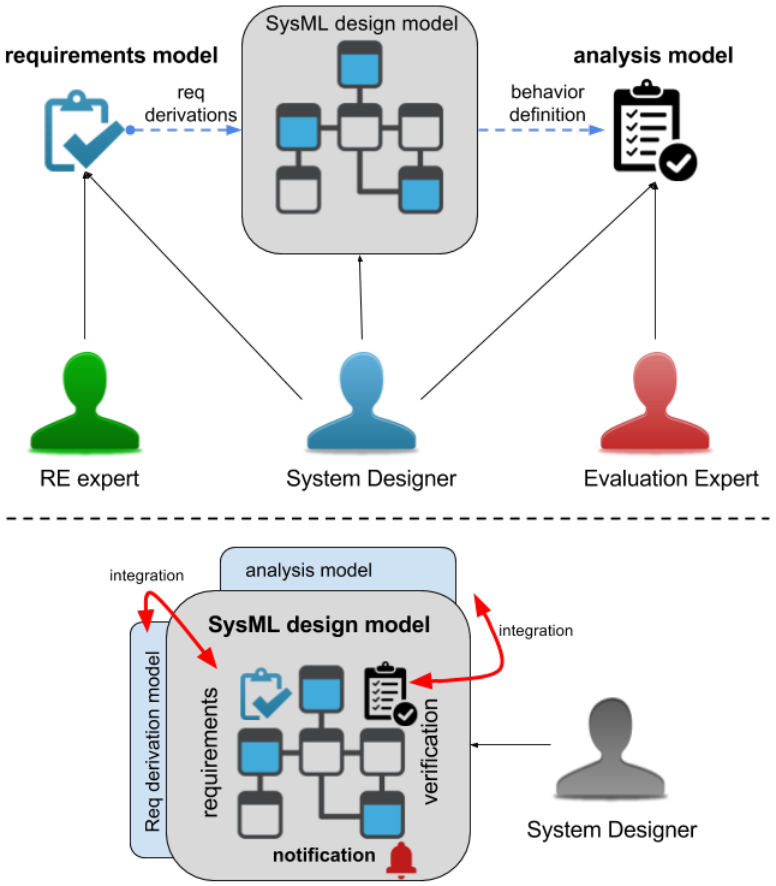


Figure 8.1: Contribution Overview

The overall contribution of this thesis is summarized in Figure 8.1. Until now, many stakeholders should use their own tools in order to design an information system. These tools could not effectively exchange data because each of them has its own data structures. An integrated design environment is provided so as the stakeholders participating in the design of an information system can be synchronized in order to effective design and evaluate the

actual information system. The knowledge of a requirements engineer expert, an analysis model expert and a system designer were utilized to constitute this environment, which is capable of executing model analysis to verify the imposed requirements, and this process is transparent to the system designer.

8.3 Limitations

As the intention of this thesis was to propose an approach for EIS architecture design, some bottlenecks arose, when someone tries to apply this approach to other domains. First of all, a domain expert is needed in order to implement simulation library components corresponding to the SysML model. Moreover, the selection of an appropriate evaluation method e.g., different simulation environments remains an open issue. As oftentimes stated, many stakeholders are involved in the design process of an EIS. Practically not all of them are properly communicating during this process.

Another key issue that was not addressed here, is the derivation of quantitative NFRs from generic described requirements using natural language. Maybe a query based system, domain specific and knowledge-based could contribute towards this direction.

Last but not least, the responsibility of the system designer to ensure requirements verification by alterations of the system model, remains open. A recommendation system would be useful to act as a self-healing system. Take as an example, an efficient recommendation system based on previous knowledge that could suggest auto-adjustments.

Chapter 9

Conclusions - Future Work

Contents

9.1 Conclusions

The process of EIS design is challenging, as it involves specialist from many domains, such as system designers, network architects, requirements engineers, simulation experts etc. All of them need a common canvas to be able to communicate. The proposed approach provides this particular canvas and the required tools in order to have a central model that could be used from anyone participating to this process and that is consistent and able to reflect the changes from one perspective to other perspectives.

The main objective of this research was the proposition of an MBSD approach for EIS architecture design using SysML. Motivated by the lack of efficient mechanisms for the verification of quantitative NFRs defined in SysML models, focus was given on the detailed representation of quantitative NFRs in SysML and their verification using quantitative methods. To this end, SysML was properly extended, while automated and efficient verification of SysML requirements via simulation was explored. Proposed concepts were applied in the information system domain, focusing on the design of EIS architectures, while performance requirements were focused.

The integrated framework, implemented to support the proposed approach, illuminates the role of models and standards towards solutions that enforce knowledge exchange and combined use of diverse proprietary tools. It aims to facilitate the system designer providing feedback about the performance of the system.

To explore the effectiveness, the proposed approach was applied to a complex case study. Chapter 7 presented a case study where an EIS was designed, evaluated and re-adjusted following the proposed approach. The transformation of system model to executable simula-

tion code was successful and the simulation results were incorporated to system model in order to verify the NFRs. The design environment informed the system designer about the non-verified NFRs, giving him the opportunity to make decisions to improve the performance of the system.

9.2 Future Work

This research put emphasis on the evaluation of performance requirements. Performance is a critical issue that a designer should examine when building a large scale EIS. Of course there are other issues such as *security*, *safety*, *usability*, *legal* and many others [46]. A forthcoming research direction could be to explore more types of NFRs, as there is a lot of interest in the literature [131, 132].

Moreover, research interest also exists in order to analyze existing requirements for regulatory compliance [133]. The cost of noncompliance is high, including fines, cost of court representation, government audits, and workforce training. Ensuring compliance to laws, regulations, and standards in a constantly changing business and compliance environment is one of the major challenges companies face today. As a result, compliance in EIS is an issue of major importance.

Additionally to the fact that the system designer is notified about the non-verified requirements, a complementary recommendation system, that would rely on existing and obtained knowledge, could make the evaluation process to act as a self-healing system. Utilizing the automation, on a non verified requirement occurrence, a set of running tasks/operations could be defined so as to support the concept of self-organizing networks in systems engineering. This presupposes that a knowledge base is available in order to decide the specific tasks that would make the requirement satisfiable again, ensuring that there are no other unverified requirements.

As a forthcoming research challenge, the application of the proposed approach on other system domains such as transportation and communications are already under investigation and partially implementation [134].

Moreover, the incorporation of techo-economic analysis during the design process it could be explored. Cost is one of the major factor that anyone should take into consideration when developing information systems. Into this context, a model-driven techno-economic targeting the estimation of economic parameters of cloud service deployment, which is able to assist decision support procedures for cloud users, cloud providers and cloud brokers. SysML could be adopted as a modeling language for describing cloud architectures as SoS, emphasizing cost properties. As an example, the Total Cost of Ownership (TCO) for cloud infrastructure and services could be explored [135].

Last but not least, due to the rise of cloud computing and the extensive use of web services, the proposed approach could fit the needs of any modern information system running

in the cloud and communicating with other software components via web services. A set of cloud computing specific non-functional requirements are necessary to support this customization.

Bibliography

1. J. F. Sowa and J. A. Zachman, “Extending and Formalizing the Framework for Information Systems Architecture,” *IBM Systems Journal*, vol. 31, no. 3, pp. 590–616, 1992.
2. IEEE, “IEEE System and Software Engineering - Architectural Description: Std 42010,” Tech. Rep., May 2009.
3. “MagicDraw UML.” [Online]. Available: http://www.magicdraw.com/
4. M. Nikolaidou, A. Tsadimas, N. Alexopoulou, and D. Anagnostopoulos, “Employing Zachman Enterprise Architecture Framework to systematically perform Model-Based System Engineering Activities,” in *HICSS-42*, 2009, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2009.189
5. A. Tsadimas, M. Nikolaidou, and D. 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*, ser. SAC ’12. New York, NY, USA: ACM, 2012, pp. 1057–1062. [Online]. Available: http://doi.acm.org/10.1145/2231936.2231941
6. M. Nikolaidou, A. Tsadimas, and D. Anagnostopoulos, “Model-based enterprise information system architecture design using SysML,” in *IEEE Systems Conference 2010*, April 2010.
7. A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, “Evaluating software architecture in a model-based approach for enterprise information system design,” in *SHARK ’10*. New York, USA: ACM, 2010, pp. 72–79.
8. ——, “Handling non-functional requirements in information system architecture design,” in *ICSEA ’09*, 2009, pp. 59–64.
9. A. Tsadimas, G.-D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, “Integrating simulation capabilities into sysml for enterprise information system design,” in *System of Systems Engineering (SOSE), 2014 9th International Conference on*. IEEE, 2014, pp. 272–277.
10. M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, “Simulating sysml models: Overview and challenges,” in *System of Systems Engineering Conference (SoSE), 2015 10th*. IEEE, 2015, pp. 328–333.
11. A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. IGI Global, 2013, ch. 8: Model-Based System Design Using SysML: The Role of the Evaluation Diagram, pp. 236–266.
12. A. Tsadimas, “Model-based enterprise information system architectural design with sysml,” in *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*. IEEE, 2015, pp. 492–497.
13. G.-D. Kapos, V. Dalakas, A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, “Model-based sys-

tem engineering using SysML: Deriving executable simulation models with QVT," in *SysCon*. IEEE International Systems Conference, 2014.

14. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "A consistent framework for enterprise information system engineering," in *EDOC*. IEEE Computer Society, 2006, pp. 492–496. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/EDOC.2006.6
15. M. Nikolaidou, A. Tsadimas, N. Alexopoulou, and D. Anagnostopoulos, "Employing zachman enterprise architecture framework to systematically perform model-based system engineering activities," in *HICSS*, 2009, pp. 1–10.
16. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Using UML to model distributed system architectures," in *CAINE*. ISCA, 2005, pp. 91–96.
17. M. Nikolaidou, A. Tsadimas, N. Alexopoulou, A. Dais, and D. Anagnostopoulos, "A UML profile utilizing enterprise information system configuration," in *ICECCS*. IEEE Computer Society, 2006, pp. 77–88. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICECCS.2006.48
18. INCOSE, *Systems Engineering Handbook*, version 3.2.2 ed. San Diego, CA, USA: International Council on Systems Engineering, 2012.
19. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Extending UML 2.0 to augment control over enterprise information system engineering process," in *ICSEA*. IEEE Computer Society, 2006, p. 10. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICSEA.2006.41
20. N. Alexopoulou, A. Tsadimas, M. Nikolaidou, A. Dais, and D. Anagnostopoulos, "Introducing a UML profile for distributed system configuration," in *ICEIS: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006*, 2006, pp. 542–545.
21. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Facilitating enterprise information system engineering through a UML 2.0 profile: A case study," *Information Resource Management Association (IRMA 2007), Vancouver, British Columbia, Canada*, 2007.
22. ——, "Accommodating EIS UML 2.0 profile using a standard UML modeling tool," in *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*. IEEE, 2007, pp. 26–26.
23. B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/336512.336523
24. J. A. Estefan, "Survey of model-based systems engineering (MBSE) methodologies," vol. 25, pp. 1–80, May 2008.
25. S. Kapos, Georgios-Dimitrios, "Model-oriented approach for automating sysml system models simulation," Ph.D. dissertation, Harokopio University of Athens, 70, El. Venizelou Str, Kallithea, 09 2016.
26. R. H. von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.
27. L. M. Jessup and J. S. Valacich, *Information systems today*. Prentice Hall Professional Technical Reference, 2002.
28. S. J. Kapurch, *NASA Systems Engineering Handbook*. DIANE Publishing, 2010.
29. Definition of information system engineering. [Online]. Available: http://www.accessscience.com/content/information-systems-engineering/343950
30. Definition of enterprise system. [Online]. Available: http://en.wikipedia.org/wiki/Enterprise_system

31. D. L. Olson and S. Kesharwani, *Enterprise information systems: contemporary trends and issues*. World Scientific, 2010.
32. J. Schekkerman, *How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework*. Trafford, 2003.
33. A. Reichwein and C. J. Paredis, "Overview of architecture frameworks and modeling languages for model-based systems engineering," in *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2011, pp. 1341–1349.
34. S. Leist and G. Zellner, "Evaluation of current architecture frameworks," in *SAC*, H. Haddad, Ed. ACM, 2006, pp. 1546–1553. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141635
35. "Institute For Enterprise Architecture Developments." [Online]. Available: http://www.enterprise-architecture.info/
36. M. Nikolaidou and N. Alexopoulou, "Enterprise Information System Engineering: A Model-Based Approach Based on the Zachman Framework," in *HICSS'08*. IEEE Computer Society, 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/HICSS.2008.148
37. I. P1471, "IEEE Recommended Practice for Architectural Description of Software-intensive Systems–-Std. 1471-2000," New York, NY, USA, 2000.
38. M. Jarke, P. Loucopoulos, K. Lyytinen, J. Mylopoulos, and W. N. Robinson, "The brave new world of design requirements," *Inf. Syst.*, vol. 36, no. 7, pp. 992–1008, 2011.
39. M. Fonoage, I. Cardei, and R. Shankar, "Mechanisms for requirements driven component selection and design automation," *Systems Journal, IEEE*, vol. 4, no. 3, pp. 396 –403, sept. 2010.
40. E. R. Byrne, "IEEE Standard 830: Recommended Practice for Software Requirements Specifications," 1998.
41. A. W. Wymore, *Model-Based Systems Engineering*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1993.
42. J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using non-functional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, pp. 483–497, 1992.
43. L. Zhu and I. Gorton, "UML profiles for design decisions and non-functional requirements," in *SHARK-ADI '07*. Washington, DC, USA: IEEE Computer Society, 2007, p. 8.
44. R. Jain, A. Chandrasekaran, G. Elias, and R. Cloutier, "Exploring the impact of systems architecture and systems requirements on systems integration complexity," *Systems Journal, IEEE*, vol. 2, no. 2, pp. 209 –223, june 2008.
45. M. H. Kacem, M. Jmaiel, A. H. Kacem, and K. Drira, "A UML-based approach for validation of software architecture descriptions," in *TEAA*, 2006, pp. 158–171.
46. M. Glinz, "On non-functional Requirements." 15th IEEE International Requirements Engineering Conference, 2007.
47. A. v. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *Fifth IEEE International Symposium on Requirements Engineering (RE'01)*, aug 2001, p. 249.
48. L. Balmelli, D. Brown, M. Cantor, and M. Mott, "Model-driven systems development," *IBM Systems Journal*, vol. 45, no. 3, pp. 569 –585, 2006.
49. O. M. G. Inc, "Systems Modeling Language (SYSML) Specification, Version 1.2," June 2010.
50. International Council on Systems Engineering. INCOSE. [Online]. Available: http://www.incose.org/
51. O. M. G. Inc, "UML Superstructure Specification, Version 2.1.2," November 2007.

52. B. Nolan, B. Brown, L. Balmelli, T. Bohn, and U. Wahli, *Model Driven Systems Development with Rational Products*, IBM Red Book, 2008.
53. ISO, "Information technology –- open distributed processing –- use of UML for ODP system specifications," october 2009. [Online]. Available: ISO/IECCD19793
54. S. Izukura, K. Yanoo, T. Osaki, H. Sakaki, D. Kimura, and J. Xiang, "Applying a model-based approach to IT systems development using SysML extension," in *MoDELS*, ser. Lecture Notes in Computer Science, vol. 6981. Springer, 2011, pp. 563–577. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24485-8
55. "INCOSE System Enineering Terms Glossary," INCOSE, October 1998. [Online]. Available: http://www.incose.org/ProductsPubs/techresourcecenter.aspx
56. P. Kruchten, *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.
57. P. Ralph and Y. Wand, "A proposal for a formal definition of the design concept," in *Design requirements engineering: A ten-year perspective*. Springer, 2009, pp. 103–136.
58. Object Management Group. OMG. [Online]. Available: http://www.omg.org/
59. OMG, "Model Driven Architecture. Version 1.0.1," June 2003. [Online]. Available: Availableonlineviahttp://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf
60. F. S. David, *Model driven architecture: applying MDA to enterprise computing*. Wiley publishing, Inc. USA, 2003.
61. *Object Constraint Language Specification, version 2.0*, Object Modeling Group, jun 2005. [Online]. Available: http://www.omg.org/technology/documents/formal/ocl.htm
62. OMG, "Meta object facility (MOF) 2.0 Query/View/Transformation specification," *Transformation*, no. April, pp. 1–230, 2008. [Online]. Available: http://www.omg.org/spec/QVT/1.0/PDF/
63. "Sysmlforum." [Online]. Available: http://www.sysmlforum.com/
64. O. M. G. Inc, *Systems Modeling Language (SYSML) Specification, Version 1.3*, Std., June 2012. [Online]. Available: http://www.omg.org/spec/SysML/1.3/PDF
65. "SysML-faq." [Online]. Available: http://sysmlforum.com/sysml-faq/
66. J. Siddiqi and M. C. Shekaran, "Requirements engineering: The emerging wisdom," *IEEE Software*, vol. 13, no. 2, pp. 15–19, 1996.
67. A. Terry Bahill and S. J. Henderson, "Requirements development, verification, and validation exhibited in famous failures," *Syst. Eng.*, vol. 8, no. 1, pp. 1–14, Mar. 2005. [Online]. Available: http://dx.doi.org/10.1002/sys.v8:1
68. A. Law, *Simulation modeling and analysis*, 4th ed., ser. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2006.
69. U. Herzog, "Formal methods for performance evaluation," in *Lectures on Formal Methods and PerformanceAnalysis*, ser. Lecture Notes in Computer Science, E. Brinksma, H. Hermanns, and J.-P. Katoen, Eds. Springer Berlin Heidelberg, 2001, vol. 2090, pp. 1–37. [Online]. Available: http://dx.doi.org/10.1007/3-540-44667-2_1
70. G. Fishman, *Discrete-event simulation: modeling, programming, and analysis*. Springer Science & Business Media, 2013.
71. B. P. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation*, 2nd ed. Academic Press, 2000.
72. C. Seo and B. Zeigler, "Devs namespace for interoperable devs/soa," in *Simulation Conference (WSC), Proceedings of the 2009 Winter*, Dec 2009, pp. 1311–1322.
73. G.-D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. IGI Global, 2013, ch. 10: An Integrated

Framework to Simulate SysML Models Using DEVS Simulators, pp. 305–332.

74. ——, “An integrated framework for automated simulation of SysML models using DEVS,” *Simulation*, vol. 90, no. 6, pp. 717–744, 2014.
75. M. Cantor, *Rational Unified Process for Systems Engineering, RUP SE Version 2.0, IBM Rational Software white paper*, IBM Corporation, May 2003.
76. ——, “Rational Unified Process for Systems Engineering Part II: System Architecture,” *The Rational Edge*, 2003.
77. E. Huang, R. Ramamurthy, and L. F. McGinnis, “System and simulation modeling using SysML,” in *WSC ’07: Proceedings of the 39th conference on Winter simulation*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 796–803.
78. O. Schonherr and O. 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, December 2009, pp. 1711–1718.
79. D. Kimura, T. Osaki, K. Yanoo, S. Izukura, H. Sakaki, and A. Kobayashi, “Evaluation of it systems considering characteristics as system of systems,” in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, june 2011, pp. 43 –48.
80. W. Schamai, P. Helle, P. Fritzson, and C. J. J. Paredis, “Virtual verification of system designs against system requirements,” in *Proceedings of the 2010 international conference on Models in software engineering*, ser. MODELS’10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 75–89. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008503.2008514
81. O. M. G. Inc, “UML profile for MARTE: Modeling and analysis of real-time embedded systems specification, version 1.0,” November 2009.
82. H. Espinoza, D. Cancila, B. Selic, and S. Gérard, “Challenges in combining SysML and MARTE for model-based design of embedded systems,” in *ECMDA-FA*, ser. Lecture Notes in Computer Science, vol. 5562. Springer, 2009, pp. 98–113. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02674-4
83. ITU, “User requirements notation URN - language definition,” ITU, ITU-T Reccomendation Z.151, Nov. 2008.
84. M. Bajaj, D. Zwemer, R. Peak, A. Phung, A. Scott, and M. Wilson, “Slim: collaborative model-based systems engineering workspace for next-generation complex systems,” in *Aerospace Conference, 2011 IEEE*, 2011, pp. 1–15.
85. D. Knorreck, L. Apvrille, and P. de Saqui-Sannes, “Tepe: A sysml language for time-constrained property modeling and formal verification,” *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 1, pp. 1–8, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1921532.1921556
86. I. Ober, S. Graf, and I. Ober, “Validating timed UML models by simulation and verification,” *International Journal on Software Tools for Technology Transfer*, vol. 8, no. 2, pp. 128–145, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10009-005-0205-x
87. OMG, *SysML-Modelica Transformation (SyM)*, Nov. 2012. [Online]. Available: http://www.omg.org/spec/SyM/1.0/PDF/
88. W. Schamai, “Modelica Modeling Language (ModelicaML): A UML Profile for Modelica,” Tech. Rep., 2009. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-20553
89. O. Batarseh and L. F. McGinnis, “System modeling in sysml and system analysis in arena,” in *Proceedings of the Winter Simulation Conference*, ser. WSC ’12. Winter Simulation Conference, 2012, pp. 258:1–258:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2429759.2430107
90. L. McGinnis, E. Huang, K. S. Kwon, and V. Ustun, “Ontologies and simulation: a practical ap-

proach," *Journal of Simulation*, vol. 5, no. 3, pp. 190–201, 2011.

91. R. Peak, R. Burkhart, S. Friedenthal, M. Wilson, M. Bajaj, and I. Kim, "Simulation-based design using SysML part 1: A parametrics primer," in *INCOSE Intl. Symposium*, San Diego, CA, USA, 2007, pp. 1–20.
92. L. McGinnis and V. Ustun, "A simple example of SysML-driven simulation," in *Winter Simulation Conference (WSC), Proceedings of the 2009*. IEEE, 2009, pp. 1703–1710.
93. R. Wang and C. Dagli, "An executable system architecture approach to discrete events system modeling using SysML in conjunction with colored petri nets," in *IEEE Systems Conference 2008*. Montreal: IEEE Computer Press, April 2008, pp. 1–8.
94. M. dos Santos Soares and J. L. M. Vrancken, "Model-driven user requirements specification using SysML," *JSW*, vol. 3, no. 6, pp. 57–68, 2008.
95. A. A. Kerzhner, J. M. Jobe, and C. J. J. Paredis, "A formal framework for capturing knowledge to transform structural models into analysis models," *J. Simulation*, vol. 5, no. 3, pp. 202–216, 2011.
96. W. Schamai, "Model-based verification of dynamic system behavior against requirements: Method, language, and tool," Ph.D. dissertation, Linköping University, SE-581 83 Linköping, Sweden, 10 2013.
97. M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Challenges in sysml model simulation," *Advances in Computer Science: an International Journal*, vol. 5, no. 4, pp. 49–56, 2016.
98. "INCOSE Handbook SE Process Model," INCOSE, September 2003. [Online]. Available: http://g2sebok.incose.org/
99. M. W. Maier, D. Emery, and R. Hilliard, "Ansi/ieee 1471 and systems engineering," *Systems Engineering*, vol. 7, no. 3, pp. 257–270, 2004.
100. A. Aurum and C. Wohlin, *Engineering and Managing Software Requirements*. Springer, 2005.
101. C.-W. Ho, L. Williams, and B. Robinson, "Examining the relationships between performance requirements and "not a problem" defect reports," in *RE '08: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 135–144.
102. L. Lee and P. Kruchten, "Visualizing software architectural design decisions," in *ECSA*, 2008, pp. 359–362.
103. P. Kruchten, *Rational Unified Process: an Introduction*. Reading/MA: Addison-Wesley, 1998.
104. D. Brown and J. Densmore, "The new, improved RUP SE Architecture Framework," 2005, iBM Rational Edge.
105. *IEEE Std 15288 -2004, Systems Engineering -System Life Cycle Processes*, Institute for Electrical and Electronic Engineers, June 2005.
106. A. Fatolahi and F. Shams, "An investigation into applying UML to the Zachman Framework," *Information Systems Frontiers*, vol. 8, no. 2, pp. 133–143, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10796-006-7977-8
107. D. J. de Villiers, *Using the Zachman Framework to assess RUP*, Rational Edge, 2001.
108. H.-P. Hoffmann, *Harmony-SE/SysML Deskbook: Model-Based Systems Engineering with Rhapsody, Rev. 1.51, Telelogic/I-Logix white paper*, Telelogic AB, May 2006.
109. K. Pohl and E. Sikora, "Supporting the Co-Design of Requirements and Architectural Artifacts," in *15th IEEE International Requirements Engineering Conference (RE'07)*, India Habitat Center, New Delhi, 2007, pp. 258–261.
110. D. E. Emery and R. Hilliard, "Every architecture description needs a framework: Expressing archi-

tecture frameworks using ISO/IEC 42010," in *WICSA/ECSA*, 2009, pp. 31–40.
111. "Eclipse Integrated Development Environment." [Online]. Available: https://eclipse.org
112. "Eclipse Papyrus Open Source UML tool." [Online]. Available: https://eclipse.org/papyrus/
113. "Modelio Open Source UML tool." [Online]. Available: https://www.modelio.org/
114. "Visual Paradigm." [Online]. Available: http://www.visual-paradigm.com/
115. "Enterprise Architect." [Online]. Available: http://www.sparxsystems.com/products/ea.html
116. I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 869–891, 2013.
117. J. Holt and S. Perry, *SysML for Systems Engineering. 2nd Edition: A Model-Based Approach*, ser. Computing and Networks Series. Institution of Engineering and Technology, 2013. [Online]. Available: https://books.google.gr/books?id=JlRHAgAAQBAJ
118. O. M. Group, "Meta object facility (MOF) 2.0 core final adopted specification," Object Management Group, Tech. Rep., 2004. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/03-10-04
119. ——, "OMG meta object facility (MOF) core specification," Object Management Group, Tech. Rep., 2013. [Online]. Available: http://www.omg.org/spec/MOF/2.4.1/PDF/
120. O. M. G. Inc, "UML Superstructure Specification, Version 2.1.2," November 2007.
121. J. Mather, "The devsjava simulation viewer: A modular gui that visualizes the structure and behavior of hierarchical devs models," Ph.D. dissertation, UNIVERSITY OF ARIZONA, 2003.
122. "EIS DEVSjava Library Components," Bitbucket, 2014. [Online]. Available: https://bitbucket.org/anargyros_tsadimas/eis-devsjava
123. MG, *SysML Plugin for Magic Draw*, 2007.
124. "MagicDraw Open API User Guide," No Magic Inc, 2013. [Online]. Available: http://www.nomagic.com/files/manuals/MagicDraw%20OpenAPI%20UserGuide.pdf
125. B. P. Zeigler and H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA. DEVSJAVA Manual*, 2003. [Online]. Available: www.acims.arizona.edu/PUBLICATIONS/publications.shtml
126. "Netbeans." [Online]. Available: https://netbeans.org/
127. "medini QVT," ikv++ technologies ag, 2013. [Online]. Available: http://projects.ikv.de/qvt
128. B. W. Boehm, R. Madachy, B. Steece *et al.*, *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.
129. D. Brown, J. Densmore, and S. J. Vaughan-Nichols, "Web services," pp. 18–21, 2002, IBM Rational Edge.
130. "XML Metadata Interchange (XMI), v2.1.1," 2007. [Online]. Available: http://www.omg.org/spec/XMI/2.1.1/PDF/index.htm
131. J.-F. Pétin, D. Evrot, G. Morel, and P. Lamy, "Combining SysML and formal methods for safety requirements verification," in *22nd International Conference on Software & Systems Engineering and their Applications*, Paris, France, Dec. 2010, p. CDROM. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00533311
132. G. Pedroza, L. Apvrille, and D. Knorreck, "Avatar: A sysml environment for the formal verification of safety and security properties," in *New Technologies of Distributed Systems (NOTERE), 2011 11th Annual International Conference on*. IEEE, 2011, pp. 1–10.
133. J. C. Maxwell, A. Anton *et al.*, "Checking existing requirements for compliance with law using a production rule model," in *Requirements Engineering and Law (RELAW), 2009 Second International*

Workshop on. IEEE, 2009, pp. 1–6.

134. C. Kotronis, A. Tsadimas, G. D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "Simulating sysml transportation models," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2016, pp. 001 674–001 679.
135. E. Filiopoulou, P. Mitropoulou, A. Tsadimas, C. Michalakelis, M. Nikolaidou, and D. Anagnostopoulos, "Integrating cost analysis in the cloud: A sos approach," in *2015 11th International Conference on Innovations in Information Technology (IIT)*, Nov 2015, pp. 278–283.

Publications

This chapter presents the publications that were produced throughout this research work, that started from my M.Sc. thesis.

Book Chapters

B1. A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. IGI Global, ch. 8: Model-Based System Design Using SysML: The Role of the Evaluation Diagram, pp. 236–266.

Journal Papers with Review

J1. M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Challenges in sysml model simulation," *Advances in Computer Science: an International Journal*, vol. 5, no. 4, pp. 49–56, 2016.

J2. A. Tsadimas, G.-D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "Simulating simulation-agnostic sysml models for enterprise information systems via devs," *Simulation Modelling Practice and Theory*, vol. 66, pp. 243 – 259, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X16300259

Conferences Papers

C1. C. Kotronis, A. Tsadimas, G. D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "Simulating sysml transportation models," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2016, pp. 001 674–001 679.

C2. E. Filiopoulou, P. Mitropoulou, A. Tsadimas, C. Michalakelis, M. Nikolaidou, and D. Anagnostopoulos, "Integrating cost analysis in the cloud: A sos approach," in *2015 11th International Conference on Innovations in Information Technology (IIT)*, Nov 2015, pp. 278–283.

C3. A. Tsadimas, "Model-based enterprise information system architectural design with

SysML," in *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*. IEEE, 2015, pp. 492–497.

C4. M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, and D. Anagnostopoulos, "Simulating SysML models: Overview and challenges," in *System of Systems Engineering Conference (SoSE), 2015 10th*. IEEE, 2015, pp. 328–333.

C5. A. Tsadimas, G.-D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "Integrating simulation capabilities into SysML for enterprise information system design," in *System of Systems Engineering (SOSE), 2014 9th International Conference on*. IEEE, 2014, pp. 272–277.

C6. G.-D. Kapos, V. Dalakas, A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Model-based system engineering using SysML: Deriving executable simulation models with QVT," in *SysCon*. IEEE International Systems Conference, 2014.

C7. A. Tsadimas, M. Nikolaidou, and D. 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*, ser. SAC '12. New York, NY, USA: ACM, 2012, pp. 1057–1062. [Online]. Available: http://doi.acm.org/10.1145/2231936.2231941

C8. M. Nikolaidou, A. Tsadimas, and D. Anagnostopoulos, "Model-based enterprise information system architecture design using SysML," in *IEEE Systems Conference 2010*, April 2010.

C9. A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Evaluating software architecture in a model-based approach for enterprise information system design," in *SHARK '10*. New York, USA: ACM, 2010, pp. 72–79.

C10. ——, "Handling non-functional requirements in information system architecture design," in *ICSEA '09*, 2009, pp. 59–64.

C11. M. Nikolaidou, A. Tsadimas, N. Alexopoulou, and D. Anagnostopoulos, "Employing Zachman Enterprise Architecture Framework to systematically perform Model-Based System Engineering Activities," in *HICSS-42*, 2009, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2009.189

C12. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Accommodating EIS UML 2.0 profile using a standard UML modeling tool," in *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*. IEEE, 2007, pp. 26–26.

C13. ——, "Facilitating enterprise information system engineering through a UML 2.0 profile: A case study," *Information Resource Management Association (IRMA 2007), Vancouver, British Columbia, Canada*, 2007.

C14. N. Alexopoulou, A. Tsadimas, M. Nikolaidou, A. Dais, and D. Anagnostopoulos, "Introducing a UML profile for distributed system configuration," in *ICEIS: Databases and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006*, 2006, pp. 542–545.

C15. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Extending UML 2.0 to augment control over enterprise information system engi-

neering process," in *ICSEA*. IEEE Computer Society, 2006, p. 10. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICSEA.2006.41

C16. ——, "A consistent framework for enterprise information system engineering," in *EDOC*. IEEE Computer Society, 2006, pp. 492–496. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/EDOC.2006.6

C17. M. Nikolaidou, A. Tsadimas, N. Alexopoulou, A. Dais, and D. Anagnostopoulos, "A UML profile utilizing enterprise information system configuration," in *ICECCS*. IEEE Computer Society, 2006, pp. 77–88. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICECCS.2006.48

C18. M. Nikolaidou, N. Alexopoulou, A. Tsadimas, A. Dais, and D. Anagnostopoulos, "Using uml to model distributed system architectures," in *CAINE*. ISCA, 2005, pp. 91–96.

Short Bio

Anargyros Tsadimas was born in Lamia on 22 October 1979. Since 2004 is working at the Harokopio University as a research associate where is currently Technical Laboratory Staff at the Department of Informatics & Telematics.

He received his B.Sc. in *Applied Informatics* from the University of Macedonia in 2002 and his MSc in *Advanced Information Systems* from the Department of Informatics & Telecommunications of the National and Kapodistrian University of Athens in 2005.

Since 2008 he is adjunct lecturer at the Department of Informatics & Telematics, teaching the laboratories part of the courses: Operating Systems, Distributed Systems, System Analysis and Software Technology.

His research interests lie in the field of Modeling & Simulation of Systems, Distributed Systems and Enterprise Information Systems Engineering. He has several publications in international conference proceedings and he has been participated in numerous R&D projects.

Last but not least, he is open source enthusiast. Since 2007 he has entangled with web applications development and systems administration tasks, mainly administering UNIX and Solaris operating systems.

Acronyms

ATL ATLAS Transformation Language.
AVATAR Automated Verification of reAl Time softwARe.

CIM Computation Independent Model.
COCOMO Constructive Cost Model.
CWM Common Warehouse Meta-model.

DES Discrete event simulation.
DEVS DEVS abbreviating Discrete Event System Specification is a modular and hierarchical formalism for modeling and analyzing general systems that can be discrete event systems which might be described by state transition tables, and continuous state systems which might be described by differential equations, and hybrid continuous state and discrete event systems. DEVS is a timed event system.
DIPLODOCUS DesIgn sPace exLoration based on fOrmal Description teChniques, Uml and SystemC.

EAF Enterpsise Architecture Framework.
EIS Enterprise Information System.
ES Enterprise Systems.

IaaS Infrastructure as a Service.
IEEE Institute of Electrical and Electronics Engineers.
IEEE-Std-1471-2000 IEEE Recommended Practice for Architectural Description of Software Intensive Systems.
INCOSE International Council on Systems Engineering.
IS Information System.
ISE Information Systems Engineering.
ISO International Organization for Standardization.
ITU International Telecommunication Union.

JAXB Java Architecture for XML Binding.
JRT Joint Realization Table.

LAN Local Area Network.

MARTE Modeling and Analysis of Real Time and Embedded systems.
MB-EISE Model-based Enterprise Information System Engineering.
MBE Model-Based Engineering.
MBSD Model-Based System Design.
MBSE Model-Based Systems Engineering.
MDA Model Driven Architecture.
MOF Meta-Object Facility.

NFP Non-Functional Properties.
NFR In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.

OCL Object Constraint Language.
OMG Object Management Group.
OOSEM Object-Oriented Systems Engineering Method.

PaaS Platform as a Service.
PEAS Packaged Enterprise Application Software.
PIM A Platform-Independent Model is a model of a system that does not have any technology-specific implementation informationshortplural.
PLM Product Lifecycle Management.
PSM A Platform-Specific Model is a model of a system that has technology-specific implementation information shortplural.

QoS Quality of Service.
QVT Query / View / Transformation.

RE Requirements Engineering.
RUP Rational Unified Process.
RUP-SE Rational Unified Process for Systems Engineering.

SaaS Software as a Service.
SE Software Engineering.
SLIM Systems LIfecycle Management.
SoS System of Systems.
SysE Systems Engineering.
SysML The Systems Modeling Language (SysML) is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems.

TCO Total cost of ownership is a financial estimate intended to help buyers and owners determine the direct and indirect costs of a product or system. It is a management accounting concept that can be used in full cost accounting or even ecological economics where it includes social costs.
TEPE TEmporal Property Expression language.

UML Unified Modeling Language.
URN User Requirement Notation.

VLAN Virtual Local Area Network.
VSL VSL is a textual language defined in MARTE. It specifies expressions for constraints, properties and stereotype attributes. It enables the value specification, at model level, in tagged values, body of constrainst and in any UML element.
vVDR Virtual Verification of Designs against Requirements.

XMI XML Metadata Interchange.
XML Extensible Markup Language.
XSLT EXtensible Stylesheet Language Transformations.

ΕΠΣ Εταιρικά Πληροφοριακά Συστήματα.
ΜΛΑ Μη-Λειτουργικές απαιτήσεις.

Index