

Animation-Based Service Specification, Verification and Validation

THÈSE N° 6551 (2015)

PRÉSENTÉE LE 6 MARS 2015

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE MODÉLISATION SYSTÉMIQUE

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Biljana BAJIĆ-BIZUMIĆ

acceptée sur proposition du jury:

Prof. M. Odersky, président du jury

Prof. A. Wegmann, directeur de thèse

Prof. V. Kuncak, rapporteur

Prof. Y. Pigneur, rapporteur

Prof. I. Rychkova, rapporteuse



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2015

Thesis Statement

Service specifications are used to represent the service systems on different levels of abstraction: from business down to IT. High-level service specifications are mostly used for communication among different participants, to catalyze the discussions between them; but only service specifications modeling IT systems have enough details to be simulated and executed. As a consequence, it becomes difficult to create precise specifications at high-levels of abstraction, potentially leading to severe project problems. This can be compensated with the use of formal methods and code generation techniques to obtain abstract, yet precise specifications with services that can be simulated and prototyped. By capturing design decisions, the high-level service specifications can be refined into more detailed ones, with the possibility to validate the specification at any level of abstraction.

Abstract

[Context] With the expansion of services and service science, service systems have become an important abstraction for the service revolution. Service is defined as the application of resources (including competences, skills, and knowledge) to make changes that have value for another (system). The service system is a configuration of people, technologies, and other resources that interact with other service systems to create mutual value. Many systems can be viewed as service systems, including families, cities, and companies, among many others. Therefore, services became very important for unifying concepts from various disciplines. Service specifications are used to represent service systems on different levels of abstraction: from business down to IT.

[Motivation and Problem] Traditionally, high-level service specifications are used only for communication among different participants, to catalyze the discussions between them; but only the specifications modeling IT systems have enough details to be simulated and executed. As a consequence, it becomes difficult to create precise high-level specifications and make sure that the implemented services are those that correspond to the business needs, potentially leading to severe project problems. Therefore, the challenge is to create abstract, yet precise service specifications, while keeping the relation between specifications at different levels of abstraction.

[Idea and Results] In this work, we use formal methods and code generation techniques to create service-prototypes from service specifications at any level of abstraction, keeping the relations between different specifications. Stakeholders can try out the prototypes and give feedback regarding services that are being provided. This way, prototypes are used to validate the specifications and detect inconsistencies and unexpected behavior.

[Contribution] The contributions of our work are threefold. First, we provide the visual formalism for service specification and simulation, by adding the necessary concepts to the existing method SEAM. Second, we define two design spirals: for service specification and for service validation and verification. The service specification spiral enables us to keep the relation between several service specifications. It includes steps with explicit design decisions on how to refine high-level specifications in order to include all the details necessary for providing the identified services. The validation and verification spiral is used to validate and verify specifications at any level of abstraction. Finally, it provides an

environment that enables the simulation and prototyping of service specifications that are then used for their validation and verification.

[Relevance] In addition to the theoretical contribution to the knowledge base of service design, we also provide the tools and guidelines that help business and IT analysts create and validate the service model, as confirmed by a survey conducted with practitioners. We illustrate the application of this work with a case study based on a consulting project we conducted at EPFL.

Keywords: Service Design, Service Refinement, Service Validation, Service Verification, Model Prototyping, Model Simulation

Stellungnahme

Service-Spezifikationen werden verwendet, um Service-Systeme auf unterschiedlichen Abstraktionsebenen zu definieren: vom Business-Level bis zur IT. High-Level-Service-Spezifikationen für Geschäftsmodelle dienen vor allem der Kommunikation zwischen den verschiedenen Parteien als Diskussionsbasis, wobei nur IT-Spezifikationen genügend Details für Simulationen bieten. Entsprechend schwierig ist es, präzise Geschäftsmodelle zu formulieren, was Tür und Tor für schwerwiegende Probleme im Projekt öffnet. Dies kann durch den Einsatz von formalen Methoden und Codegenerierungs-Techniken kompensiert werden. Dies resultiert in abstrakten und dennoch präzisen Spezifikationen und kann simuliert und in einem Prototypen implementiert werden. Durch die detaillierte Erfassung von Design-Entscheidungen können die High-Level-Service-Spezifikationen verfeinert werden, was erlaubt, die Spezifikation jeder Abstraktionsebene zu überprüfen.

Zusammenfassung

[Kontext] Im Laufe des Ausbaus der Dienstleistungen und dessen Wissenschaft wurden Service-Systeme zu einer wichtigen Abstraktion in der Dienstleistungsrevolution. Service wird definiert als die Anwendung von Ressourcen (einschliesslich Kompetenzen, Fertigkeiten und Wissen), um Wertsteigerungen in für einen anderen (oder ein anderes System) herbeizuführen. Das Service-System ist eine Anordnung von Menschen, Technologien und anderen Ressourcen, die mit anderen Dienstleistungssystemen interagieren, um gegenseitigen Nutzen zu schaffen. Viele Systeme können als Service-Systeme verstanden werden, wie z.B. Familien, Städte und Unternehmen. Entsprechend wurden Dienstleistungen für vereinheitlichende Konzepte aus verschiedenen Disziplinen sehr wichtig. Service-Spezifikationen werden verwendet, um unterschiedliche Service-Systeme auf unterschiedlichen Abstraktionsebenen zu repräsentieren: von Business bis zur IT.

[Motivation und Problem] Traditionell dienen High-Level-Service-Spezifikationen für Geschäftsmodelle vor allem der Kommunikation zwischen den verschiedenen Parteien als Diskussionsbasis, wobei nur IT-Spezifikationen genügend Details für Simulationen bieten. Entsprechend schwierig ist es, präzise Geschäftsmodelle zu formulieren, was Tür und Tor für schwerwiegende Probleme im Projekt öffnet. Die Herausforderung besteht darin, abstrakte und dennoch präzise Service-Spezifikationen zu erstellen und dabei die Beziehung zwischen Daten auf verschiedenen Abstraktionsebenen zu erhalten.

[Idee und Ergebnisse] Diese Arbeit befasst sich mit High-Level-Service-Spezifikationen und deren Umwandlung in genauere Spezifikationen. Es kommen formale Methoden und Code-Generierungstechniken zum Einsatz, um die Service-Prototypen aus den Service-Spezifikationen auf jeder Abstraktionsebene zu erstellen, wobei die Beziehung zwischen verschiedenen Spezifikationen gewahrt bleiben. Die Benutzer können die Prototypen ausprobieren und Feedback zum erhaltenen Service geben, wobei Daten validiert sowie Inkonsistenzen und unerwartetes Verhalten aufgedeckt werden können.

[Beitrag] Diese Arbeit leistet drei Hauptbeiträge. Zum einen beschreibt sie einen visuellen Formalismus für Service-Spezifikation und Simulation, indem sie das bestehende SEAM-Verfahren um Konzepte erweitert. Das zweite Ergebnis ist eine Reihe von spiralförmigen Prozessen: für Service-Spezifikation sowie für Service-Validierung und

Verifizierung. Der Spiralprozess ermöglicht es, die Beziehung zwischen verschiedenen Service-Spezifikationen aufrecht zu erhalten. Ebenso beschreibt der Spiral-Prozess die Schritte mit expliziten Design-Entscheidungen, um die High-Level-Spezifikationen zu verbessern, inklusive aller notwendigen Angaben damit die identifizierten Dienste gewährleistet werden können. Validierung und Verifizierung innerhalb des Spiral-Prozesses stellt dann das Einhalten der Spezifikationen über die Abstraktionsebenen hinweg sicher. Zum Dritten beschreibt die Arbeit ein Rahmenwerk für die Simulation und das Erstellen der Service-Spezifikations-Prototypen, welche dann zur Validierung und Verifizierung verwendet werden können.

[Relevanz] Neben theoretischen Beiträgen zur Wissensgrundlage im Service-Design beschreibt die vorliegende Arbeit auch Werkzeuge und Richtlinien für Business- und IT-Analysten, die erlauben, das Geschäftsmodell zu validieren. Die praktische Relevanz wurde mit einer Umfrage unter Fachpersonen bestätigt. Zudem führten wir eine Fallstudie im Rahmen eines Beratungsprojektes an der EPFL durch.

Stichworte: Service-Design, Service-Verfeinerung, Service-Validation, Service-Verifikation, Modell-Prototypen-Entwurf, Modell-Simulation

Acknowledgements

First and foremost, I would like to sincerely thank to my thesis director prof. Alain Wegmann for all the guidance and support during my work at EPFL. It was very motivating and inspiring to work with him thanks to his great energy and enthusiasm with which he has always approached any new idea we discussed. He helped me broaden my intellectual horizon by challenging some basics that we very often take as granted. He taught me the beauty of seeing the big picture and not just the pieces of the puzzle, which are often the primary focus of us engineers. He taught me as well the importance of telling a good story. I thank him for the freedom he gave me in my time and “space” management. Thank you Alain for everything, it was such a great pleasure to work with you!

I thank to prof. Claude Petitpierre, my supervisor as well, with whom I had a pleasure to work in the first few years of my PhD. Thank you for all the comments and discussions we had and for creating Arcimboldo, a framework that helped me develop some parts of my thesis. I also thank to all the members of LTI for all the time we spent together.

I would like to express my gratitude to prof. Viktor Kuncak and prof. Irina Rychkova for always being ready for discussions and for all the constructive comments and ideas they gave me during the years I worked on my thesis. I thank the other members of my PhD committee: prof. Yves Pigneur, who contributed to my thesis with valuable academic and industrial insights in business domain, and prof. Martin Odersky who kindly accepted the role of the president.

I thank to Marko Ivanović (GSK, SC Johnson), Lazar Krstić (McKinsey&Company), Olivier Hayard (ITECOR), Anastopoulos Giorgio (EPFL), and Alexandre Herrmann, who dedicated their time to read about my research and participate in the survey. Their comments helped me to better understand the practical value of my research and gave directions for future work.

I thank to Didier Rey and Alain for providing me the great opportunity to test some parts of my thesis in a real project. Thanks for your trust in me and for giving me a chance to conduct interviews and workshops with the relevant stakeholders, despite not having much experience with business analysis before. It was a very valuable experience for me!

I am grateful to all the members of LAMS for making LAMS a great place to be in. Thanks Gorica for all the great time we had together, for being wonderful friend and colleague. Merci Gil pour tout les discussions philosophiques et pour être très patient avec nous quand nous avons parlé français. Thanks Anshuman for the very useful comments you were giving me for my work. Thanks George for making our life sweeter by always bringing all the cakes and fruits to the lab, I will miss that in my future job. Thanks Aarthi for organizing all the events and for making sure I always have my key and adapter for presentations. Thanks Blaise for bringing the relaxed atmosphere to the lab and for taking all our wishes into account when developing SEAMCad. Thanks to all the students whose projects I supervised, whose work helped me understand better some aspects of my thesis. Thanks to everyone else who was in LAMS and to the administrative staff as well. LAMS, thanks to all of you for all the professional and non-professional discussions we had together, it was a pleasure to spend time with you!

I wish to thank to all my friends, who made my life in Switzerland an unforgettable experience. I thank to Nataša, Zlatko and Lenka for being my Swiss family; it is difficult to thank you in just one paragraph, when I could write the whole thesis about our life here. Thank you for all the great moments we had together, for all the positive energy you brought into my life, for providing me not just a place to stay, but home, and for baby sitting me together with Lenka when I was finishing my thesis: you really made me feel as if I am at home. I thank to Giorgia, Sean and Alice for being my other Swiss family. I feel so lucky that I had a chance to meet someone so special as you are, I do not think this happens twice in life. There are no words with which I can express how special you are, I can just wish everyone to have someone like you in their lives. Thank you for being you and for making me change my perspective of this world! I thank Mira for all subtly, sophisticated, deep, bohemian, philosophical discussions we had about our work and life, that I missed in the engineering world. Thank you for all the great moments we had, new insights you brought into my life and of course for helping me bind and submit my thesis!

I thank Mira, Danica and Renata for being great company for many lunches and coffee breaks, for all the fun moments we had together, and for all professional and non-professional comments and advices they gave me. I thank to all my friends who accommodated me in Lausanne. I thank to all friends for making these 5 years special, with all the fun moments we had, filled with parties, skiing, biking, wine tasting, traveling, hens parties, professional and non-professional discussions, lunches, coffee breaks: Milena, Miloš, Andrej, Ivan, Jovana, Milan, Aleksa, Valentina, Maja, Petar, Gemma, Xavier, Michael, Olya, Alejandro, Dražen, Peđa, Aleks, Gvero, Aca, Anja, Boško, Bojana, Massimo, Michelle, Radhakrishna, Jasmina, Mina, Dragana, Marina, Lazar, and many others. Merci beaucoup Alejandro pour être mon tandem en français et pour aimer mon accent français même si c'est très horrible ☺ Danke vielmals Michael für alle Unterichte auf Deutsch und Schwyzerdütsch, für alle wunderbare Momente die wir zusammen hatten und für last-minute Übersetzung meiner Zusammenfassung ☺

I thank to my parents for everything they gave me and for all the love and support I got during these years and before that. As the time passes, I realize more and more how great parents they are and the importance of the values they thought me. The greatest credentials for this thesis probably go to my sister Jelena, who has been my fascination since I was born ☺ She taught me to read, write, play the piano, tie laces, make trouble and everything else. She brings music, heart, soul and life to our engineering family. Without you, I would never see the world through same eyes and I would never be the same person. I am so lucky to have you! ☺ I thank to my nephew Marko, who brought so much joy and happiness into our lives and whose smile has always been the great inspiration and great source of energy!

My greatest thanks I address to my husband Lazar, who was always there for me during this 5 years and before, with his advices, encouragements, love, care and understanding and who is always able to make me laugh.

Table of Contents

List of Figures	11
List of Tables	13
1 Introduction	15
1.1 Main Concepts	16
1.2 Motivation	17
1.3 Business Animated Service Specification (BASS).....	18
1.4 Assessment of Research Project Design.....	22
1.5 Thesis Organization	25
2 The State of the Art	27
2.1 Modeling Methods	27
2.2 Use of Formal Methods and Code Generation Techniques in Modeling.....	38
2.2.1 Model Formalization, Analysis and Verification.....	39
2.2.2 Model Simulation	41
2.2.3 Refinement.....	42
2.2.4 Alloy Applications	43
3 Knowledge Base.....	44
3.1 Systemic Modeling Paradigm and SEAM	44
3.2 Model Transformation	49
3.3 Model Verification and Alloy.....	50
3.4 Code Generation and Arcimboldo	53
4 BASS: Concepts and Visual Formalism	56
4.1 BASS: Concepts and Principles	56
4.1.1 Service and Service System Modeling	56
4.1.2 Declarative Behavior Modeling.....	59
4.1.3 Expressing Semantics of Services: Functional Units	60
4.1.4 Use of Invariant.....	63
4.1.5 Modeling Service Multiplicity	63
4.1.6 Properties and Data	64
4.1.7 Simplified and Full Notation	65
4.2 BASS: Meta-model (Abstract Syntax)	67
4.3 BASS: Visual Notation (Concrete Syntax) and Semantics	70
5 BASS Service Design Spirals: Theory	75
5.1 Service Specification Spiral: Theory	75
5.1.1 Initial Model Design	76
5.1.2 Step 1: External services	77
5.1.3 Step 2: Internal organization	77
5.1.4 Step 3: Internal services.....	78
5.1.5 Step 4: Internal responsibilities.....	78
5.1.6 Step 5: Independent service systems (optional)	79
5.2 Validation and Verification Spiral: Theory.....	79
5.2.1 Initial Model Design	80
5.2.2 Step 1: Generate Samples	80
5.2.3 Step 2: Correct Model Based on Samples	80
5.2.4 Step 3: Check Assertions	81
5.2.5 Step 4: Correct Model Based on Assertions	81
6 BASS Service Design Spirals: Case of Order Creation at Générale Ressorts	82
6.1 Working Example: The Case of Order Processing at Générale Ressorts.....	82
6.2 Service Specification Spiral: Case of Order Creation at Générale Ressorts.....	86

6.2.1	Initial Model Design	86
6.2.2	Step 1: External Services	87
6.2.3	Step 2: Internal organization	87
6.2.4	Step 3: Internal services.....	89
6.2.5	Step 4: Internal responsibilities.....	90
6.2.6	Step 5: Independent service systems (optional)	92
6.3	Validation and Verification Spiral: Case of Order Creation at Générale Ressorts	94
6.3.1	Initial Model Design	94
6.3.2	Step 1: Generate Samples	95
6.3.3	Step 2: Correct Model Based on Samples	96
6.3.4	Step 3: Check Assertions	97
6.3.5	Step 4: Correct Model Based on Assertions	98
7	BASS2Alloy and Back: Transforming BASS Service Model to Alloy and Back..	99
7.1	Input and Output of Simulation	99
7.2	Transforming BASS Model to Alloy Specification	100
7.3	Service Simulation.....	105
7.4	Transforming Alloy Instance to BASS Model.....	106
8	BASS2Java: Transforming BASS Service Model to Application Prototype	107
8.1	Input (Service Specification Template) and Output (Java Application) of Prototyping.....	108
8.2	Mapping Prototypical Specification to Arcimboldo Project.....	112
8.3	Generating GUI.....	116
8.4	Transforming Static Elements (Data)	118
8.5	Transforming Dynamic Elements (Logic).....	120
9	The Practical Impact: Evaluation of the Developed Theory in Practice	125
9.1	Case Study: The Case of Research Project Management at EPFL	125
9.1.1	Initial Model Design	126
9.1.2	Cycle 1: EPFL+	127
9.1.3	Cycle 2: Support organization.....	131
9.1.4	Benefits of Using Proposed Method	136
9.2	Comparison with Other Methods.....	137
9.3	Conducted Survey: Practical Feedback	140
10	Conclusion and Future Research Directions	145
10.1	Theoretical Contribution	145
10.2	Practical Recommendations.....	145
10.3	Limitations and Drawbacks	147
10.4	Future Work.....	147
	Appendix I: SEAM Representation of Hevner Research Framework	151
	Appendix III: Case of Order Creation at Générale Ressorts in Alloy	152
	Appendix IV: Survey Questionnaire	163
	Computer Data Storage at EPFL	164
	Computer Data Storage at EPFL: BASS Models.....	165
	Questionnaire	169
	Glossary	171
	Bibliography	173
	Publications.....	182

List of Figures

Figure 1: Service specification spiral.....	19
Figure 2: Validation and verification spiral.....	20
Figure 3: SEAMCad: The existing tool for creation of BASS models.....	21
Figure 4: Model-to-model transformation tool.....	21
Figure 5: BASS2Alloy and Alloy2BASS tool.....	22
Figure 6: BASS2Java transformation tool.....	22
Figure 7: Information Systems Research Framework.....	23
Figure 8: Diagrams in UML ([16]).....	29
Figure 9: Functional and organizational refinement in SEAM.....	48
Figure 10: Model transformation process (from [104]).....	49
Figure 11: Alloy model of the family example (from [111])	52
Figure 12: Result of simulation of the family example in Alloy Analyzer tool (from [111])	53
Figure 13: Arcimboldo workbench: overview (from [112]).....	54
Figure 14: Arcimboldo example (from [112]).....	55
Figure 15: Service system modeling in BASS.....	57
Figure 16: Service implementation with collaboration.....	58
Figure 17: Service implementation with process.....	58
Figure 18: Full notation to show service	60
Figure 19: Predefined functional unit: generic template	61
Figure 20: Predefined functional unit: graphical representation (Service design).....	62
Figure 21: Predefined functional unit: Alloy representation (Service simulation).....	62
Figure 22: Predefined functional unit: Arcimboldo representation (Service prototyping)	62
Figure 23: Full notation to show service multiplicity.....	64
Figure 24: Diagram in Figure 15 with full notation	65
Figure 25: Diagram in Figure 24 with multiplicity simplification.....	66
Figure 26: Diagram in Figure 24 with service representation simplification.....	66
Figure 27: Diagram in Figure 24 with service semantic simplification	67
Figure 28: BASS: Meta-model.....	68
Figure 29: One cycle of BASS service specification spiral.....	76
Figure 30: Order-to-cash cycle - Adapted from [121]	83
Figure 31: The result of initial model design	87
Figure 32: Step 2: Conceptualize - Identifying provider's roles.....	88
Figure 33: Step 2: Decide - Allocating properties to roles	88
Figure 34: The result of step 2	89
Figure 35: Step 3: Conceptualize - Identifying internal collaborations.....	89
Figure 36: Step 3: Decide - Allocating functional units to collaborations	90
Figure 37: The result of step 3	90
Figure 38: Conceptualize - Identifying processes and services	91
Figure 39: Decide - Allocating functional units to services	91
Figure 40: The result of step 4	92
Figure 41: Step 5: Cutting connecting lines and specifying shared properties.....	93
Figure 42: The result of step 5	93
Figure 43: Anomaly due to Underspecification: "Missing Customer".....	96
Figure 44: Anomaly due to Underspecification: "Delivery to the Wrong Address"	97

Figure 45: Simulation process overview.....	99
Figure 46: Input and output of simulation for Order Creation at GR.....	100
Figure 47: Alloy meta-model for signatures (from [65])	101
Figure 48: Order Creation at Générale Ressorts - Example of Model Instance	105
Figure 49: An example of model instance shown in BASS	106
Figure 50: Overview of BASS to Java transformation	107
Figure 51: Input: Prototypical specification template	109
Figure 52: Output: Page navigation of prototype.....	110
Figure 53: Input of prototyping for Order Creation at GR	111
Figure 54: Output of prototyping for Order Creation at GR.....	111
Figure 55: Descriptor file and the main object	114
Figure 56: The main object: roles	118
Figure 57: Role.tpl for generated page for OrderEntryPerson shown in Figure 54	118
Figure 58: The main object: dataTables.....	120
Figure 59: Expanding template JPA.tpl for generating entity beans (and database tables)..	120
Figure 60: The main object: actions	121
Figure 61: Expanding Logic.tpl to Logic.java	122
Figure 62: Service specification at Générale Ressorts with preconditions	123
Figure 63: State machine for Order Creation business transaction.....	124
Figure 64: Result of conceptualization.....	127
Figure 65: EPFL+: Step 1 Conceptualize	127
Figure 66: EPFL+: Step 1 Decide	128
Figure 67: EPFL+: Result of step 1	128
Figure 68: EPFL+: Step 2 Conceptualize	129
Figure 69: EPFL+: Step 2 Decide	129
Figure 70: EPFL+: Result of step 2.....	130
Figure 71: EPFL+: Step 4 Conceptualize	130
Figure 72: EPFL+: Step 4 Decide	131
Figure 73: EPFL+: Result of step 4.....	131
Figure 74: Support organization: Step 2 Conceptualize	132
Figure 75: Support organization: Step 2 Decide	132
Figure 76: Support organization: Result of step 2.....	133
Figure 77: Support organization: Step 3 Conceptualize	133
Figure 78: Support organization: Step 3 Decide	134
Figure 79: Support organization: Result of step 3	134
Figure 80: Support organization: Step 4 Conceptualize	135
Figure 81: Support organization: Step 4 Decide	135
Figure 82: Support organization: Result of step 4	136

List of Tables

Table I: Guidelines for Assessment of Design Science Research - Adapted from [7].....	23
Table II: Comparing modeling methods focused on behavior modeling.....	37
Table III: BASS: Meta-model Concepts and Corresponding Business Terms.....	69
Table IV: BASS: Visual representation of meta-model concepts and their semantics	71
Table V: Order Creation: Section Main	83
Table VI: Order Creation: Typical Course of Events.....	84
Table VII: Order Delivery: Section Main	84
Table VIII: Order Delivery: Typical Course of Events	85
Table IX: Mapping input and output of simulation for Order Creation at GR	100
Table X: Correspondence between BASS method and Alloy meta-model elements.....	101
Table XI: Mapping input and output of prototyping for Order Creation at GR	112
Table XII: The main object elements and corresponding modeling concepts	115
Table XIII: Table for capturing Order Creation transactions	123
Table XIV: Comparing service blueprinting and our method	138

1 Introduction

In this thesis, we propose a visual modeling method, named Business Animated Service Specification (BASS), for service specification, verification and validation. It is aimed at helping business and IT analysts to create more precise models for the business cases they are working on.

To better understand a business case and act more effectively on it, we propose using visual models for the description, analysis and communication of concepts. While creating a model, the business/IT analyst observes some aspect of reality, and sees a set of entities, which is a subset of the total number of entities available in reality. This set of entities is called the universe of discourse. The analyst builds a set of concepts, which we call his conceptualization, by interacting with his universe of discourse. This conceptualization is the basis of his understanding of the business case. He then creates a model in the representation domain as a set of entities called modeling constructs.

In our proposed method, models are created using the modeling constructs based on the Systemic Enterprise Architecture Method (SEAM) [1]. As a systemic method, it is focused on modeling the systems, with its theoretical foundations in General Systems Thinking (GST) [2]. According to this theory, a system can be of any nature (IT, human, company, etc.). Throughout the thesis, whenever the system type is not specified explicitly, by system we mean IT, human, company, or any other.

With the popularization of services and service science, the authors of [3] propose service system as an important abstraction for the service revolution. According to [3], service is defined as the application of resources (including competences, skills, and knowledge) to make changes that have value for another (system). The BASS model describes a hierarchy of service systems as a configuration of people, technologies, and other resources that interact with other service systems in order to create mutual value. Many systems can be viewed as service systems, including families, cities, and companies, among many others [3].

The BASS model shows service specifications at any level of abstraction; these specifications correspond to different viewpoints that the analysts develop to simplify their understanding of systems. The model can show the main service of one service system, or its implementation with a collaboration or with a process that involves many service systems. Collaboration defines how a component's service systems implement the main service together, showing only the net effect of their interaction. Process defines the responsibilities of each of component's service systems in implementing the main service. In this thesis, by service specification, we mean the BASS visual model of the service at any level of abstraction.

Several efforts have been made to introduce service concepts to the existing systemic method SEAM [4], [5], [6]. In this thesis, we propose BASS, which extends SEAM with the concepts for service simulation and validation.

The method BASS includes the following:

- The set of concepts and the visual formalism for service specification based on SEAM. We have extended SEAM with concepts for service simulation, such as a functional unit, event, send and receive properties. Service specifications on different levels of abstractions are expressed using visual models. This is explained in Chapter 4.
- Two spirals: (1) A service specification spiral that guides analysts on what design decisions to make in order to provide and implement services that would satisfy the customer; (2) A validation and verification spiral for checking the models on any level of abstraction and resolving identified anomalies. This is explained in Chapters 5 and Chapter 6.
- Tools for model creation, transformation, simulation and prototyping. This is explained in Chapters 7 and Chapter 8.

BASS models can be created by business/IT analysts themselves or during workshops with different stakeholders. Over-formalizing the models during the workshops can distract participants from focusing on details of the modeled situations, and make them focus too much on the modeling rules. Therefore, we use simplified BASS for this purpose. Once the overall conclusion has been made during the workshop, specific models can be improved by using the validation and verification spiral. This way, in the first iteration we can focus on the overall story and in the second, on the finding anomalies, missing business rules and other.

In this chapter, we first clarify the main concepts used in the thesis, such as verification, validation, simulation and prototyping. Then, we explain our motivation for the work in the thesis. Next, we give a brief overview of the proposed method. We end the chapter with an assessment of the research design pursued in this work. For this purpose, we use the seven guidelines defined in Design Science [7].

1.1 Main Concepts

In the proposed method, service specifications at any level of abstraction can be simulated and prototyped. By simulation, we mean generating instances of the model that satisfy the constraints of the model. These instances correspond to the state of the system before and after service execution. This is achieved using translation to Alloy language and simulation with the Alloy Analyzer tool.

By prototyping, we mean generating the prototype of a Java application in which the user can test the behavior of the service system by entering input values and observing outputs. This is achieved using the Arcimboldo framework for easier prototyping process. As the goal is only to obtain feedback about the behavior of the service system, and not to generate the full application, we do not consider the requirements such as scalability, reliability, availability and security. Therefore, we refer to this process as prototyping and not execution.

Simulation with Alloy is used for service validation and verification. Prototyping with Arcimboldo is used for validation. Validation is checking that a service or a system meets the needs of the customer or other stakeholders. Therefore, it includes the acceptance from customers and stakeholders. Both with simulation and prototyping, we use instances and prototypes of the model to obtain feedback from customers and other stakeholders.

Verification is checking that a service or a system conforms to the specifications, regulations or imposed conditions. It is often an internal process and does not include stakeholders. In our case, we use simulations to verify that the model conforms to the meta-model, well-formedness rules and other constraints.

1.2 Motivation

”Modeling is much more fun when you get instant, visual feedback. When you simulate a partial model, you see examples immediately that suggest new constraints to be added.” [2] We agree with Daniel Jackson, the founder of the Alloy specification language: It is very effective to use simulation to get instant, visual feedback on how design decisions influence the design. Examples of the partial specification can help us to realize what constraints in the model are missing. This way, model simulations can help us to involve the customer and other stakeholders in the design, by including their feedback already at the early stages.

Traditionally, high-level service specifications are used only for communication among different participants, to catalyze the discussions between them, but only the specifications modeling IT systems have enough details to be simulated and executed. As a consequence, it becomes difficult to create precise business models and to make sure that the implemented services are those that correspond to business needs, potentially leading to severe project problems. Therefore, the challenge is how to help business and IT analysts to create abstract, yet precise service specifications, keeping the relation between specifications on different levels of abstraction.

Services have been applied in different domains, including business and IT specifications. Throughout the history, IT specification and implementation have changed. IT experts have always sought to improve productivity by using more abstract specifications, thus also closer to the business specification and design. The new level of abstraction has automatically been transformed to the earlier ones. After the first (1GL) and second (2GL) generation programming languages, moving to the third generation languages (3GL) has improved productivity significantly. Several lines of 2GL code have been replaced with only one in 3GL. At first, functionality was specified directly in code. Then models, as the next level of abstraction, were introduced. They facilitate the communication among different users. For a very long time, the de-facto standard for modeling was the Unified Modeling Language (UML) [8]. However, the UML models are mostly maintained separately from code, so inconsistency was one of the main issues of such a specification. Also, the vocabulary used in UML is too technical and not easily understandable by domain experts. The final level of abstraction was introduced with domain-specific modeling (DSM) [9]: it abstracts the vocabulary relevant to a specific domain. This vocabulary is understandable to domain experts. Also, the models in DSMs are used as primary artifacts in the development process: there are source models instead of source code. Hence, there is no problem with inconsistency between models and code. Also, it is possible to execute partial models with DSMs, which helps us to find errors in the models and check their validity [9].

On the other side, business experts have sought to improve productivity by being more precise and concrete. Business specification was first given in plain language, such as

English. Then, for better communication, business experts started using sketches and visual methods. Finally, they use more concrete and precise business models that can be executed, such as business process model and notation (BPMN) [10].

However, with existing business specification methods it is not possible to simulate partial and high-level business models. Usage of models for business specification that can be directly simulated and prototyped, for partial and complete models, could increase productivity, validate models and detect possible errors at early stages of the design and in this way decrease costs. Therefore, the challenge is to provide a way to simulate high-level service specifications in order to validate and verify them.

The nature of business models suggests the use of methods that provide rich representations that appeal to non-technical stakeholders as well. However, such created models are difficult to verify and validate automatically. Formal methods and code generation techniques can be used to provide more formal models. This requires the formalization specialist as a support to the stakeholder who provides information and makes decisions. The degree of formality for a support framework therefore needs to reflect this relationship. Formalization can facilitate reasoning; however, overformalizing can be harmful.

So far, formal methods and code generation techniques have been applied in visual modeling for IT-focused models, in which the goal is to specify the details of IT implementation. They are not applied for high-level service models, in which the goal is to show the overall perspective of the system and abstract the behavior for the customer. Thus, the challenge is to apply formal methods and code generation techniques to simulate and prototype service specifications at any level of abstraction, including high-level models. This would enable their validation and verification and help business and IT analysts to create more precise models.

1.3 Business Animated Service Specification (BASS)

We propose the method named Business Animated Service Specification (BASS). It is intended for service specification and validation based on SEAM. We extend SEAM notation with the elements necessary to model the services and simulate and prototype their behavior. As we have mentioned, simulation results with the snapshots of the system at the given moment in time, for example before and after the service execution. The simulation is used to verify the model against the meta-model, to discover hidden business rules and to resolve possible anomalies by observing the behavior of the system. Prototyping results with the application prototype in the given target language. This way the designer can interactively, by entering the input parameters, check how the system behaves and together with the stakeholders validate if the behavior corresponds to the business needs.

The BASS method includes: the concepts and visual formalism necessary to specify services; service specification spiral and validation and verification spiral; tools for model transformation, simulation and prototyping.

The BASS concepts and visual formalism are based on SEAM and are explained more in Chapter 4. The two spirals are based on a spiral model proposed by Boehm. The service specification spiral (Figure 1) includes four main activities in each cycle of the spiral, represented with four quadrants of the spiral. These activities correspond to the main steps used to transform the initial model (created in the conceptualization phase of the spiral) to the more detailed models, reflecting the main principles of the services and service science. The result of each step is shown as a dot on a dashed line is the new model. One iteration of the spiral contains two models showing the modeled organization with its environment (for instance, customer), two models showing the internal organization required to implement the necessary services, and one model showing the responsibilities of each of internal organizations in implementing the services.

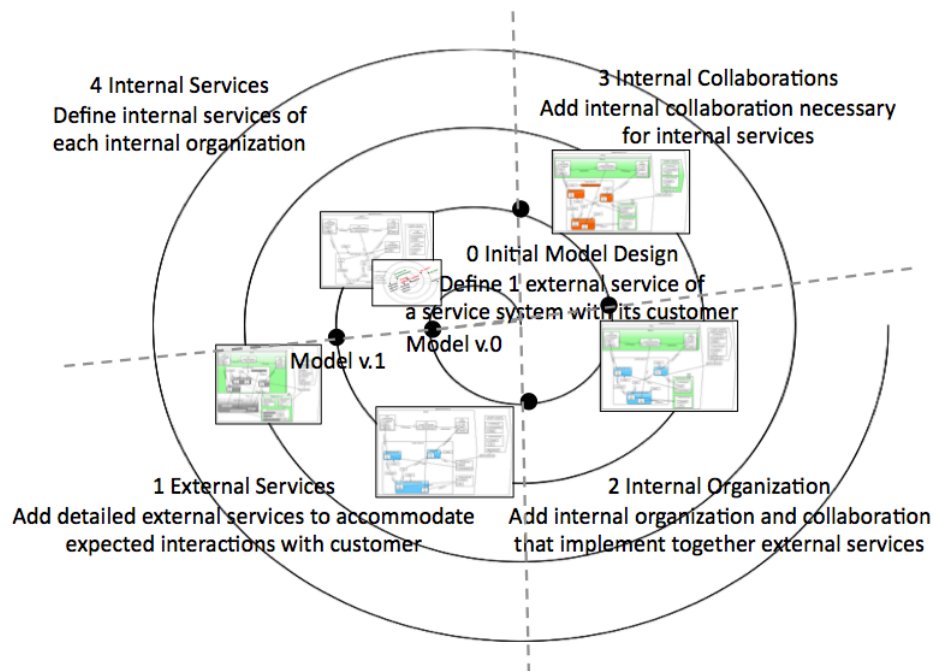


Figure 1: Service specification spiral

The initial model contains the company or organization in its environment and the external service, i.e., the service relevant to the environment, such as a customer. Following the steps of the spiral, we add details necessary for providing this service to the customer. Once we finish with one cycle in the spiral, we can continue to refine one of the sub-organizations by following the steps of the spiral. We continue the process until we reach the level of details required for the analyzed project. This refinement process is described in [11]. This spiral process is the core of the method and it is always used for service specification either by the business and IT analysts themselves, or for facilitating the workshops with stakeholders. The detailed explanation of the steps of the spiral can be found in Chapter 5.1 and the illustration with an example in Chapter 6.2.

In addition to the service specification spiral, for each model in the process we can optionally use simulation and prototyping for validation and verification of the model. The application prototypes and model instances generated with simulation can be used to receive feedback from stakeholders; and in this way, to validate the model. The spiral for creating the

models that include service validation and verification is shown in Figure 2. This spiral has four main activities that are combined in one cycle. Initially, the first model is created in Alloy and then it is validated and verified by following the steps of the spiral. The Alloy specification of model in Alloy can be created manually or automatically using the tool. A detailed explanation of the steps of the spiral can be found in Chapter 5.2 and the illustration with an example in Chapter 6.3. The initial idea of spiral with the example of application is described in [12].

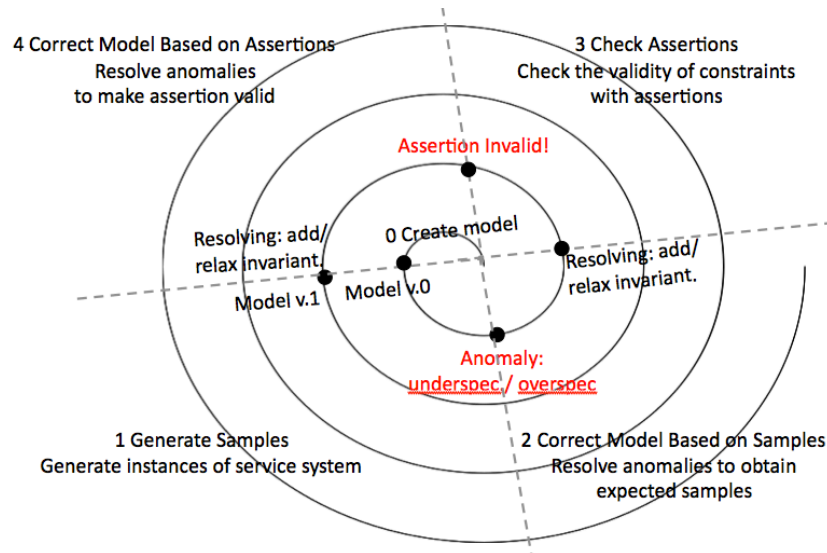


Figure 2: Validation and verification spiral

Finally, BASS contains the tools for service specification, validation and verification. To specify services, we use tools for model creation and transformation. The tool used for model creation is SEAMCad (Figure 3). SEAMCad is the existing tool also used for creation of SEAM models. Using the stereotypes, we add special modeling concepts used for BASS models. The initial model is always created in SEAMCad.

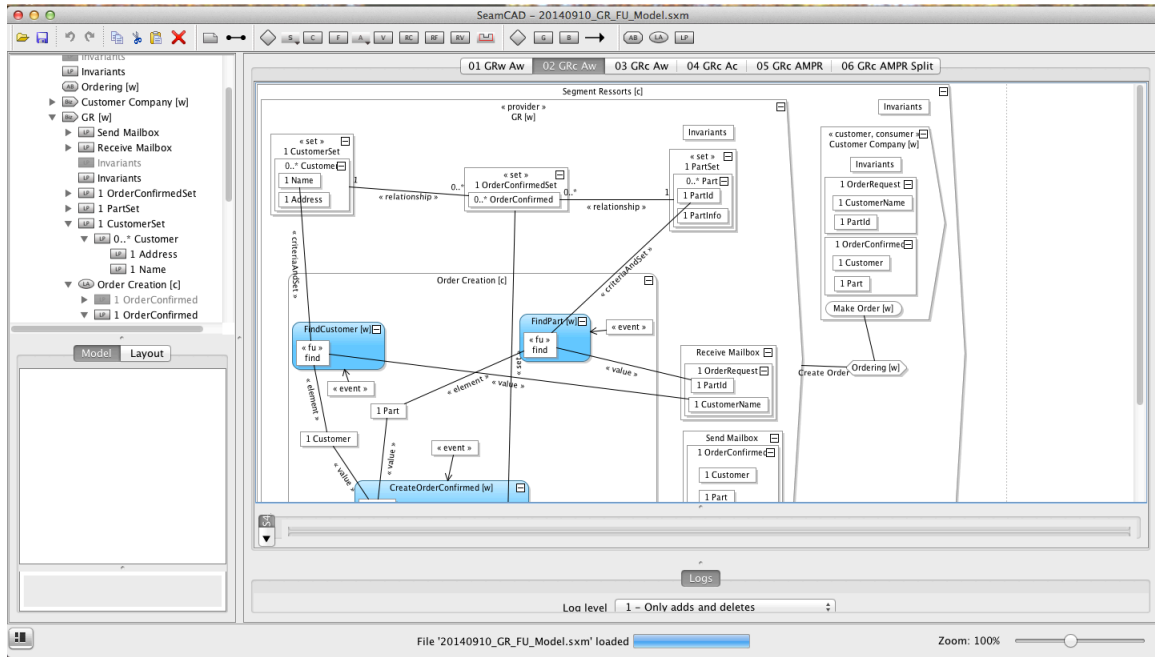


Figure 3: SEAMCad: The existing tool for creation of BASS models

We developed a tool for model transformation (Figure 4) to refine the initial model to more detailed models, following the steps of the service specification spiral. For the BASS model as input, and design decisions captured in the tool, the tool generates the refined BASS model as output. Transformation from input to output corresponds to one step in the service specification spiral described in Chapter 5.1 and Chapter 6.2.

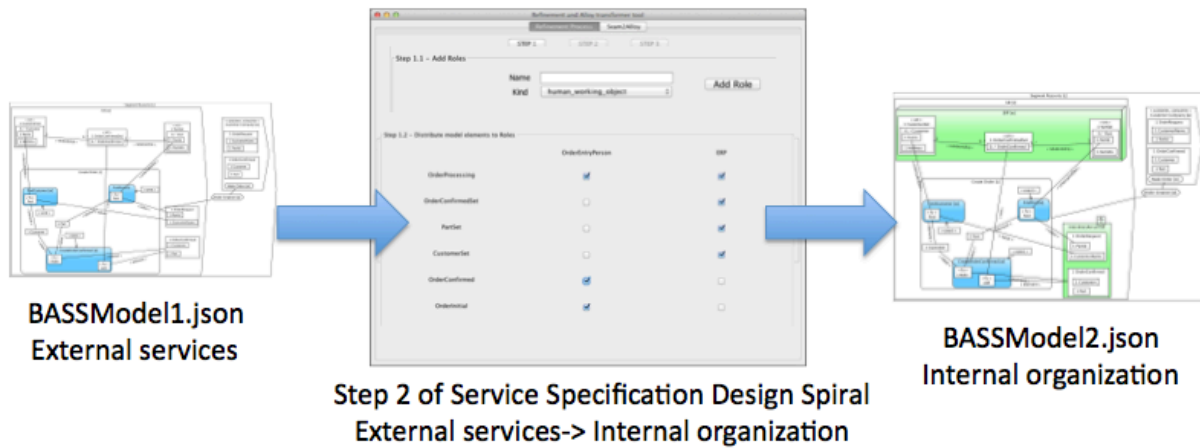


Figure 4: Model-to-model transformation tool

For model simulation, we developed the tool that transforms the BASS model to an Alloy specification with the possibility of running the Alloy specification and observe the generated instances. This tool also has the possibility of transforming the generated Alloy instance back to BASS thus making it more readable to people already familiar with BASS notation. The tool is shown in Figure 5. The details of transformation are captured in Chapter 7.

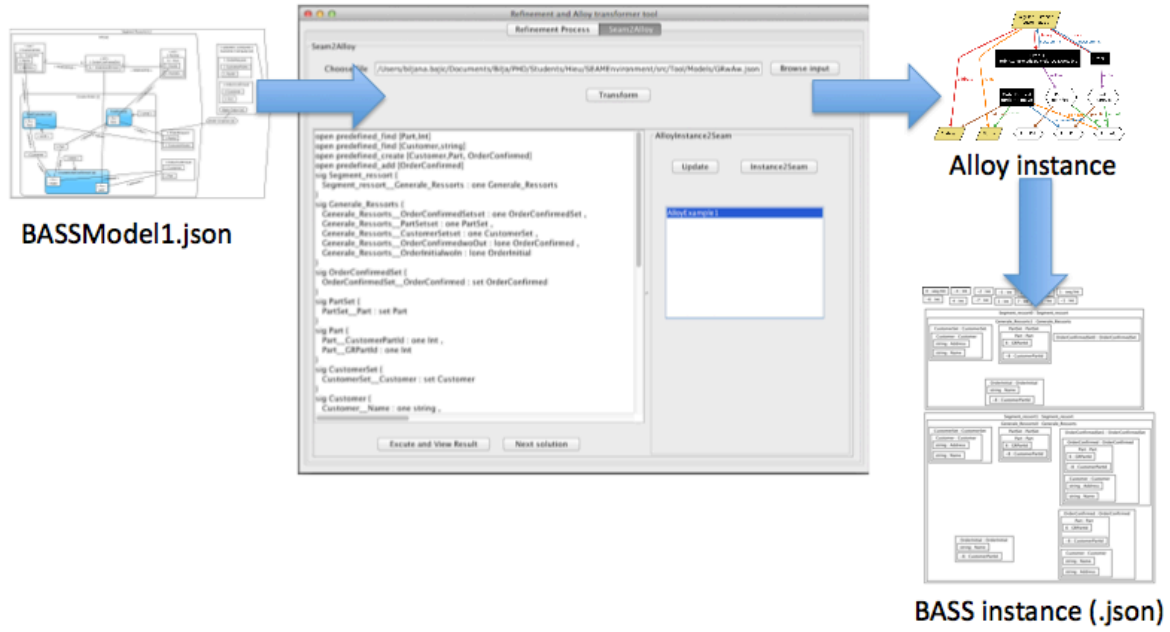


Figure 5: BASS2Alloy and Alloy2BASS tool

For model prototyping, we have developed a tool that transforms BASS model to Arcimboldo project and Java prototype as shown in Figure 6. The details of transformation are captured in Chapter 8.



Figure 6: BASS2Java transformation tool

1.4 Assessment of Research Project Design

The research approach used in this work is the Design Science research methodology, more specifically the research framework proposed by Hevner [7], illustrated in Figure 7.

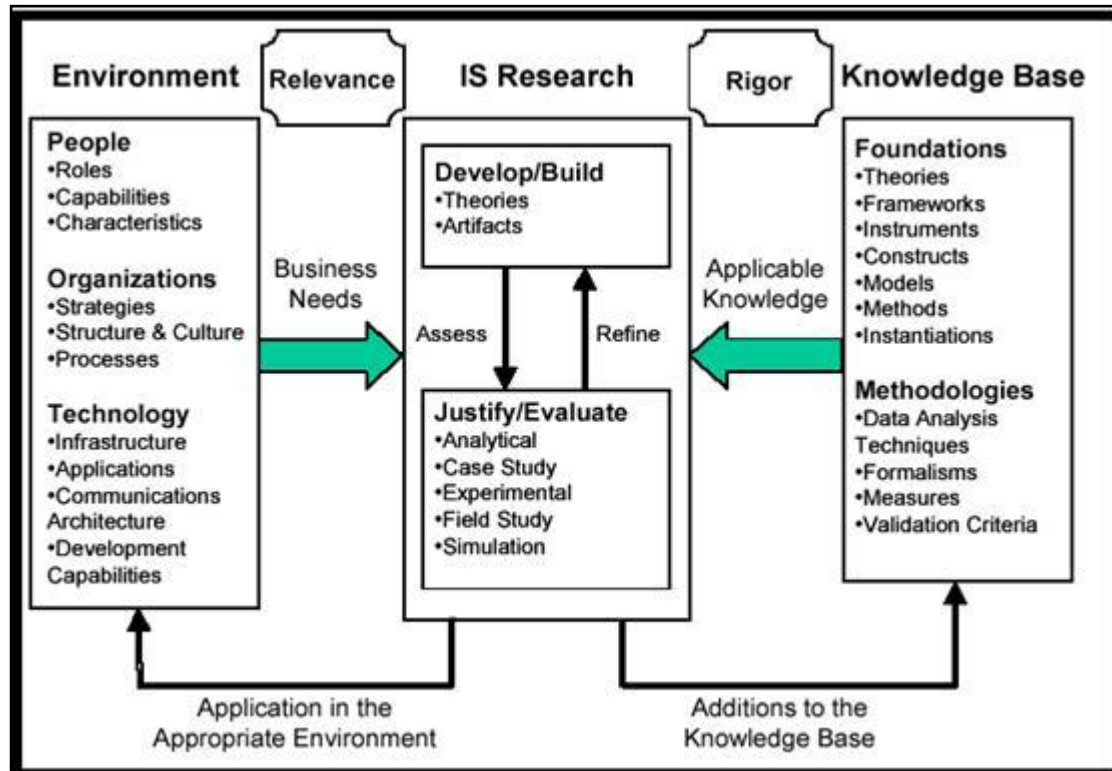


Figure 7: Information Systems Research Framework

As opposed to the natural sciences, e.g. physics and biology, which focus on explaining the phenomena studied, the design sciences, which include all forms of engineering, medicine, aspects of law, architecture and business, focus on usefulness. Design science research requires creation of an innovative, purposeful artifact for a special problem domain. The main artifact of this work is the method for service design, simulation and prototyping. Its goal is to help business and IT analysts in their projects by modeling precise business specification and keeping the relation with lower level specifications and finally with IT specification. Overview of this research framework represented in SEAM is shown in **Appendix I**.

In [7] Hevner presents a set of seven guidelines for conducting, evaluating and presenting design science research. In Table I, we list these guidelines along with their description and discuss how the work presented in this thesis meets these guidelines.

Table I: **Guidelines for Assessment of Design Science Research - Adapted from [7]**

Guideline	Description	Discussion
“Design as an Artifact”	“Design Science research must produce a viable artifact in the form of a construct (vocabulary & symbols), a model	The main artifact produced through this work is a method for service specification, validation and verification, which includes: - The visual formalism for service specification

	(abstractions & representations), a method (algorithms & practices), or an instantiation (implemented & prototype systems).”	<p>- The service specification spiral including the steps on how to transform high-level specification to the more detailed one, to make sure we provide the required services to the customer</p> <p>- The validation and verification spiral including the steps on how to validate and verify model corresponding to the service specification at any level of abstraction</p> <p>- The tools for modeling, simulation and prototyping of service specification.</p>
“Problem Relevance”	“The objective of Design Science research is to develop technology-based solutions to important and relevant business problems.”	<p>This work is addressing the following questions:</p> <ol style="list-style-type: none"> 1. How to help business and IT analysts to validate the models and create more precise models even on business level? 2. How to help business and IT analysts to use customer-oriented approach for modeling business cases, which puts all people relevant for providing the service on the same page, enabling them to see their role in customer satisfaction?
“Design Evaluation”	“The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well executed evaluation methods.”	The design evaluation method used belongs to the observational class. We have conducted a case study and demonstrated the benefits of the proposed method on it. Also, we have conducted a survey with practitioners to evaluate if this work helps the business and IT analysts in modeling business cases.
“Research Contributions”	“Effective Design Science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.”	<p>This work contributes to the service specification by providing a way to simulate service specifications already at early stages. Also, it contributes to the service specification with the process containing the design decisions to be made to model all the details necessary to provide required service. In addition, it shows how formal methods and code generation techniques can be applied in business domain as well.</p>
“Research Rigor”	“Design Science research relies upon the application of rigorous methods in both the construction and	From a rigor perspective, the proposal made in this research is based on general systems thinking theory, model verification and validation, and service science .

	evaluation of the design artifact.”	
“Design as Search Process”	“The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.”	We developed our search for an effective artifact within these constraints: - abstract, yet precise business model - service specification method providing visual feedback .
“Communication of Research”	“Design Science research must be presented effectively both to technology oriented as well as management-oriented audiences.”	To facilitate the communication of research results to a wider community, including both the technology-oriented and management-oriented audiences, we developed a set of service specification guidelines as well as a modeling and simulation tools to facilitate the use of proposed method. These guidelines were found to be effective in conveying the research results to the diverse participants of survey that was conducted using these guidelines as well as to the participants of different conferences and workshops.

1.5 Thesis Organization

This document is organized to capture all the research elements as explained by Hevner research framework in Figure 7. We first explain the business needs that motivated this research and explain why the research is relevant for real-world problems in Chapter 1. Then, we describe the used knowledge base that gives rigor to the research in Chapter 2 and Chapter 3. We explain the proposed method BASS in Chapter 4, Chapter 5, Chapter 7 and Chapter 8. We evaluate the work in Chapter 9 using case studies and the survey with practitioners. Finally, we explain the contributions and limitations of the thesis and give directions for future research in Chapter 10.

In Chapter 2 of this document, we analyze the state of the art. As the research question is how to create more precise models, we first give overview of modeling methods and comparison between them. Then, we describe how formal methods and code generation techniques are used in this modeling.

In Chapter 3, we describe the theories used to develop the proposed method. The proposed method is an extension of the existing Systemic Enterprise Architecture Method (SEAM). Therefore, we first explain General Systems Thinking (GST) and SEAM. Then, we give overview of service science with explanations of service-related concepts. Next, we explain model verification techniques and Alloy [13], declarative language used for simulation in this thesis. Finally, we explain model validation techniques and Arcimboldo, approach for generating various kinds of application, used to create application prototype in this thesis.

In Chapter 4, we describe the main concepts and visual formalism of proposed method. We first describe SEAM advancements that were necessary in order to model, simulate and prototype services. Then, we give an overview of the used visual notation and meta-model of the proposed method.

Chapter 5 describes the main processes of the proposed method: service specification spiral process and validation and verification spiral process. It describes the process of modeling services and validating them using simulation and prototyping.

Chapter 6 illustrates the processes explained in Chapter 5 on the working example. The working example is based on the consulting project we have conducted in the company Générale Resorts. It is described in plain English and with UML use cases as proposed in [14]. After the description of the working example, we illustrate the application of the two processes on this example.

Chapter 7 describes how BASS models are transformed to Alloy specification. This is illustrated on the working example described in Chapter 6.

Chapter 8 describes how BASS models are transformed to Arcimboldo project and then to application prototype in the given target language. This is illustrated on the working example described in Chapter 6.

Chapter 9 describes how the method has been evaluated. Used evaluation method belongs to the category of the observational methods. We illustrate the benefits of the method with the case study based on the project we have conducted. Next, we give comparison with other methods. Finally, we explain the survey we have conducted with practitioners and give the feedback we have received from them about the usefulness of the proposed method in practice.

Finally, Chapter 10 summarizes the work done. It describes the contributions of this thesis as well as the limitations of the proposed method. It also gives the possible continuation of this work and possible future directions of research.

At the beginning of each chapter, we give an overview of the chapter's content.

For the reader interested in service design, we recommend focusing on Chapters 5.1, 6.1 and 6.2.

For the reader interested in simulation and prototyping we recommend focusing on Chapters 5.2, 6.1 and 6.3. For the reader interested in details of transformation to Alloy and Arcimboldo we recommend focusing on Chapter 7 and Chapter 8, respectively.

For practitioners working with business and/or IT specification, we recommend reading Chapter 2 for related work in these fields and Chapter 6 explaining specification with the proposed method, as well as 9.3 for the feedback we got from practitioners.

2 The State of the Art

In this chapter, we give an overview of the literature and methods related to the work presented. We propose method for visual modeling, validation and verification of services. Therefore, we analyze the related work in two fields: visual modeling methods including service specification methods; and application of formal methods and code generation techniques to model validation and verification.

We first present the visual modeling methods used in general in business and IT, as well as methods intended for service specification. We explain the set of related methods and the table comparing all of them. Then, we discuss the work related to the application of formal methods and code generation techniques to modeling, i.e. model validation and verification. We discuss different examples of formal methods and code generation techniques applied to modeling methods and more specifically application of Alloy to different domains.

2.1 Modeling Methods

In this chapter, we explain various modeling methods and languages used in business and IT in general, as well as methods intended for service specification. We also give the table comparing the methods by several criteria.

The Catalysis approach [15] is a component-oriented development method that analyzes and designs in three levels: business (problem domain terminology, business process, roles, collaborations), component specification (component and/or system interface) and component implementation (internal architecture and insides of the system and/or component). It uses its own notation inspired from UML. It uses the same notation on all levels of design. The basic concepts in Catalysis are the object and the action. The object represents a cluster of information and functionality; the action represents anything that happens: an event, task, job, message, change of state, interaction, or activity. Catalysis places the action on an equal footing with the object, because good decoupled design requires careful thought about what actions occur and what they achieve [15].

Models of objects and actions in Catalysis have 3 parts: static, dynamic and interactive. The static part of the model represents the state of an object at any given time. The main purpose of this part is to provide a vocabulary in which actions are described. The interactive part of the model deals with interactions between objects. It shows how the responsibility for achieving a goal is divided among collaborating objects and how object interactions can be abstractly described. The dynamic part deals with the changes that happen to the state as events occur. It specifies objects' behavior (a list of actions it can take part in and the way it responds to them). At any level of design, we use the models containing static, dynamic and interactive part.

An object's behavior can be described with a type specification. A type specification is a set of action specifications that share a static model that provides a vocabulary about the state of any member of the type. It is possible to define subtypes of other types that inherit all specification of the supertype and may add further specification.

An action specification gives information only about the effects of the action without any information about what occurs inside. It is represented with pre- and postcondition. A postcondition is a read-only Boolean function that specifies the outcome of the action. It defines a relationship between the states before and after an action has happened. Its focus is on the effect of an action on an object's internal state. It can also specify results of an action that are returned to the invoker. A precondition is a read-only Boolean function that defines when the associated postcondition is applicable. If the precondition is not true when the action starts, we cannot tell what the outcome will be. There may exist some other action specification for that precondition. Precondition and postcondition can be written in any language. In Catalysis they are written using OCL. This way, actions are specified declaratively with pre- and post- conditions, without specifying explicitly the sequence of steps.

An action specification generalizes all occurrences of the action. However, it is very different from an action implementation. A specification is a Boolean expression (a relation between the inputs, initial state, final state and outputs). An implementation chooses a particular algorithmic sequence of steps, selects a data representation, etc. One action specification can have many implementations. Similarly, there are specification types used to write specifications that describe how a client can use a component and design type that will be implemented.

An alternative view of an action specification is a state chart. It shows state transitions. A state is represented by a Boolean attribute: an object either is or is not in that state at any time.

Another major theme of Catalysis besides behavior modeling is precise abstraction: the ability to look at a design or a model in only as much detail as necessary and without loss of precision. Both actions and objects can be shown from different views, on different levels of details. The zoomed-in detailed views and zoomed-out abstract views must be clearly related. This way, it is possible to show hierarchical representations of actions and objects.

The Unified Modeling Language (UML) [8], an Object Management Group (OMG) standard for modeling software, 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. Much academic and industrial research has been put into the development success of UML since its inception. Hence, UML is the de facto international standard notation for software design.

The latest version of UML is 2.5 [8], which includes 14 diagrams grouped into 2 categories: structure and behavior, as shown in Figure 8. Structure diagrams emphasize the things that must be present in the system being modeled. Behavior diagrams emphasize what must happen in the system being modeled. Behavior category includes a few diagrams that represent different aspects of interaction. Interaction diagrams relate a system structure defined in the structure diagrams with its behavior, specified in behavior diagram.

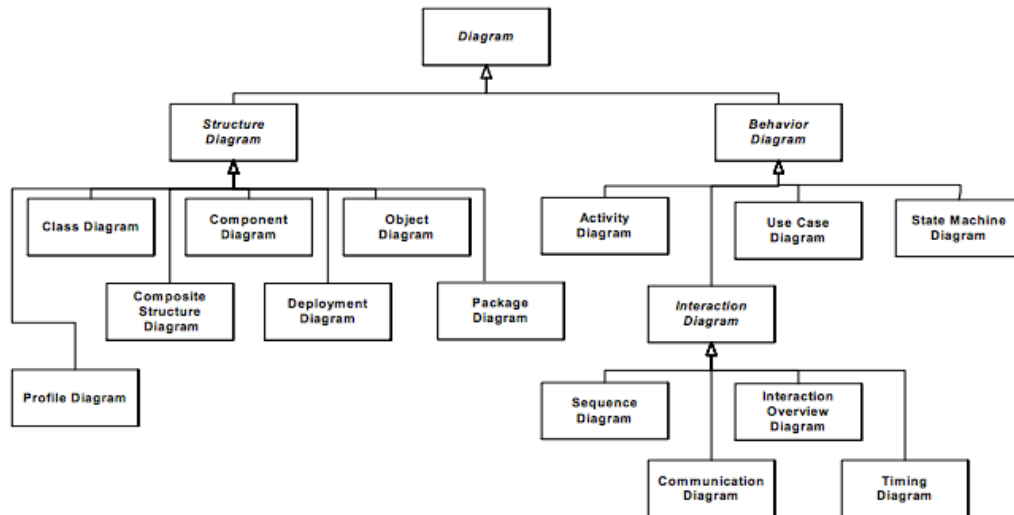


Figure 8: Diagrams in UML ([16])

Some of the diagrams can have a hierarchical structure, such as: activity diagram, state machine diagram, use case diagram and package diagram. For example, a state machine diagram defines state machines and submachines; activities are composed of activity nodes that can be also activities, etc.

There is a semantic relationship between the UML diagrams of different types, i.e. they are complementary. However, these relationships have to be maintained mostly manually by the designer. Traceability in UML can be expressed using traceability relationships, in form of diagrams and table views of related model elements, broken relationships and dependencies between model elements. UML 2.5 specification [8] does not address explicitly the traceability issue.

Traditionally, the UML has been associated more with software engineering and systems design than with analysis and modeling of business processes. However, standard UML 2.x provides a rich set of behavioral models, which are very useful in modeling the processes, activities, people and information critical to every business.

Designers and practitioners worldwide have developed integrated development environments (IDEs) and other tools that allow UML designers to: draw diagrams easily (drag & drop), generate code automatically, apply design patterns, understand and represent requirements, reverse engineer a design problem or perform impact analysis, to name only a few.

Systems Modeling Language (SysML) [17] was developed by OMG as an extension of a subset of the UML for systems engineering using UML's profile mechanism, modeling wide range of systems, which may include hardware, software, information, processes, personnel, and facilities. It removes many of UML's software-centric constructs, creating the language smaller both in diagram types and total constructs. SysML reuses seven of UML 2.x's fourteen diagrams, and adds two diagrams (requirement and parametric diagrams) for a total of nine diagram types.

SysML defines blocks as modular units of system description. Blocks group both structural and behavioral features (properties, states, operations) to describe a system of

interest. The Block Definition Diagram in SysML defines features of a block and relationships between blocks. The Internal Block Diagram in SysML captures the internal structure of a block. Blocks can be decomposed into parts that are also blocks. Relations between the blocks can be used for traceability. SysML models can be translated to Matlab/Simulink [18] and other simulation packages [19].

Archimate [20], an Open Group Standard, is an enterprise modeling language that enables enterprise architect to describe, analyze and visualize the relationships between business domains in an unambiguous way. It is supported by different tool vendors and consulting firms. The current standard is Archimate 2.1. It is focused on coherence and overview of an enterprise rather than specificity and details. It provides a common language for describing different aspects of an enterprise, such as business processes, information flows, products, applications and infrastructures. This facilitates the design, assessment and discussion of consequences of changes within and between these business domains.

Archimate provides integrated view of organization's architecture, structured in three main layers [21]:

- Business layer – products and services offered to external customers and corresponding business processes performed by business actors and roles
- Application layer – supports the business layer with application services that are realized by software components
- Technology layer – supports the application layer with infrastructure services needed to run applications realized by hardware and software.

Although the concepts used in layers are similar, each layer introduces several more concrete concepts specific for that layer or renames some of the existing concepts. Each of the layers can be further divided into sub-layers.

The relation between layers is formed by use and realization relations. Use relation shows how the higher layers make use of the services in lower layers. Realization relation shows how the elements of the lower layers realize the comparable elements of the higher layers.

The Zachmann framework (framework for information system architecture) [22] is an enterprise architecture framework that provides formal and structured way of defining an enterprise. It is created based on analogy between traditional building architecture and IT systems architecture. It suggests to use two-dimensional classification matrix whose rows describe different perspectives of the system and columns describe different types of description, i.e. the kind of questions that can be asked about a given view. Initial framework is proposed in [22] and suggests a six by three matrix. It has been extended with three more questions introduced in [23] to a six by six matrix. The framework has evolved throughout the years [24] and has several versions; all of them capture enterprise's aspects with the use of a structured matrix. The rows of the matrix correspond to the following perspectives:

- Planner (scope) – the big picture of organization with the scope of the project; corresponds to the executive summary for a planner or investor.

- Owner (business model) – business entities and processes related to the daily running of business and how IT system supports that; corresponds to the business model that constitutes the design of business.
- Designer (system model) – the design of the IT system that fulfills owner's business needs; corresponds to the system model designed by system analyst who must determine the data and functions that represent business entities and processes.
- Builder (technology model) – the construction of the IT system specified by the designer; corresponds to the technology model including the details related to the programming languages, I/O devices, or other technology.
- Sub-contractor (detailed implementation) – the construction of IT system's components; corresponds to the detailed specification given to the programmers who are not concerned with the overall structure of the system.
- Functioning enterprise – actual data, processes, departments, employees, IT systems, applications, etc. of the organization; corresponds to the instances of the concepts that are separated in one row whereas the other five contain the abstractions.

Each row represents a distinct, unique perspective. All cell models of one row build a complete model for that perspective. For each view, it is possible to ask several questions to analyze different aspects of the organization from certain perspective. These questions, shown in the columns of the matrix, are:

- What entities are involved? - data and information relevant to the perspective.
- How are they processed? - processes relevant to the perspective.
- Where are they located? – networks (from organizations to the communication networks) relevant to the perspective.
- Who works with the system? – people relevant to the perspective.
- When does event occur? – time information relevant to the perspective.
- Why are these activities taking place? – motivational aspects relevant to the perspective.

In addition to the main matrix, Zachman proposes to fill the intra-row matrices as well. They document the relation between different cells in a row (data-to-process, process-to-network, etc.). They are important in the design process for checking the dependence between different aspects, such as data, process, network, etc. In [25] it is described how additional inter-row matrices can be used to relate the corresponding elements of different perspectives. These relations are important for refining one perspective to the other and are not captured in Zachman framework originally. It provides the relations between different elements of one perspective, but not the relation between the corresponding elements of different perspectives.

There are several proposals of processes for using the Zachman framework. However, only a few deal with the conceptualization of the Zachman framework, such as [25].

Object-Process Methodology (OPM) [26] is a holistic approach for study and development of information systems. It integrates the object-oriented and process-oriented paradigms, putting the two main aspects of the system: structure and behavior of equal

footing. The main elements of OPM ontology are entities (stateful objects and processes) and links (structural and procedural). Objects are things that exist. Processes are things that transform objects. Structural links express static relation between entities. Procedural links express the connection between entities that describe the behavior of the system.

OPM model is a set of inter-related Object-Process Diagrams (OPDs). Each set of OPDs has its semantically equivalent English-like textual description represented using Object-Process Language (OPL), constrained subset of English readable by both human and machine. OPM has only one diagram type, showing both the structure and the behavior (objects with states and processes). It does not deal with multiplicity of objects and processes.

To deal with complexity, OPM offers three refinement/abstraction mechanisms [27]:

- Unfolding/folding – for refining/abstracting the structural hierarchy of the thing
- In-zooming/out-zooming – exposes/hides the inner details of a thing
- State expressing/suppressing – exposes/hides the states of an object.

Using these mechanisms, it is possible to design the system on any level of details without losing comprehension of resulting specification.

Using OPCAT [27], the tool for modeling OPM, it is possible to generate the code in the given target language (such as Java) from the formal model description represented with OPL. It is also possible to animate the model showing the existing objects, generated objects, currently performed process, etc. In addition, it is possible to generate UML diagrams and documentation from the OPM specification.

Business Process Modeling Notation (BPMN) [10], an OMG business process modeling standard, provides a graphical notation for specifying business processes. Its goal is to support business process modeling by providing a standard notation that is comprehensible to business users yet represents complex process semantics for technical users. BPMN notation is mostly focused on the representation of a system's behavior and proposes a variety of model elements for the behavior specification. It models the processes as predefined sequence of activities. This is effective for predefined, fully specified, repeatable business processes.

BPMN specifies one diagram type called business process diagram (BPD). In a BPD, two hierarchies can be captured: by using combinations of swim lanes, a hierarchical structure of organizations can be modeled; and by using combinations of BPMN processes, sub-processes, and tasks, organization behavior can be modeled with different levels of details. The former corresponds to the organizational refinement, the latter to the functional refinement. Traceability between tasks and activities in BPD is explicit and maintained by the sequence and message flows (connections).

There are many tools for modeling using BPMN and executing BPDs. The full list of tools is available in <http://www.bpmn.org/>. Currently, there are 74 BPMN implementers, showing that BPMN is becoming the de-facto standard for business process modeling.

Van der Aalst did a lot of work in the field of declarative process modeling. He has introduced case handling – new paradigm for supporting flexible business processes [28]. He illustrates case handling with the use of FLOWer software tool, developed by Pallas Athena,

which fully supports this paradigm [29]. He also proposes a ConDec [30] language for declarative, flexible modeling of business processes based on temporal logic. ConDec can also be executed as described in [31]. All this work together with the work and interest of other researchers and practitioners in declarative process modeling has led to the creation of another OMG standard: CMMN.

Case Management Model and Notation (CMMN) [32] is another OMG specification for modeling activities, whose first version has been published in 2014. It uses the notation whose goal is to be comprehensible by both business and technical users. It models the activities that are not predefined and repeatable, but depends on the current situation and the ad hoc decisions by the knowledge workers regarding a particular situation, a case [32]. Some examples where this kind of business process modeling is effective are: patient care and medical diagnosis, application and claim processes in insurance, maintenance and repair of machines and equipment, etc.

CMMN provides the notation for depiction of behavioral elements of one case, such as Stage (episodes of one case), Tasks, Milestones and Event Listeners. It does not provide visual modeling of the information model elements of the case [32]. Only information model elements involved in the behavior of the case are in CaseFileItems.

CMMN has one diagram type modeling the case. Case instance is composed of information represented by caseFileModel, behavior represented by casePlanModel and roles corresponding to the humans expected to participate in the case represented by caseRoles. The diagram contains the case with possibly many stages (fragments of case logic), tasks, milestones and event listeners. Stages can be expanded and contain other stages, tasks, milestones, and event listeners, similar to sub-process expansion in BPMN. This corresponds to the functional refinement. Assignment of roles to participants, such as individuals or teams, is not included in the scope of CMMN [32]. Therefore, there is no organizational hierarchy included in CMMN.

There are several tools for case management modeling following different standards. Some of them are listed in <http://www.businessprocessincubator.com/tools/case-management.html>. Currently, there are 27 of them. Recently, the vendors started also merging business process modeling with case management modeling, such as IBM BPM tool in the newest version 8.5.5 from June 2014.

Design & Engineering Methodology for Organizations (DEMO) [33] is a methodology for modeling, (re)designing and (re)engineering organizations. It provides a theory about construction and operation of organization. It proposes set of elements for modeling business processes and information systems based on four main concepts: communication, information, action and organization. DEMO shows organizations in two different ways: as black-box, showing input and output variables and the relation between them (called transfer function); and as white-box, replacing the system with a structure of subsystems which transfer functions are more understandable (called functional decomposition). System can also be decomposed with constructional decomposition into several subsystems not necessarily related to their transfer functions, but to the subsystems that are components of the bigger system.

DEMO defines organization as a set of three aspect-organizations: the B-organization (business), the I-organization (information) and the D-organization (document). I-organization supports B-organization and D-organization supports I-organization.

DEMO has five diagram types describing these layers: actor - transaction diagram, actor - bank diagram, process - structure diagram, objects - fact diagram, action - rule specification. The construction model specifies the construction of the organization in terms of transactions, actors, information banks, and information links between them. The process model and the state model are considered as the next detailing level of the construction model – they describe each transaction as a set of states and transitions. The action model specifies the action rules and can be seen as the second detailing level of the construction model. Traceability between modeled aspects is captured in DEMO using cross-model tables.

Service blueprinting is a customer-focused approach for service innovation and service improvement. The approach was first described by Linn Shostack in the Harvard Business review in 1982 [34]. Since then, the approach has evolved significantly to the customer-focused approach for visualizing the service processes, points of customer contact and underlying support processes in organization that drive and support customer-focused service execution. It is used for service innovation, quality improvement, customer experience design, and a strategic change with focus on customers. It depicts services on multiple levels of analysis. This way, it facilitates creation of integrated view of the entire service process and creates a common ground to different members of organization. Employees and internal units can relate better to what their contribution to the integrated service system is. It also enforces customer-orientation among members of organization.

Typical service blueprint has five components [35]:

- Customer Actions – all the steps that customer takes in the process of service delivery. They are the central part of service blueprint.
- Onstage/Visible Contact Employee Actions – actions that are performed by onstage contact employee in face-to-face contact with the customer; they are separated from the customer by the line of interaction.
- Backstage/Invisible Contact Employee Actions – actions performed by backstage contact employee in face-to-face contact with the customer (such as telephone calls); they are separated from onstage actions by the line of visibility.
- Support Processes - actions of employees and units who are not contact employees required for the service to be delivered; they are separated from contact employees by the internal line of interaction.
- Physical Evidence – tangibles that the customer comes in contact with that could influence his perception of the quality of service.

The process of building a service blueprint always starts by identifying the service process, customer segment for which the service is modeled and the goal of blueprinting (designing a desired service process for the new service, analysis of how the service is being offered for the existing service). Once the service is captured from customer's perspective, necessary actions of onstage and backstage contact employees are modeled. Next, these

actions are mapped to the required support processes. Finally, the physical evidences for every customer action step are modeled.

Some of the main characteristics and benefits of service blueprinting are:

- It is customer-focused.
- It provides a common platform for customers, employees and managers.
- It provides a common point of discussion for new service development or service improvement.
- It gives employees an overview of the entire service process.
- It puts everyone involved in service design on the same page, improving the communication and precision in the initial phases of service design.
- Can be used for any level of analysis, from micro-processes to macro-processes.
- Easy to share and update.
- Facilitates the comparison of actual and desired service and comparison with the competitor's processes.
- Forces people to take cross-disciplinary and cross-functional view of a service. Technology, supply chain and other people can see how their work contributes to the customer's experience. Management staff can see underlying technology, employee interactions and details needed to create customer's experience.
- It needs to be updated whenever there is a change in the service process.

There are several other methods for service design or application of other modeling methods to the services [36], [37]. We have explained the details of the most used service design method. To the best of our knowledge, there is no service design method that can simulate the models and provide the prototypes of the service behavior that could be used to promptly evolve the specification in response to a feedback received from a business specialist.

There are many other modeling methods and languages that we do not show in the comparison table. We give brief overview of these methods here. e3Value [38] provides an ontology to conceptualize and visualize eBusiness idea and to be able to do an analysis and profitability assessment of the eBusiness model for all parties involved. The i* framework [39] focuses on modeling properties such as goals, beliefs, abilities, commitments; and on modeling strategic relationships. Enterprise Knowledge Development (EKD) [40] is a multi-model, participatory enterprise modeling approach that involves a model for conceptual structures, and interlinked sub-models for goals, actors, business rules, business processes and requirements to be stated. Business Motivation Model (BMM) [41] models several concepts from goals, down to processes and technologies.

Table II compares the main modeling methods based on a set of criteria that are relevant for this work. As we compare modeling methods from different domains, we define first several general criteria for distinguishing the main differences, such as which development phase it is used in (early, late); is it focused on an integrated view of the organization or on details; is it service focused, meaning two things: is it customer-oriented, i.e. does it start the analysis from the customer's needs and is the internal organization defined based on the

participation of people and organizations in the service design and implementation, instead of on traditional hierarchical organization. We also categorize the methods based on the fact whether they are IT focused. By this, we mean that the method is mainly intended for modeling IT systems needed for the business, i.e. it is usually not used for high-level analysis of business cases, but is focused on details of an IT system. This also changes the spirit of the method. Once we categorize the methods, we define if they have support for model verification, refinement verification and simulation of the models. Finally, we describe if the methods use declarative or imperative process modeling and if the static part and behavior are modeled in the same diagram. The criteria for which we were not able to find more information in documentation to confirm our assumptions, we add “?” in the corresponding table field.

We present in more details explanations of model verification, simulation and refinement verification in the next sub-chapter, where we explain how formal methods and code generation techniques are being applied for model verification, refinement verification and model simulation.

From the table we can conclude that simulation, model and refinement verification are applied mostly for the methods that are IT focused, show detailed view, and are used in late design phase. There are not many methods in which it is possible to simulate high-level models. SysML and OPM are the only ones providing that possibility. However, SysML is intended mostly for the specialized domains, such as aerospace, automotive, health care, etc. OPM can be applied to modeling organizations in general, but it is not intended for services, i.e. it does not start with modeling customer’s needs and does not include service template in which it is proposed how to identify internal structure based on the services that need to be delivered.

Table II: Comparing modeling methods focused on behavior modeling

	Early/late requirements	Integrated/ detailed view	Service focused	IT focused	Model verification	Simulation	Refinement verification	Imperative/ declarative process	Static and behavior in 1 diagram
Catalysis	Early	Integrated	No	Yes	No?	No	No?	Both?	Yes
UML	Late	Detailed	No	Yes	Yes	Yes	Yes	Imperative	No
SysML	Late	Detailed	No	No	Yes	Yes	No?	Imperative	No
Archimate	Early	Integrated	No?	No	No	No	No?	Imperative	Yes
Zachman	Early	Integrated	No	No	No	No	No	/	/
OPM	Early	Integrated	No	No	Yes	Yes	No?	Imperative	Yes
BPMN	Late	Detailed	No	Yes	Yes?	Yes	No?	Imperative	No?
CMMN	Late	Detailed	No	Yes	Yes?	Yes	No?	Declarative	No?
ConDec	Late	Detailed	No	Yes	No?	Yes	No?	Declarative	No
DEMO	Late	Integrated	No	Yes	Yes?	Yes?	No?	Imperative	Yes?
Service blueprinting	Early	Integrated	Yes	No	No	No	No	Imperative	No?

2.2 Use of Formal Methods and Code Generation Techniques in Modeling

We categorize application of formal methods and code generation techniques in modeling in: application for model verification, application for model simulation and application for refinement verification. Code generation techniques are used only for the model simulation. We first give general background and then list different applications in the domain of modeling. Finally, we describe application of Alloy that is used in our method in different domains and how it is used in our method.

Formal methods are mathematically rigorous techniques for specification, design and verification of systems. They provide a means to examine the entire state space of a system design and establish a correctness of safety property that is true for all possible inputs [42]. This is achieved by use of model checking tools, such as Alloy [43], [44], NuSMV [45] or theorem provers, such as Isabelle [46], [47]. Model checking is used to build a finite state system and check satisfiability of a formula in the given domain. It is possible to detect if the formula does not hold. However, if it holds, it means that it holds in the given domain, i.e. in that finite state space, which is not a guarantee that it holds in general. Theorem provers describe a system with a set of mathematical formulas, and use theorems to prove mechanically that certain properties hold.

Application of formal methods to hardware and software system design can contribute to reliability and robustness of design. Therefore, they are mainly applied to specification and verification of safety-critical systems, in which the rigorous quality is extremely important. For example, they are used to develop software for nuclear power plant's reactor protection system [48], the traffic alert and collision avoidance system [49], the safety automatism for the various railway systems installed by Alstom and Siemens [50], etc. In addition to the safety-critical systems development, formal methods are used in different areas, including routers, Ethernet switches, routing protocols, security protocols, for functional verification of microprocessors, for development of processors, by companies such as Intel, AMD, IBM and many others.

Despite the fact that the formal methods are very powerful in providing formal specification that can be rigorously validated and verified, there were hesitations on applying them to domains different than safety-critical systems. Some practitioners believe that the role of full formalization has been overemphasized [51], [52]. In practice, the modeled systems are too complex, the notation appears complex, and the output is too difficult for domain expert to understand and to extract meaningful information [48]. Therefore, benefits of the formal methods are often gained at a heavy price. This is especially true for the specification of requirements and high-level designs whose purpose is to be simple and easily understandable, so overformalizing them can be harmful. Hence, many lightweight formal methods that

emphasize partial specification and focused application have been proposed. One of the most well-known lightweight approaches to formal methods is Alloy [43], [44], the language that is used in this thesis as well. With lightweight formal methods, application of formal methods to high-level modeling became more common [53], [54].

High-level modeling methods provide different levels of precision. They usually provide rich representation that non-technical stakeholders find appealing, but are often difficult to check automatically. To achieve automated analysis of models, formal methods can be useful. However, the nature of the early-phase analysis suggests that formality should be used carefully. Formal methods can be difficult to construct and overformalizing can be harmful [55]. When used for higher-level design, formal methods are used for: model formalization (formal specification), analysis and verification; model simulation and validation; proving the refinement between models. We give examples of each of these uses. In model simulation section, we also give the examples of code generation techniques used to simulate and execute the models. Finally, we explain Alloy language that we use in this thesis and its application in different domains.

2.2.1 Model Formalization, Analysis and Verification

UML includes a formal constraint notation Object Constraint Language (OCL) [56]. It is precise text language for defining constraints and object query expressions on any Meta-Object Facility (MOF) [57] model or meta-model, including UML. It abstracts the formal mathematical expressions, making the language understandable to practitioners. It enables the designer to express additional constraints on the model that cannot be expressed using the diagram notation, making it more precise. It can be used to express constraints on variable values within the model and since UML2.0 it also provides fully defined action notation. However, it cannot be used to analyze the model and prove the correctness of certain properties. Also, OCL has been criticized by some formal methods users for being cumbersome compared to traditional set based modeling notations [58]. Therefore, there are a lot of approaches for formalizing UML for the purpose of analysis, which propose to use Z, B, Object-Z or Alloy as underlying semantics for UML or for analysis of UML models. One of them is UML-B.

The UML-B [59] is a profile of the UML, which defines a subset and specialization of UML that can be translated into B [60] language. It is created to provide more precise semantics to UML models, so that it can be used for modeling critical systems as well. It enables modeling of various types of problems at different levels of abstraction. It enables verification of UML-B models, as well as verification of the refinement relations between models at different level of abstraction.

It enriches UML notation with use of stereotypes to specialize the meaning of UML entities and use of details, such as invariants and guards as constraints of operations. UML-B provides diagrammatic formal modeling notation. It hides the

complexity of B and packages mathematical constraints and action specifications into small sections in the context of its owning UML entity. This way, for the modeling information that cannot be expressed diagrammatically it uses UML-B clauses, which correspond to the packaged B clauses, as tagged values that can be attached to relevant entities.

It provides the tool to automatically translate UML-B model to the B specification. The B specification can then be used to verify the correctness of UML-B model and detect errors at early stages of design. UML-B contributes to UML with the notion of a refinement relation between abstract and concrete models and the ability to verify this relationship, based on B refinement.

Another approach to UML formalization [61] uses Z notation as a supplement to UML models to express syntax and semantics of the models. The authors chose to use Z instead of OCL, because it provides better facilities for proofs. Their goal is to make UML models more precise and analyzable. In order to prove a property for a given UML model, the authors apply a set of transformation rules until the desired conclusion is reached. Then they use Z representation to prove that whenever the transformation is applied to the diagram, the resulting diagram is a valid deduction of the original diagram.

The authors of [62] describe the translation of UML specifications with OCL constraints into B. The authors of [63] describe the mapping of class diagram features into B machines. They combine UML with B method to provide formal specification of database applications, which are data-centric [64]. UML2Alloy [65] is a tool that transforms UML class diagrams with OCL constraints to Alloy specification, which can be automatically analyzed by Alloy Analyzer tool [13] to identify the design faults in specification. It is also possible to transform the fault instances back to UML showing the examples that illustrate the identified faults in specification.

Another category of tools for analysis of UML models relies on theorem provers. The KeY tool [66] formalizes OCL and enables analysis of models with an interactive theorem prover. HOL-OCL [67] is another tool that transforms OCL to Higher Order Logic (HOL) formulas that can be analyzed by the Isabelle [46], [47] theorem prover. All these methods require guidance and special expertise to operate the theorem prover environment.

Authors of [68] have extended SysML by introduction of basic concepts of goal model from KAOS method. Model created with extended SysML can be translated to B. Using B specification and the tools, it can be verified if the model satisfies all the requirements.

Formal Tropos [69] is a part of the Tropos project to develop an agent-oriented software development methodology for specifying early and late requirements using actors, goals and the dependencies between them. It is based on i*

and extended with a temporal specification language inspired by KAOS [70] to provide a precise description of dynamic aspects. Once the model is created in Eclipse, it is translated to the intermediate language from which the finite state model is built in NuSMV. NuSMV checks the generated model and shows counter-examples and errors in graphical form. It checks if the specification is self-contradictory, that the types in the model are correct, that the requirements are not overspecified or underspecified and that all necessary states can be reached from initial state. This approach is focused on the agent properties such as goals, beliefs and abilities.

The Stimulus Response Requirements Specification (SRRS) notation [71] is the formalized version of the Thread Capability (TC) notation developed in-house by Raytheon of Canada for specifying their complex commercial and military air traffic control systems. TC and SRRS are designed for the specification of large, software intensive systems with complex data requirements [72]. SRRS extends TC notation so that it can be analyzed by the tools, which helps modelers develop and verify specification. All the required pieces of information are entered into the templates, that are then translated to S [73], the mathematical notation based on Z [74]. A dedicated validator of the type checker tool is used to find hundreds of different kinds of semantical errors present in S specification.

KAOS [70] is a software development methodology that supports the whole process of requirements elaboration – from high-level goals that should be achieved by the composite system to the operations, objects and constraints to be implemented by the software part of it. It provides a multi-paradigm specification language and a goal-directed elaboration method. The language combines semantic nets for the conceptual modeling of goals, constraints, agents, objects and operations in the system; temporal logic for the specification of goals, constraints and objects; and state-based specification for the specification of operations [75]. Each construct in the KAOS language has an outer semantic net layer for declaring a concept, its attributes and its links to other concepts; and inner formal assertion layer for formally defining the concept. Formal assertions are written in a real-time temporal logic. Formal representation enables one to create precise specification and to precisely refine the goals into sub-goals, as explained in the refinement section.

2.2.2 Model Simulation

USE tool (UML Specification Environment) [76] provides a way to animate UML models and generate instances that can be used to evaluate the model. It generates the snapshots that correspond to the rules expressed in the model. It provides a way to check if a specific instance of the model conforms to the constraints.

The authors of [77] plan to use formal methods for simulation of business transactions in the existing method and toolset “Efficient” [78], which will enable

business experts to validate the defined business services and models through an animator tool. The toolset will allow business experts to “play” the model as if those were already implemented.

Code generation techniques are also used to simulate and execute different models. BPMN specification includes mapping to the Business Process Execution Language (BPEL). Therefore, BPMN models have their corresponding BPEL representation and can be executed generating the application using the user forms and roles in the BPMN model corresponding to the poles and lanes. A lot of tools implementing BPMN and CMMN have option to execute the models represented with these two notations.

OPM can also execute its models using the tool OPCAT [27], and generate the code in the given target language (such as Java) from the formal model description represented with OPL. It is also possible to animate the model showing the existing objects, generated objects, currently performed process, etc.

Domain specific modeling uses models as a source for code generation. There are a lot of environments for creating and using domain-specific modeling languages: commercial tools MetaEdit+ and Actifsource, open source tools such as GEMS, academic such as GME. Domain-specific languages frameworks have also been added to the existing IDEs, such as Eclipse and Microsoft’s DSL Tools. They all provide a way to create domain-specific modeling language and to use it for code generation.

2.2.3 Refinement

In addition to the analysis of a model, formal methods have been applied to prove the refinement between different models. Authors of [79] propose a way to validate the refinement between the state machines using pi-calculus. They propose six kinds of UML state diagram’s assembly mechanisms, with the corresponding pi-calculus formal semantics. They use pi-calculus semantics to validate if the UML state diagrams are observation equivalent.

As we have mentioned, UML-B is also using the corresponding B specification to verify the relation between abstract and concrete models based on B refinement.

One of the steps of KAOS methodology is to refine goals into sub-goals. Goal decomposition done by hand is usually incomplete and sometimes inconsistent. Also, interesting alternatives may be overlooked. In order to provide formal support for building goal refinements that are complete, proved correct and integrate alternatives, the authors of [75] propose usage of a set of refinement patterns. These patterns are generic and propose different ways to decompose the goal into sub-goals. Once they are proved as correct and complete, they can be used to refine the goals, by keeping

the model correct and complete without a need to show tedious mathematics involved in proofs. They are proved to be correct by use of formal representation of the concepts in KAOS.

2.2.4 Alloy Applications

Our approach is based on Alloy, a lightweight formal specification language developed at MIT. The area of Alloy application is very large: it has been used for model analysis [65], verification [4], [80], constraint checking [81], automatic model completion [82], enterprise modeling [83], software architecture [84], service testing [85], and others.

To the best of our knowledge, all current Alloy applications in the domain of service design are targeting software architects, developers and other technical specialists. The examples include measuring QoS of a composite orchestration [85], [86], verification and specification of IT services [4], [85], and others.

As we can conclude, formal methods provide a good way to create precise models by formalizing otherwise soft representations used in high-level designs, such as for business and service models. However, overformalization can be harmful and difficult to apply in practice. Therefore, different modeling methods have applied different approaches to solve this problem: by using formal languages as a supplement to the diagrams, by abstracting elements of formal languages, by applying lightweight formal methods, and others. As a result, formal methods have been used a lot in modeling for model verification, model simulation and refinement verification. All of them are mostly applied for the methods that are IT focused, such as UML. They are not applied for the business models on high-level of analysis.

To sum up, model verification and simulation are applied mostly on the IT focused modeling methods or for a specific domain. We propose a method that provides a way to create precise models on high-level of analysis, which can be verified and simulated. In addition, our method is also service focused, meaning we always start by modeling customer's needs and provide a service template that includes organizing people, technology and organizations based on the service that should be provided.

3 Knowledge Base

In this chapter, we explain the knowledge base including theories, methodologies, and languages forming the base for this work. We first explain SEAM, the method upon which the proposed method is based. Its theory is based in systemic modeling paradigm. Therefore, we explain systemic modeling paradigm and SEAM.

The main service specification of BASS is based on model transformations. They are also used to simulate and prototype the models. Therefore, we explain the model transformation and what categories of transformations are used in BASS.

BASS uses formal methods with model verification and code generation techniques to animate the models. Therefore, we explain model verification and the Alloy language that is used to simulate the models. Alloy enables to use formal methods and model checking, as well as to visually show the instances of the model. Therefore, it is useful to create precise specification that can be validated by the business and IT analysts, thanks to the visual representation of the results. Also, as a declarative language it is useful to discover missing business rules in the model.

Finally, we explain code generation and Arcimboldo approach, used to prototype the models. As it is based on templates, it enables adding another level of abstraction before going directly to code.

3.1 Systemic Modeling Paradigm and SEAM

The service specification method we propose is based on SEAM, a modeling technique developed at EPFL. SEAM is a method that is based on General Systems Thinking (GST).

Banathy and Jenlink [87], seeking to provide a comprehensive description of GST, explain it as the interlinked association of three domains of inquiry: systems theory, systems philosophy (which further contains epistemology, ontology and axiology) and systems methodology. They call this set Systems Inquiry.

The systemic modeling paradigm was proposed by Wegmann in [1]. It combines Systems Inquiry and Kühn's notion of paradigm change. A paradigm is defined as "a philosophical and theoretical framework of a scientific school or discipline within which theories, laws, and generalizations and the experiments performed in support of them are formulated" [88]. The systemic modeling paradigm also extends Systems Inquiry with discipline specific theories.

Systems Theory

Systems theory, as described by Banathy and Jenlink [87] espouses the view that modern science and industry have locked themselves in a pursuit of an "ever-increasing specialization." This specialization results in specialists' inability, and often unwillingness to engage with, or even understand, other specialists.

The early system thinkers have observed that as each specialized discipline creates its own specialized vocabulary, it nevertheless uses concepts that are similar to other disciplines. It is often the vocabulary that is different but the underlying

principles are the same. The same phenomena studied by a biologist can be observed in enterprises, for example. GST was therefore designed as a *lingua franca* that would enable specialists from different disciplines to collaborate (e.g. a biologist with an economist) and understand each other. GST seeks to define general principles that can be applied to any phenomena across established disciplines, thereby complementing the specialist's view.

SEAM is a method built on a systemic grounding. Much like GST is interested in federating scientific disciplines, when intervening in organizations, there is a need to understand and transcend the specialist's view of the stakeholders (often called "silos" today) that compose the organization. While doing so, the designer should be careful not to alter too much the stakeholders' way of working because their effective action depends on them remaining specialists.

Systems Philosophy

As noted by Banathy and Jenlink [87], interest of GST with general principles that transcend disciplines implies a close link with philosophy. They define systems' philosophy as consisting of three components: Ontology, Epistemology and Axiology (Ethics). Ontology describes what things are, e.g. what a person is, what an organization is, what a society is. Epistemology is oriented towards questioning of ontology, e.g. how we know what a person, an organization, or a society is? Banathy and Jenlink contend that these two aspects are intimately linked because it is often impossible to completely separate what we know from how we know it. Finally, axiology is concerned with the notions of value, ethics and aesthetics. It underlines the choices made by systems thinkers when they select some aspects of reality for attention, rather than others. Are these choices good, bad, beautiful, ugly, moral or not that constitutes the questions that axiology aims to reply to.

Parting from Banathy and Jenlink's explanation we explain the SEAM philosophy starting from epistemology rather than ontology.

The *SEAM epistemology* is interpretative [89] or interpretive [90]. This means that we believe that each stakeholder creates his specialized knowledge of his work by interacting with the work artifacts and through his relationships with other specialists in his domain. We call universe of discourse this set of entities that the stakeholder sees, which is a subset of the total number of entities available in reality. Each stakeholder builds a set of concepts, which we call his conceptualization, by interacting with his universe of discourse. This conceptualization is the basis of his understanding of the world.

Other terms that convey a similar meaning to the universe of discourse and conceptualization can be found in Vickers's appreciative system [91], [92]. Vickers explains that people and organizations develop readiness to see some aspects of reality. This readiness is necessary for effective action, but is also a barrier to collaboration with others because it makes it difficult to see things from a distinct point of view.

What we call the *SEAM ontology*, in-line with the standard use of the term ontology in computer and information sciences, are the model elements with which an

enterprise architect describes the stakeholders' conceptualizations and the shared model that the stakeholders should agree about.

In the SEAM ontology we use the term *working object* to designate a system in the conceptualization. For example, a working object named “EPFL School” in the model maps to a system that the modeler understands as being a school in conceptualization. The name EPFL helps mapping to the specific school “EPFL” in the universe of discourse. This explains how the model element in the model relates to entities in the universe of discourse.

The ontology in the form of the working object allows benefitting from the domain specific theories proper to SEAM (e.g., refinement, model checking). A working object refers to a service system [92], [93] in the sense that it shows the way value is co-created rather than an organizational entity, such as a company. The working object “EPFL School” may therefore contain other working objects that map to organizations that most stakeholders will think of as external to EPFL, for example, an IT supplier. Having the “IT supplier” working object within the “EPFL School” working object shows that the service provided by the EPFL School includes the service provided by the IT supplier.

The *SEAM axiology* refers to the choices the specialists make about what to include in their model. These choices can have two aspects: aesthetics and ethics. Aesthetics include practicality and simplicity. The modeler needs to decide to model what is useful and practical in order to show the problems and possible solutions. The goal is not to make an exhaustive universal list of what exists in a company, but rather to analyze a concrete challenge. The modeler needs also to find a way to attain simplicity. The modeler should use the abstraction mechanisms of SEAM to illustrate the situation concisely. Even if it is concise, the model should keep the important systemic model elements (such as service system boundaries in the “to-be” model), so that the stakeholder can understand what is represented. Ethics – the model also captures the ethical choices of the modeled enterprise. For example, is the shareholder the primary “customer” of the company or should it be the “normal” customer. Axiology is useful to explain these two kinds of choices. It is associated with heuristics, such as, for example, that it is usually beneficial to first understand the “real” customer rather than the shareholder.

Systems Methodology

Systems methodology is the study and creation of methods for intervention. Banathy and Jenlink [87] divide systems methodology into two domains of inquiry: the study of methods (their creation and improvement) and the practical use of these methods. The methods are used for the analysis of systems and systems problems, the design, development and implementation of systems and the management of systems in general. The method depends on the problem context and content as well as the type of systems in which the problem is situated. A specific methodology needs to be chosen from the wide range of available frameworks using a solid justification and analysis of the investigated problem.

The SEAM methodology prescribes the way a designer uses the SEAM theory and philosophy to produce results. The methodology is a collection of techniques, some of which are well-known to enterprise architects (such as the as-is and to-be modeling). Others were imported from other disciplines, e.g. contextual inquiry [94]. Because it is often costly and time consuming to do contextual inquiry in practice, we use an alternative technique of using concrete names of people and organizations (e.g., EPFL School rather than simply School) as well as anecdotes in workshops. This helps stakeholders to remember the context they were in when facing some problems. Without this context, they may often forget to give many details about their work. A related technique encouraged in SEAM is to collect supporting evidence about concrete situations in the form of e.g., pictures, letters, and emails. We also recommend developing a model bottom-up and top-down at the same time. We obtain the best results when the modeling sessions are short and iterative.

The first paper about SEAM was published in 2003. SEAM is used in consulting for business-IT alignment [95], [96], enterprise architecture [97] and business strategy [98]. It is used for teaching enterprise architecture, service-oriented architecture, requirements engineering [99], [100] and business strategy for IT services.

SEAM provides a consistent set of modeling principles and constructs to model an enterprise at different abstraction levels both from organizational (company, department, IT infrastructure, etc.) and behavioral standpoints (services and processes at different granularity) [1].

SEAM is inspired by the Catalysis approach [15] and by the Unified Modeling Language (UML) [8], [101]. The Catalysis approach is a component-oriented development method based on the two main concepts: object and action. Some of the main principles overtaken from Catalysis and adapted in SEAM are: hierarchical model structure, functional and organizational refinement, localized and distributed action [15]. SEAM proposes a concrete implementation of Catalysis and extends Catalysis to business modeling. Moreover, SEAM notation takes the UML notation and proposes one kind of diagram that includes a subset of the element kinds found in the 14 UML diagrams (UML 2.4.1). A SEAM diagram is a combination of the UML deployment diagram, use case diagram and class diagram.

SEAM is used in early design phases for early requirements gathering. It can be seen as a method for a system pre-design and used as a complement to the UML or the Business Process Modeling Notation (BPMN) languages. Its purpose is not to show the low-level design, but to delimit the problem, analyze and discuss the stakeholder's viewpoints, show their objectives and the processes in which they are involved, without showing all the details. In practice, the method is used with pen, paper, post-its, flip-charts, etc.

SEAM provides three representations to model both the business and IT context:

- Goal-Belief model - Describing the motivation of the business and of the IT stakeholders;

- Behavior model - Describing the services, processes implementing the services and responsibilities of the business and of the IT stakeholders
- Supplier Adopter Relationship model – Describing the features supported by the system, and what benefits these features provide to the different entities constituting the situation being modeled.

An example of how all of them can be used during the case analysis is explained in [102]. In this thesis, we deal with the behavior model only. Therefore, we focus on explaining the theory relevant to that part of SEAM. SEAM has an explicit systemic paradigm, as explained in [1], which sets it apart from most other enterprise architecture methods.

A SEAM model represents systems that people perceive in their reality. These are represented as modeling elements that we call working objects. We can represent the externally visible behavior. For this we represent the working object as a whole (black box). The externally visible behavior is represented as a working object's service. We can also represent the systems construction (i.e. the component working objects). In this case, we show the working object as a composite (or white box, transparent box). The process of going from working object as a whole to working object as a composite is named organizational refinement, as we refine the organization structure. Similarly, we can refine the service as a whole to an service as a composite, showing its constituting sub-services. We refer to this as functional refinement, as we refine the functional part of the system (Figure 9).

The working objects in our models appear with the appropriate business term, e.g., markets, segments, value networks (group of companies), companies, departments, people, IT system, IT infrastructure, etc. We can represent any kind of systems with a working object. The same modeling rules apply regardless of the nature of the modeled system.

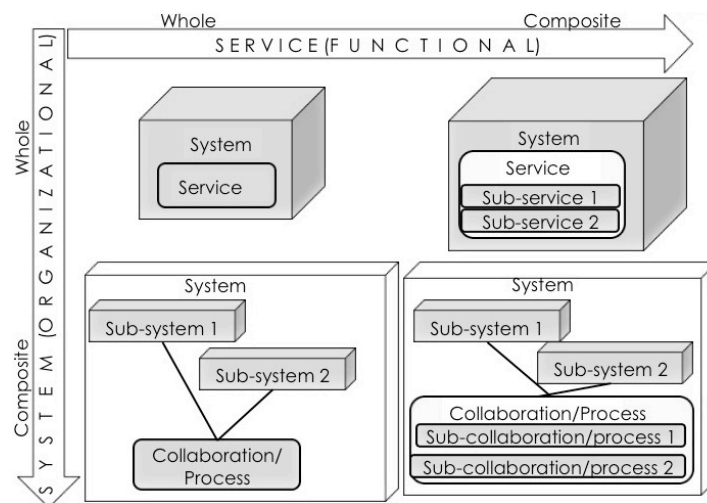


Figure 9: Functional and organizational refinement in SEAM

3.2 Model Transformation

A model transformation, in Model-Driven Engineering (MDE) is used to save effort and reduce errors in a model evolution from abstract specifications to concrete specifications and potentially to their implementations. By [103] model transformation is defined as: “A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.” The transformation process is illustrated in Figure 10.

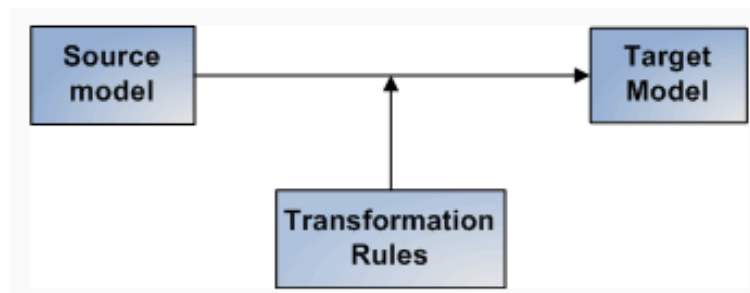


Figure 10: Model transformation process (from [104])

Based on the language of the source and target model and on their abstraction level, [105] the following categories of transformations are proposed: endogenous/exogenous and horizontal/vertical.

A transformation is endogenous if the source and the target models conform to the same meta-model (are expressed in the same language). Endogenous transformations are also called rephrasing. A transformation is exogenous if the source and the target models conform to different meta-models (are expressed in different languages). Exogenous transformations are also called translations from one language into another.

Typical examples of translation are:

- Synthesis of a higher-level, more abstract, specification (e.g., design model) into a lower-level, more concrete one (e.g., a model of a Java program).
- Reverse engineering – extracting a higher-level specification from a lower-level one.
- Migration from a model (visual specification or program) written in one language to another, by keeping the same level of abstraction.

Some examples of rephrasing are:

- Optimization – improving certain qualities, while preserving the semantics
- Refactoring – change of internal structure, while keeping the same level of abstraction.

A transformation is horizontal if the source and the target model reside at the same abstraction level. A transformation is vertical if the source and the target model reside in different abstraction levels.

Typical examples of horizontal transformations are refactoring and migration. Typical example of vertical transformation is refinement, where a specification is gradually refined.

In context of visual modeling, we distinguish the transformations of visual specifications to executable program specifications (exogenous), and the transformations of visual specifications to visual specifications. Latter transformations can be endogenous (if both models are expressed in the same visual modeling language) or exogenous (if a language of the target model is different from the language of the source model). [106]

In this thesis, we use transformation from one BASS model to another BASS model. By the given categorization, this belongs to the vertical, endogenous transformation, more precisely to refinement. We also use the simulation and prototyping of BASS models. They correspond to vertical exogenous transformations, more precisely synthesis.

3.3 Model Verification and Alloy

Model verification is the process of checking whether a certain property holds for the modeled system. This can be done by formal verification, the act of proving or disproving the correctness of a specification (in this case a model), using formal methods and mathematics. The model of a system is specified with a set of logical formulas. There are two main approaches to formal verification: model checking [107] and theorem proving based on logical inference [108].

Model checking is an approach based on exhaustive exploration of the model of the system, which can be considered as a finite state machine, with nodes representing system states, and vertices presenting transition between their states. In order to check if a model satisfies some property, the state space of the model is exhaustively explored.

The major drawback of the model checking is a state explosion problem, which originates from the fact that for real systems the size of the state space grows exponentially with the number of processes [109]. To avoid the state explosion, model checkers use different techniques, such as counterexample-based algorithms. To prove that the property of a system holds, the algorithm looks for the case, for which the property does not hold (called counterexample). In case the counterexample is found, the property does not hold. However, if there are no counterexamples, that does not prove the validity of the formula. It proves only that it is valid in the given domain.

The second approach is an automated theorem proving based on logical inference. This approach uses a set of axioms for the underlying logic and hypothesis about the system to deduct the proof for validity of the formula. [106]

Unlike model checking, this approach is not limited by the state explosion problem and if the proof is found, it implies that the formula is valid. The major drawback of this approach is that it requires the designer to understand in detail why the system works correctly, and to express this in the verification tool, by entering the sequence of theorems to be proved.

In this thesis, we use only model checking to verify and validate the models. More concretely, we use Alloy declarative language to represent the model formally and the Alloy Analyzer tool to verify the constraints and the results of simulation to validate the model with the stakeholders. Therefore, we explain Alloy in more details.

The models created with BASS can be simulated with Alloy to observe the behavior and verify and validate the models. Alloy [13] is a declarative specification language developed by the Software Design Group at MIT. Alloy is a language for expressing complex structural constraints and behavior based on first-order logic.

The Alloy Analyzer [43] is a tool for the automated analysis of models written in the Alloy specification language. Given a logical formula and a data structure that defines the value domain for this formula, the Alloy Analyzer decides whether this formula is satisfiable. Mechanically, the Alloy Analyzer attempts to find a model instance - a binding of variables to values - that makes the formula true.

The syntax of Alloy is similar to the syntax of OCL (the Object Constraint Language) for UML [110]. In the following lines, the Alloy keywords are marked in bold. Data structures are represented with signatures (**sig**) and fields. Alloy reusable expressions (i.e. functions) and constraints (i.e. facts, predicates and assertions) [44] can be used to reason about data structures and to define the relationships between them.

There are three types of constraints specified in Alloy: Fact (**fact**) is a model constraint that permanently holds; Predicate (**pred**) is a constraint that holds in specific context or for a specific part of the model only; Assertion (**assert**) is a property that the designer believes should be implied from the model and can check (command check) if it can be deduced from the other (permanent or contextual) constraints. Assertion can be presented in a shorter form `assertName: check`.

Logic of Alloy language combines the quantifiers of first-order logic (holds for every element (all), holds for some element (some), holds for no element (no), holds for at most one element (lone), holds for exactly one element (one)) with the arithmetic operators (+, -, =, etc.), set operators (union (+), difference (-), intersection (&), subset (in), equality (=)), relational (arrow(->), dot(.), transitive closure (), etc.) and logical operators (negation (not/!), conjunction (and/&&), disjunction (or/||), implication (implies/=>), alternative (else/), biimplication (iff/<=>)).

In Alloy, the model is written with Alloy language and can be simulated in the Alloy Analyzer tool. The result of the simulation is a set of instances that satisfy the constraints given in the model. It is shown in a graphical form.

We show an example of such a model in Figure 11 and Figure 12, found in [103]. Alloy code in Figure 11 models the example of the family relationships. The first line defines the module in which the family example is modeled. Then, the data structure is defined with signatures (**sig**) and fields. They are similar to Java classes. We define the signature name, their fields with cardinalities (**lone**-0 or 1, **one**-1, **set**-many, etc.) and types (*Man*, *Woman*, etc.). Signatures can also be abstract, meaning like in Java that it can be inherited (**sig** *Man extends Person*) and cannot be instantiated, i.e. in the resulting model we won't see instances of *Person*.

Once we have defined data structure, we add the constraints on the data structure. In this example, we use the **fact** that claims that there is no person who can be its own ancestor. We also express that husband and wife are symmetrical relation, i.e. if person A is a husband of person B, then person B is a wife of person A. We express this using quantifiers, set and relation operators. Unlike predicate, the fact applies globally on the whole model, so everywhere in the model, this constraint will hold.

In the right column, we use the **assertion** *noSelfFather* to check if some properties hold in the model. We use check command to check if this assertion is valid. As a result, we will get either counterexample showing the instances in which this assertion does not hold, or the result that the assertion is valid.

At the end, we define the function *grandpas*, which as Java method is reusable part of a code. It takes one Person as a parameter, and returns all its grandfathers. This function is used in a predicate *ownGrandpa*, which checks if one person can be its own grandfather. Unlike the fact, expression in predicate is valid only in that context, i.e. this constraint won't influence any other part of the code, unless the predicate is executed.

We run the predicate *ownGrandpa* with the command *run* and as a result get all the instances of the model that satisfy the constraints in the model.

<pre> module grandpa abstract sig Person { father: lone Man, mother: lone Woman } sig Man extends Person { wife: lone Woman } sig Woman extends Person { husband: lone Man } fact { no p: Person p in p.^(mother + father) wife = ~husband } </pre>	<pre> assert noSelfFather { no m: Man m = m.father } check noSelfFather fun grandpas[p: Person] : set Person { p.(mother + father).father } pred ownGrandpa[p: Person] { p in grandpas[p] } run ownGrandpa for 4 Person </pre>
---	--

Figure 11: Alloy model of the family example (from [111])

The result of simulation of the model in Figure 11 is shown in Figure 12. In fact, the result is actually the set of instances that satisfy the constraints in the model. Figure 12 depicts one of the instances. There are 4 persons shown in trapezoid forms, 2 women: *Woman0* and *Woman1*, and 2 men: *Man0* and *Man1*. The relation between them is shown with the names on lines connecting them. For example, *Woman1* is mother of *Woman0*. *Man0* is marked with *ownGrandpa*, meaning that it is the example of the person that is its own grandfather. As we can see, *Man1* is his father, meaning that *Woman0* is his mother and *Woman1* is his grandmother. This implies that her husband, *Man0*, is his grandfather. Therefore, *Man0* is his own grandfather. If we would like not to get instances like this, we would need to add additional constraints to our model. With this example, we have covered the main syntax of Alloy language and explained how the simulations in Alloy work.

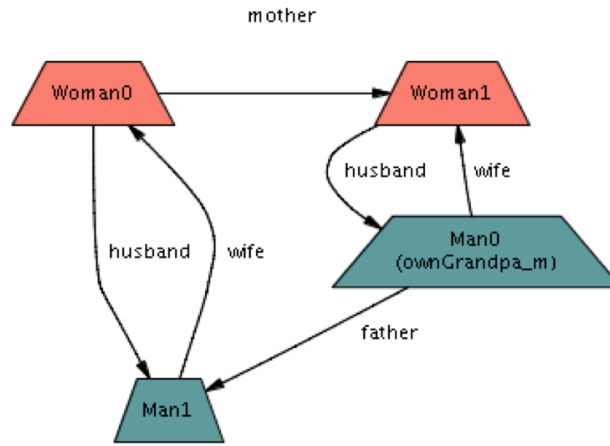


Figure 12: Result of simulation of the family example in Alloy Analyzer tool (from [111])

3.4 Code Generation and Arcimboldo

Code generation in the Model-Driven Architecture (MDA) means the user abstractly models solutions, and the automated tool derives from the model or its part, the source code. MDA was launched by OMG in 2001. Based on OMG's standards, MDA separates business and application logic from underlying platform technology. Platform-independent models (PIM) capture the business functionality and behavior of an application separately from technology that implements it. These models specify business and application logic, independent from the technologies that are used. Platform-specific model (PSM) contains the logic expressed with the concepts related to the technology used. In addition to traditional MDA tools used for code generation, in which models are mostly specified using general-purpose UML, Domain Specific Modeling (DSM) became very popular. DSM uses the modeling concepts coming directly from the application domain.

In this thesis, we use BASS models as platform-independent models to specify the services that are needed and then translate them to Arcimboldo project, containing templates in the given target language (in this case Java), corresponding to the

platform-specific model. We then use Arcimboldo framework for code generation. Therefore, we explain Arcimboldo in more details.

Arcimboldo [112] is an approach that allows developers to devise various kinds of applications with the help of templates. It includes a workbench that supports the full development process from the creation of the templates, through code generation, up to the (automatic) generation of a DSL (Domain Specific Language) that can be used to develop applications that use similar operations.

The components it manages grow bottom up with the application and provide more and more sophisticated support as the work progresses. When the project is mature enough for domain specialists to take it over, a DSL can be generated quasi-automatically. The resulting DSL can be used to extend the application, as well as to develop other applications that require similar operations. [113]

The generation of an application is based on the expansion of templates (Javascript object syntax (JSON) templates [114]). Application is created from three main types of file: the main object, templates and the descriptor file.

The main object contains JSON object with data, which when combined with the template produces the given page, class or another part of application. The syntax of this object corresponds to Javascript's. It is similar to the data dictionary in JSON templates. It enriches the standard syntax with several features, like use of predefined or user defined methods. In addition, the computations can be specified with a specific algebraic language Alog [112]. It appears in the project with extensions .jao.

The templates correspond to the JSON templates, with some particularities. They can be used to write templates for pages, classes or other parts of the application. They can be written in any target language, depending on the requirements for the application. They appear in the project with an extension .tpl.

The descriptor file contains the list of all the files that must be created, and for each file the information on which template and what part of the main object it is created from. It appears in the project with extension .fls.

Arcimboldo comes with a workbench that includes Arcimboldo editor and Arcimboldo generator (Figure 13). In addition, there is also Arcimboldo generator for DSL. Arcimboldo editor enables user to independently edit templates, the main object and description file. Combining these files, Arcimboldo generator creates the application. Once the application is mature enough, Arcimboldo DSL generator can be used to create DSL that can be used for development of similar applications.

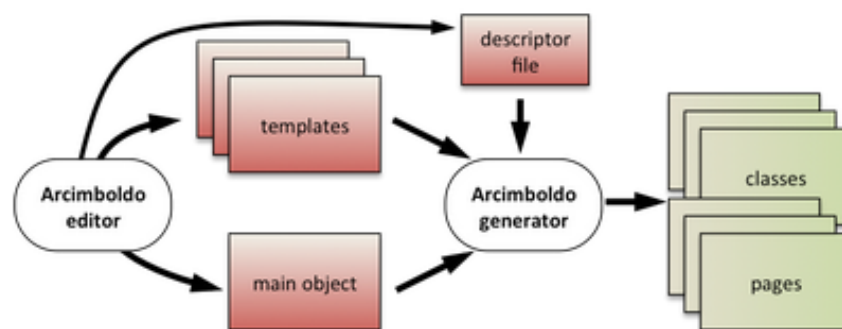


Figure 13: Arcimboldo workbench: overview (from [112])

The workbench also offers the possibility to understand where the generated parts come from. This feature extends the ability of Eclipse to open the declarations and the references of the methods, variables and classes to all types of files, by using the description object as a relay between the symbols.

We illustrate how Arcimboldo works with the example in Figure 14. On the left side, there is a template that contains an attribute `{title}`, a section `{.section subtitle}` and a repeated section, `{.repeated section list}`. Inside the section subtitle there is a category defined with the attribute `{categ_id}`. This attribute belongs to the special kind of attributes, as it ends with `_id`. This means its name can be used also as an uppercase and the element can be searched by its value when the predefined method in the main objects is used. Repeated section contains the text with the attributes `{animal}` and `{sound}`. The attributes in the template correspond to the attributes in the object, the section to a sub-object and the repeated section to an array.

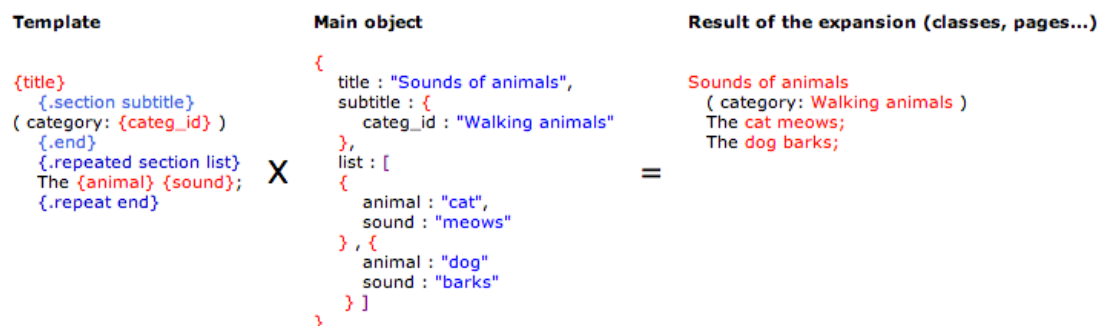


Figure 14: Arcimboldo example (from [112])

In the center of the Figure 14 is the main object from which the attribute names in the template come from. As we can see, repeated section list corresponds to the array list in the main object. Section subtitle in the template corresponds to the sub-object in the main object. The attributes in the template correspond to the attributes in the main object.

Once the template is combined with the main object, we obtain the resulting file shown on the right side of the Figure 14. When the attribute title in the template is combined with the attribute title in the main object, we get the first line: Sounds of animals. Then, section subtitle is expanded using the sub-object subtitle in the main object. Therefore, we get the line (category: Walking animals). Finally, repeated section list is combined with the array list. For each element of that array, the new line of text is obtained. So for two elements of the list, we get two lines: the cat meows; and the dog barks.

4 BASS: Concepts and Visual Formalism

In this chapter, we explain the main concepts used in the proposed method and their graphical representation. Reading this chapter is required for understanding the explanations of the method in Chapters 6, 7 and 8, as well as evaluation in Chapter 9. The proposed method is based on an existing method SEAM explained in Chapter 3.1. We first explain the main concepts and principles used in BASS. Then we explain the meta-model of the proposed method (abstract syntax) including the main concepts used in the method and their mapping to business terminology. Finally, we explain the visual formalism to model these concepts (concrete syntax) and its semantics. In this chapter, we explain the meaning of all the concepts used in BASS model. The full formal semantics of BASS is based on SEAM formal semantics explained in [106]. Additionally, we have abstracted some of the formal concepts, such as postcondition of a service using functional units, which can be represented graphically in the BASS model.

4.1 BASS: Concepts and Principles

In this chapter, we explain the main principles and concepts used to model and simulate service systems.

4.1.1 Service and Service System Modeling

SEAM as a systemic method is focused on modeling systems, which are represented with working objects. SEAM's theoretical foundations are in GST. The same modeling rules and principles apply regardless of the nature of the modeled system (IT, human, company, etc.).

With popularization of services and service science, the authors of [3] have proposed service system as an important abstraction for the service revolution. By [3], service is the application of resources (including competences, skills, and knowledge) to make changes that have value for another (system), such as customer's system. Value is improvement in a system, as judged by the system or by the system's ability to fit the environment. The service-system abstraction can be used to understand how value is created unifying the concepts from different disciplines. The service system is a configuration of people, technologies, and other resources that interact with other service systems to create mutual value [3]. Many systems can be viewed as service systems, including families, cities, and companies, among many others [3]. Therefore, service concept became a very important concept for understanding how the value is created and for unifying concepts from various disciplines.

Several efforts have been done to introduce service concepts to the existing systemic method SEAM [4], [5], [6]. In this thesis, we have extended SEAM with concepts needed to model and simulate services, such as functional unit, events, send and receive properties. Based on the best practices defined in the Information Technology Infrastructure Library (ITIL) [115], we propose to start modeling using the template shown in Figure 15.

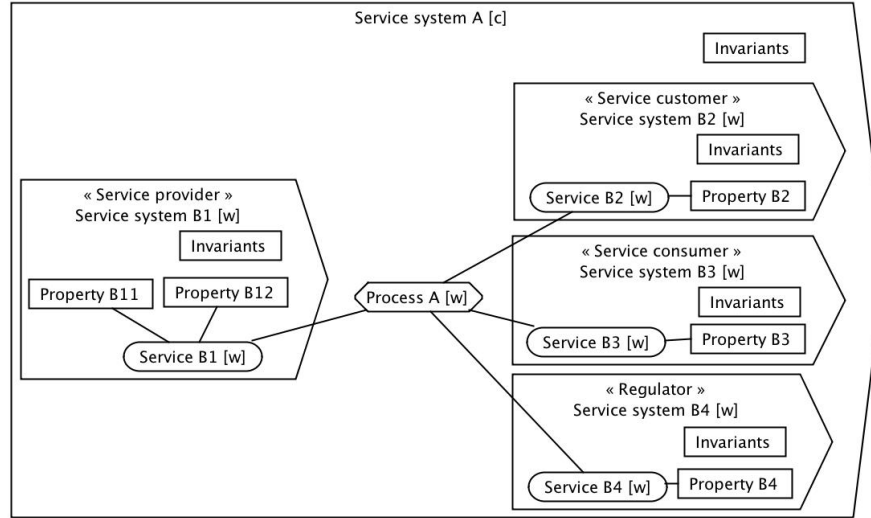


Figure 15: Service system modeling in BASS

By convention, we show the names from the model in *italics*. The main template for modeling services contains different service systems collaborating together to create the value. These service systems are shown as components of *Service system A* as a composite (white box), which correspond to the context in which we model the service systems participating in implementing the service. The component working objects (the component service systems): *Service system B1*, *Service system B2*, *Service system B3* and *Service system B4* correspond to the main roles in modeling services: service provider (providing the service), service consumer (using the service), service customer (paying for the service), regulator (implying the constraints on how to operate the service). Service provider and service consumer are always part of the service system, whereas the other roles are optionally there. Also, sometimes one service system can correspond to many roles. For instance, service consumer and service customer can be often the same service system.

These service systems collaborate together in the *Process A* with their corresponding services *Service B1*, *Service B2*, *Service B3* and *Service B4*. Each service has its properties. These properties can be of various kinds, such as quality of service, data properties, etc. In this thesis, we will consider only data properties.

Each of the component service systems can be further analyzed to model the implementation of their services. There are two ways to model service implementation: using collaboration (Figure 16) or a process (Figure 17).

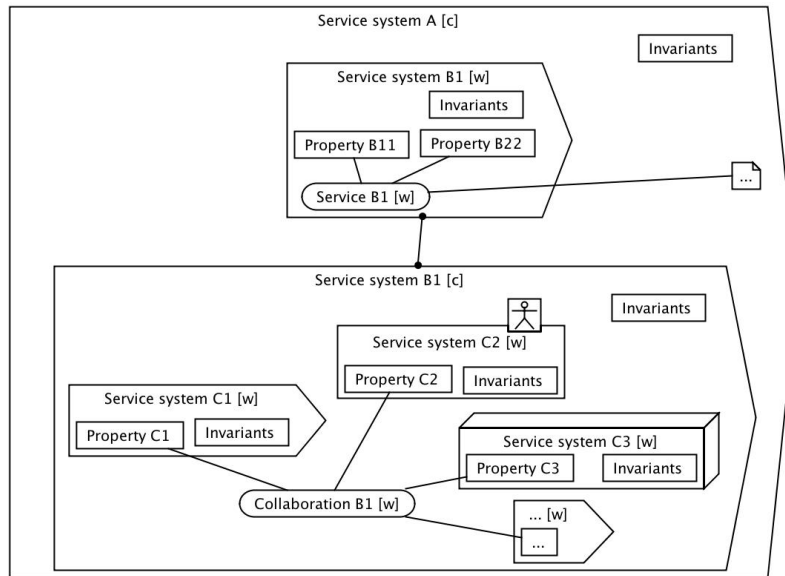


Figure 16: Service implementation with collaboration

Figure 16 shows *Service system B1* as a composite (white box) revealing its internal structure: *Service system C1*, *Service system C2* and *Service system C3*. *Service B1* is implemented with the *Collaboration B1*. *Collaboration B1* represents the way *Service system C1*, *Service system C2* and *Service system C3* collaborate to implement the *Service B1*. Collaboration corresponds to the joint action of Catalysis [15]. This means it does not state what the role of each of the systems in implementing *Service B1* is - it just gives the overall effect.

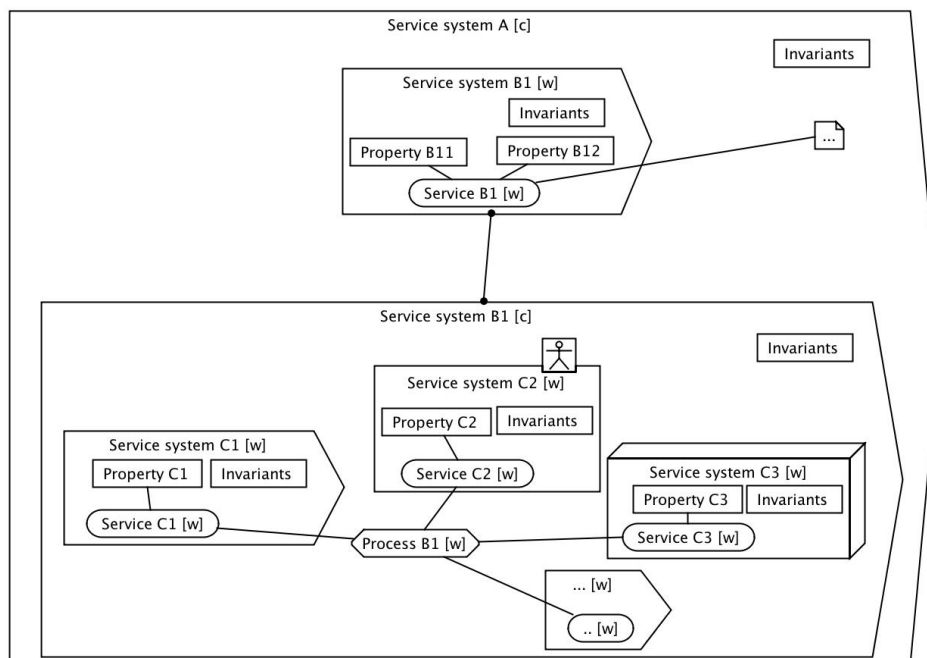


Figure 17: Service implementation with process

Figure 17 shows *Service system B1* as a composite (white box) revealing its internal structure: *Service system C1*, *Service system C2* and *Service system C3*.

Service B1 is implemented with the *Process B1*. *Process B1* represents the way *Service system C1*, *Service system C2* and *Service system C3* work together to implement the *Service B1*. Unlike collaboration, process states what the responsibilities of each of the service systems that participate in service implementation are. This is stated with the services of service systems: *Service C1*, *Service C2* and *Service C3* that the process is using for implementation. Using the process for service implementation, the logic of the implementation is stated in the services, and the process holds an invariant that states what the relations between different elements of participating service systems are, such as properties (in this thesis data).

To sum up, a working object (representing service system) as a whole provides a service. A working object as a composite implements a process that combines the services offered by the component working objects as wholes. It can also implement collaboration that shows how the component working objects collaborate to implement a service. The service, collaboration and the process are related by a refinement relationship: collaboration is a refinement of a service; a process is a refinement of a service. The process uses services of the component working objects as wholes to implement the main service. We will use this template (Figure 15) for modeling the services and processes throughout this thesis.

4.1.2 Declarative Behavior Modeling

In this thesis, we use the declarative behavior modeling based on preconditions and the effects expressed with functional units as shown in Figure 18. Each service is defined with precondition stating the condition that should hold before the service can be executed and body of the service, expressing the effect of the service shown with parameterized functional units. For each service, if the system is in a certain state defined with precondition, service affects the system in a way that is expressed with functional units. Precondition expresses the constraints on the system properties, for example that the amount on the bank account is greater than 0, when the customer wants to make the payment. Preconditions are formulas expressed with Alloy language that combine properties with logical and arithmetic operators. Effects are expressed in the diagram with functional units and their relation to the system properties. This way, it is possible to design high-level model using services, while being precise in using functional units. In case we do not need the model simulation and prototyping, we do not need to use functional units for analysis; we can only show the necessary services.

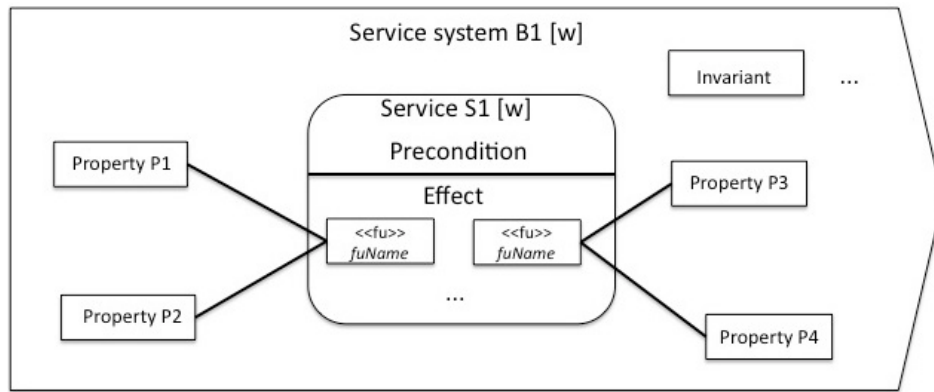


Figure 18: Full notation to show service

As explained, in our method we have three different ways to model behavior: service, collaboration and process. Service and collaboration contain the logic of behavior and are modeled as shown in Figure 18 with preconditions and effects. Process does not contain the logic expressed with functional units of behavior it just uses the services of component service systems, and holds an invariant that state the relation between elements of component service systems. This way, the behavior is modeled declaratively. This means we do not show the service implementation as a sequence of steps. Instead, we show all the services for each role in the service system with additional constraints on when they can be performed. This way, we do not restrict the behavior to only one possible path. Instead, we start with all possible scenarios and define the additional constraints on top of them. Therefore, in all examples we show there won't be a sequence of steps, like in Business Process Modeling Notation (BPMN) [10], visible in the diagram.

Besides the benefits of declarative modeling already expressed in [30], the reasons why we have chosen to use declarative modeling with precondition and service effect expressed with functional units are:

- It lets the business and IT analysts focus on high-level services, not the details of its implementation. This helps in involving the business expert in IT specification.
- Very often in the project, it is not possible to define the exact sequence of the steps in service implementation.
- There is no need to over-specify the service implementation in the initial stages of the project, which would require a lot of frequent changes during the project.

For the early design phase for which BASS method is mostly used, we believe that declarative process modeling can be effective, because it does not overspecify the model.

4.1.3 Expressing Semantics of Services: Functional Units

In order to simulate and prototype the models, it was necessary to add the concept to model the semantics of the service. In order to do that, we have introduced the functional unit. Functional unit is the predefined parameterized atomic unit of

logic. The main idea is to define library, which contains the set of generic parameterized functional units and their semantics. These units can then be combined in one service to represent its logic. In case there is no need to do simulation and prototyping of the models for the project, there is no need to show functional units.

For each functional unit, we need to define its generic template, stating its name and parameters; we also need to define its semantics expressed in different forms for service design, service simulation and service prototyping. For service design, we define how the generic functional unit in the diagram is used. For service simulation, we express the logic of generic functional unit in Alloy library (module). Finally, for service prototyping, we express the logic of generic functional unit using Arcimboldo templates in a given target language. In this thesis, we show all prototyping examples in Java.

Figure 19, Figure 20, Figure 21 and Figure 22 show the example of the find functional unit representing the search of an element in the set. Figure 19 shows the generic template of this functional unit. In the center is find functional unit shown in blue, surrounded with parameters that are necessary for this functional unit:

- In which set?: Representing the set in which we do the search.
- By what criteria? (Attribute): The attribute of the element in the set by which we do the search.
- Input: The value by which we do the search.
- Output: The result of the search, the element of the set *in which set?* whose attribute *by what criteria?* is equal to the value *input*.

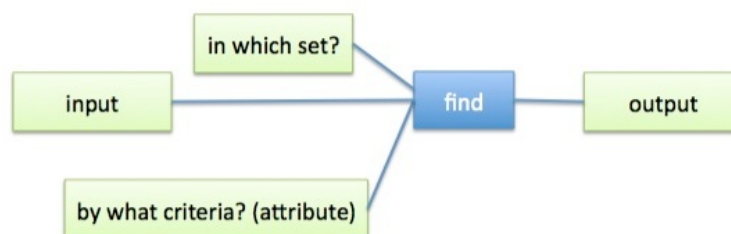


Figure 19: Predefined functional unit: generic template

Figure 20 shows how the functional unit find from Figure 19 is being used in the service design. Functional unit, marked with *fu*, is shown inside the service whose logic it defines. It is connected with the lines to the corresponding values for each of the generic parameters. Stereotypes on the lines define the parameters that are defined. For example, input parameter of find is the attribute *Name* of *OrderInitial*. The output is the data *Customer* inside the service *OrderProcessing*. Finally, criteria and set are in this case shown with one line, as by the attribute *Name* to which it is connected we can conclude also the set in which we do search, in this case *CustomerSet*.

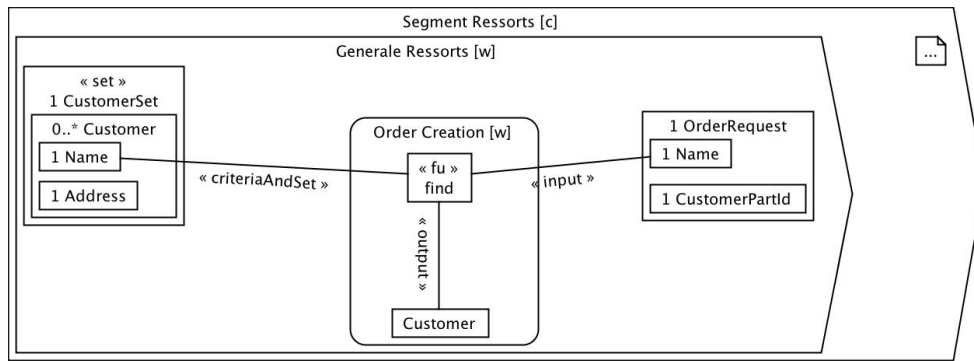


Figure 20: Predefined functional unit: graphical representation (Service design)

Figure 21 shows how the generic functional unit from Figure 19 is used in Alloy language. It is represented with the function with parameters as in generic template. The body of the function expresses the logic of the functional unit – searching for the element with given criteria in the given set.

```
module predefined_find [element,attribute]

fun find ( criteria : element -> attribute, input : one attribute, whichSet : set element) : one element {
  { output : one element | (a.criteria = input and a in whichSet)}
}
```

Figure 21: Predefined functional unit: Alloy representation (Service simulation)

Figure 22 shows how the generic functional unit from Figure 19 is used in Arcimboldo template for prototyping. The main logic of find functional unit is expressed with SQL query. The parameters are the same as from generic functional unit. And the result of research is returned as the value.

```
public {Element} find{Element}(String input) {
    EntityManager em = Manager.open();
    {Element} output = null;
    try {
        EntityTransaction tx = em.getTransaction();
        tx.begin();

        String q = "SELECT {element} FROM {WhichSet} {element} WHERE
        {repeated section attributes}
        {section stringAttribute}
        {section finder}{criteria}
        {end}
        {end}
        {end} = '" + input + "'";

        Query result = em.createQuery(q);
        Collection<{Element}> {element}s = (Collection<{Element}>) result.getResultList();
        if ({element}s.size() == 1)
            output = ((ArrayList<{element}>) {element}s).get(0);
        tx.commit();
    } finally {
        Manager.close();
    }
    return output;
}
```

Figure 22: Predefined functional unit: Arcimboldo representation (Service prototyping)

4.1.4 Use of Invariant

Invariants express the properties that must hold in certain context (during the execution of a service, inside IT system in general, etc.). From the business perspective, invariants can be used to model (business) requirements of an enterprise.

Invariants have been used both in the business and technical world: to represent and check constraints [81], to model business rules [116], process invariants related to beliefs [117] etc. In requirements engineering, KAOS methodology uses invariants for object specification, domain properties specification, and indirectly for goal specification [118]. In this thesis, we use invariants to express the constraints that must hold in the system and as a pivotal concept in improving business/IT alignment and in supporting the co-evolution of technical and business specifications.

We define the role of invariants in this thesis as follows: First, they implement the constraints required by business specification that should always hold. For example, “The order can be placed for the existing parts only”; second, they enable the designer to efficiently manage the model complexity by assuming that some of its properties always hold during an execution. They can be used to assume something that is not necessarily required or consistent with the business needs, but can help at the current stage of analysis to simplify the model. For example, “To simplify the model, let’s consider that the part’s id provided by a customer is always correct” (i.e., exists in the database); third, they are used to state the relation between the service systems collaborating together. For example, “Order in the customer company is the same as the order in the service provider company”.

We use Alloy expressions to write an invariant. Graphically, they are shown in the rectangle Invariants (Figure 15). The common practice is to show them in the upper right corner of the service system. Invariants inside one service system contain all constraints related to process or collaboration connecting many service systems or to the constraints between properties.

4.1.5 Modeling Service Multiplicity

As the proposed method is based on SEAM, the model contains instances, not the classes of the concepts in reality. We model concrete projects, people, services, etc. For example, we do not model type of organization, but the concrete company. The same applies for services, for which we show one instance of a service.

To show service multiplicity, we use different concepts to model atomic transactions and long-running transactions (business transactions). By atomic transaction, we mean as in database transaction, that this work of unit must either complete in its entirety or have no effects. We model this with service as a whole that contains functional units to express the semantics. Once the service is performed, either all the logic is performed or nothing is done at all. Long-running transactions contain smaller atomic transactions that do not necessarily need to be completed all at once. They correspond to multi-step business transaction. We model long-running transactions with service as a composite, containing many services as wholes, which correspond to the atomic transactions.

We illustrate the service multiplicity with an example of Order Creation at Générale Ressorts. The full notation is given in Figure 23. Every system has its own lifecycle. This is shown with the line connecting *IT system [w]* and *Lifecycle [w]*. Inside a lifecycle as a composite we model services, in this case *Order Creation [c]*. By default, we assume that there can be many executions of service *Order Creation [c]*. This is shown with the line connecting *Lifecycle [c]* and *Order Creation [c]* showing multiple transactions of *Order Creation [c]* with *.

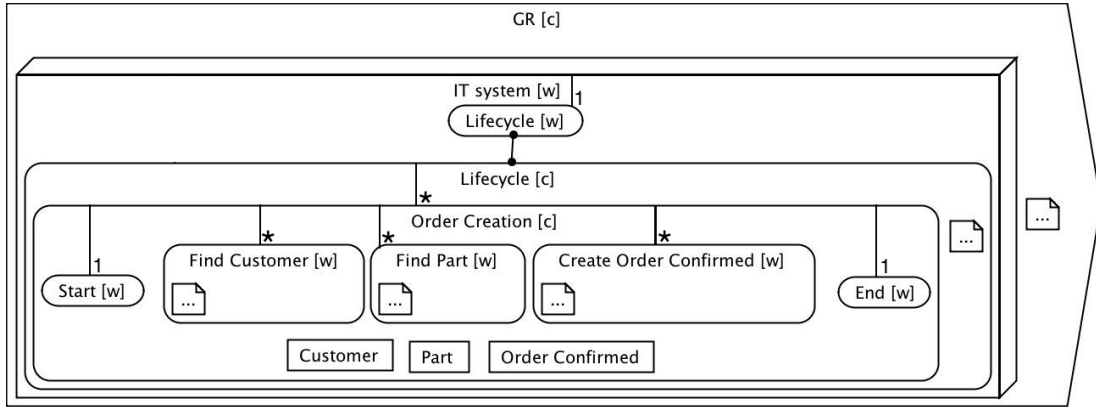


Figure 23: Full notation to show service multiplicity

Inside service as a composite, in this case *Order Creation [c]*, we have three main parts: start – containing one *Start [w]* service; middle part – containing many services, in this case *Find Customer [w]*, *Find Part [w]* and *Create Order Confirmed [w]*; end part – containing one *End [w]* service. Start and end services are services that are performed only once to start and end the long-running transactions, and they are related to the start and end state of the corresponding state machine. The middle part contains all the other services that are performed during the long-running transaction. They can be performed many times. This is shown with the lines connecting *Order Creation [c]* and *Find Customer [w]*, *Find Part [w]* and *Create Order Confirmed [w]*. This way, we model every system with its lifecycle containing long-running transactions that further contain atomic transactions. Every long-transaction has its start, living part and end. This corresponds to the systemic principle.

4.1.6 Properties and Data

As we have mentioned, each service can have its properties. These properties correspond to quality of service properties, such as business continuity, 3 days delivery guaranteed, etc.; or, they can correspond to the data used by the service, such as inputs and outputs to the service, like Order, Delivery, etc. In this thesis, we consider only the data.

For each data element in the diagram, we need to define context in which data is defined and available. It is defined by the boundary around data. For example, is it in the IT system of the company, or is it defined in the context of engineer, etc. Context determines the visibility and accessibility of data. It also determines how long

the data is available, depending on the boundary, i.e. the lifecycle in which it lives. For example, data in IT system are persistent and will be available whenever we run the application. Data inside the service will be used temporary until the (long-running) transaction is finished, for example to transfer output from one service to the other.

4.1.7 Simplified and Full Notation

We have explained the main concepts used in this method to model services and service systems using the full notation. The main template for modeling services that we propose in Figure 15 shows the simplified representation. This is the representation that is used in practice and that we use in other chapters of this thesis. The full version of this diagram is shown in Figure 24.

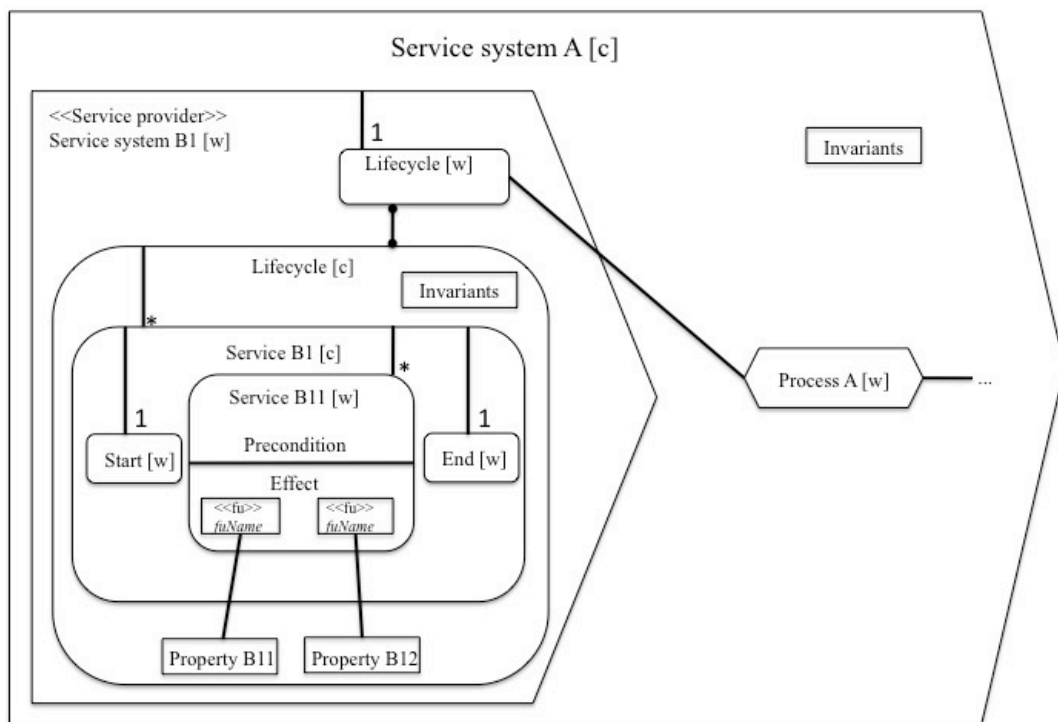


Figure 24: Diagram in Figure 15 with full notation

In practice, we do not draw all the multiplicity lines and the lifecycle. We assume that they are by default always present, i.e. that each system has its own lifecycle with the services in the middle part being performed many times, and start and end service once only. The semantically equivalent simplified version with preconditions is shown in Figure 25. As we can notice, lifecycle, start and end services and multiplicity lines are not shown in the diagram. We assume that these conditions are always by default there.

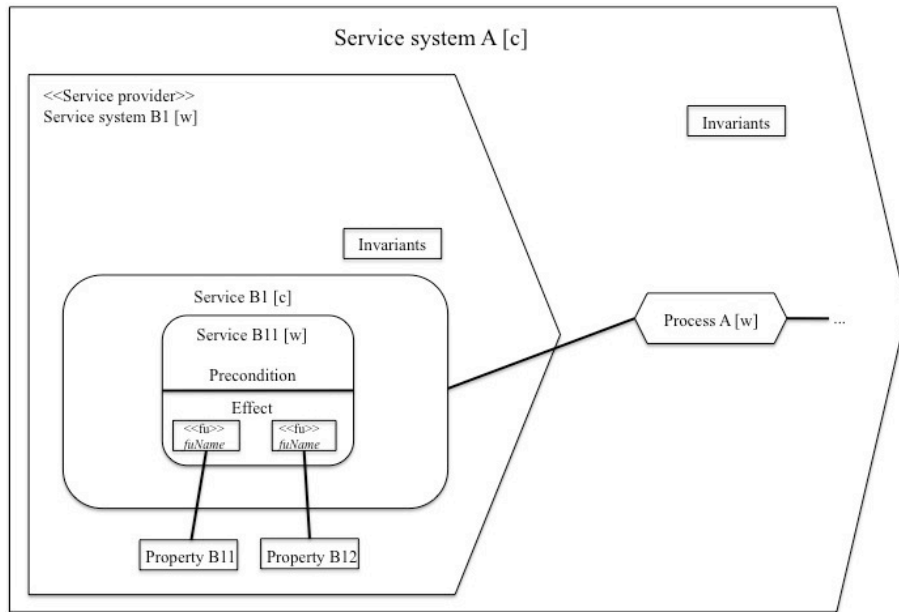


Figure 25: Diagram in Figure 24 with multiplicity simplification

In practice, we also do not show the two parts of the service stressing the precondition and effect separately. Instead, we show only the service with its functional units. Precondition is visible only as a field of the service, not in the diagram as shown in Figure 26.

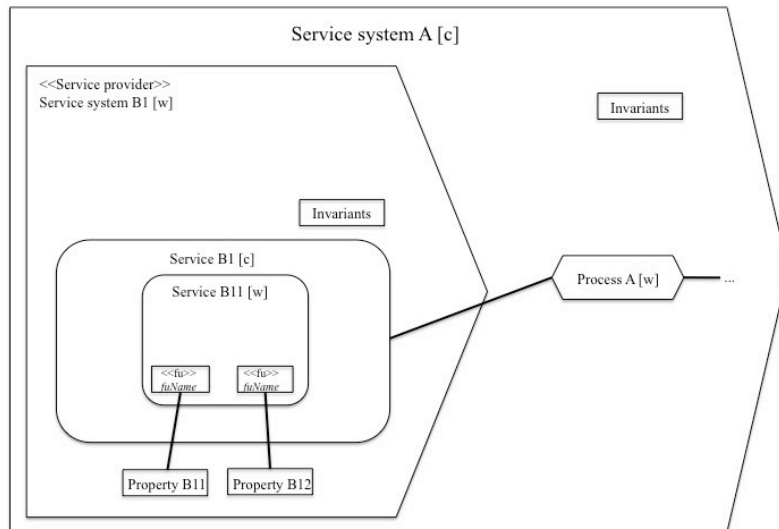


Figure 26: Diagram in Figure 24 with service representation simplification

In case there is no need to simulate and prototype the models we do not need to show the functional units. Their main purpose is to describe the semantics of the service and in that way create more precise models. In case we need to model on high-level of analysis and do not need to simulate or prototype the models, we can model without functional units as shown in Figure 27.

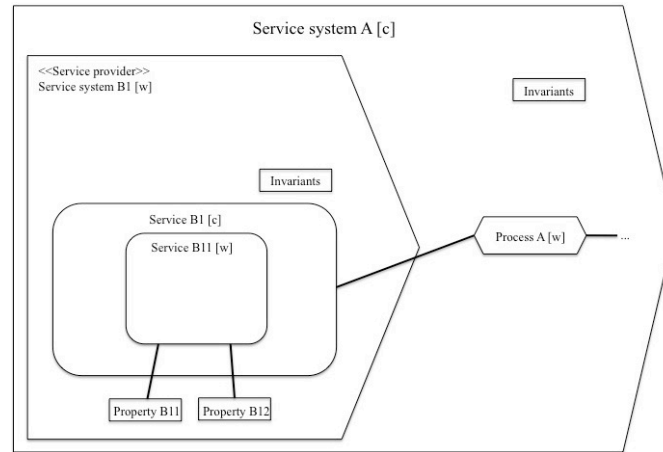


Figure 27: Diagram in Figure 24 with service semantic simplification

Once we apply the three principles to this diagram, we get the simplified representation shown in Figure 15.

4.2 BASS: Meta-model (Abstract Syntax)

In order to model and simulate service systems, we have extended SEAM with several concepts and principles, as explained. In this chapter, we give the meta-model of the proposed method containing all the necessary concepts. We first give the technical meta-model with the well-formedness rules and then the mapping to the corresponding business terms.

Figure 28 depicts the meta-model of the proposed method. As the method is focused on modeling service systems, the main concepts are service system as a whole and service system as a composite. Service system as a whole represents the organization of any kind (business organization, IT organization, human organization, etc.) without revealing its internal structure. The service system as a whole provides a service. The semantics of this service is expressed with functional units and their relation to the properties of the service system. These properties are modified once the service has been performed.

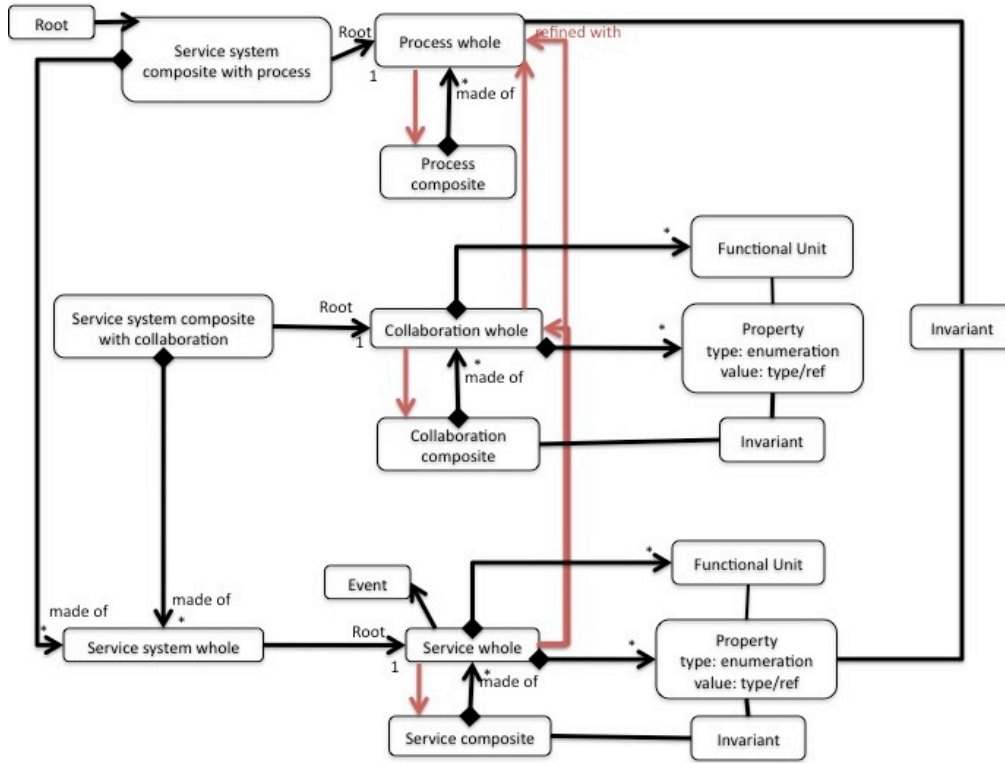


Figure 28: BASS: Meta-model

We always start with modeling service system as a composite with a process. Therefore, the root element is shown to point to this. Service systems as a whole represent components of service system as a composite. They can represent the internal structure of the service system as a composite. Service system as a composite defines how its component service systems as a whole work together to create a value. Therefore, service system as a composite provides a process or a collaboration that connects its component service systems as a whole.

Service, collaboration and process are related with the refinement relationship. Service is implemented by collaboration, and collaboration is the refinement of service. Collaboration is implemented by process, and process is the refinement of collaboration. One process uses many services from component service systems as a whole to implement a service.

Additional well-formedness rules are:

1. The process P can use the services S_1, S_2, \dots, S_n , iff they are on the same abstraction level, i.e. the same distance from the root element.
2. Invariant between the process as a whole and the properties of the service as a whole can exist only for the properties that are of receive/send type, i.e. that are shared between the service systems.
3. Exactly one service as a whole among all the services used by process as a whole has event element. This means that only one service system can be responsible for the execution of the service that the process is implementing.
4. There is exactly one root element-service system as a composite.
5. There is no service system that can be its own ancestor. This means there are no cycles in service system hierarchy.

6. There is no service that can be its own ancestor. This means there are no cycles in service hierarchy.
7. There is no collaboration that can be its own ancestor. This means there are no cycles in collaboration hierarchy.
8. There is no process that can be its own ancestor. This means there are no cycles in process hierarchy.
9. There is no property that can be its own ancestor. This means there are no cycles in property hierarchy.
10. Service can relate to the properties iff they are in the same service system as the service.
11. Collaboration can relate to the properties iff they are in the same service system as the collaboration.

The meta-model in Figure 28 is expressed with the service terminology. Table III summarizes the corresponding business terms of the concepts defined in the meta-model.

Table III: **BASS: Meta-model Concepts and Corresponding Business Terms**

Meta-model term	Business term
Service system	Market segment, company, organization, company departments, humans, etc.
Service	Externally visible behavior of the market segment, company, organization, etc.
Collaboration	Internal behavior of the market segment, company, organization, etc. without the responsibilities on who is doing what
Process	Internal behavior of the market segment, company, organization, etc. with the responsibilities on who is doing what
Functional units	Detailed description of the behavior of the market segment, company, department, etc.
Properties	Internal information relevant to the behavior of the market segment, company, organization, etc.
Invariant	Constraint set between different companies, organizations, departments, etc. or between their properties.

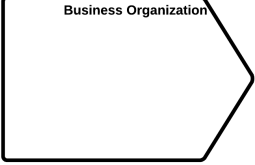
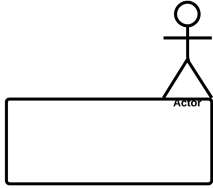
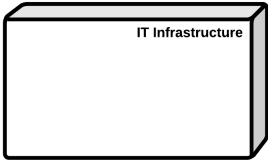
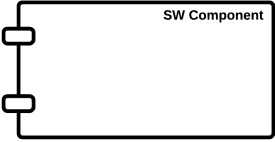
Service systems can correspond to the market segment, company, organization, department, humans, etc. Service is an externally visible behavior of the company, department, human, etc. Collaboration and process are internal behaviors of the company, department, human, etc. with a difference that process defines how the responsibilities are split in the company, organization, etc.

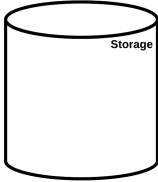


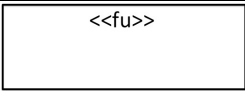
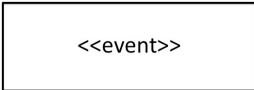
Functional units are details of the behavior of the company, department, etc. Properties are externally and internally visible information of the company, department, etc., respectively. The way they are shared between systems is expressed in an invariant of the process connecting the services of these systems.




4.3 BASS: Visual Notation (Concrete Syntax) and Semantics

In this chapter, we show the visual notation and semantics of the proposed method. We also show how this notation is mapped to the concepts explained in the meta-model. Table **IV** summarizes the visual notation elements used in the proposed method with their visual representation, name and description. Description contains information (marked in bold) of the meta-model element which this visual element models. Elements 1-5 are used to model service systems, 6-10 their behavior, 10-11 their static part, and 12-14 relationships between elements.

Table IV: **BASS: Visual representation of meta-model concepts and their semantics**

	Visual Representation	Name	Description
S E R V I C E S Y S T E M S		Business working object	This element is used to model the market segment, company, department, or some other business organization. It models one type of the service system : business organization. As any other system, it can be modeled as a whole and as a composite. Inside the system, we can model the structure with any other type of system and properties and the behavior of the business organization with business services and processes.
		Human working object	This element is used to model the human, as an engineer or manager. It models one type of the service system : human organization. As any other system, it can be modeled as a whole and as a composite. Inside the system, we can model the structure with other human organizations and properties and the behavior of the human organization with business services and processes.
		IT working object	This element is used to model IT infrastructure. It models one type of the service system : IT organization. As any other system, it can be modeled as a whole and as a composite. Inside the system, we can model the structure showing other IT infrastructures, software components and storage and the behavior of the IT organization with IT services and processes.
		Software component	This element is used to model software component. It models one type of the service system : software organization. As any other system, it can be modeled as a whole and as a composite. Inside the system, we can model the structure showing other software components, IT infrastructures and storage and the behavior of the IT organization with IT services and processes.

B E H A V I O R		Storage	This element is used to model storage. It models one type of the service system : storage organization. As any other system, it can be modeled as a whole and as a composite. Inside the system, we can model the structure showing other software components, IT infrastructures and storage and the behavior of the IT organization with IT services and processes.
		Service	This element is used to model service or collaboration, both on business and IT level. When used inside service system as a whole it is called service , and when modeled inside service as a composite it is called collaboration . It can be modeled as a whole and as a composite. We can show its structure with other services and its behavior with the functional units.
		Process	This element is used to model process , both on business and IT level. It can be modeled inside any service system as a composite or any other process as a composite. It can be modeled as a whole and as a composite. We can show its structure with other processes and the behavior with the relation to other services.
		Functional unit	This element is used to model functional unit . It is modeled inside any service and its main purpose is to show the semantics of that service, i.e. the effect of that service on the state of the system. It is connected to other properties to show the parameters of the functional unit.
		Event	This element is used to model the event in any service system as a whole. It is always related to the service as a whole. It means that this service system is responsible to provide the results of the related service. For all services used in one process, there should be exactly one related to the event, which is in charge of operating this process.

S T A T I C		Property	This element is used to model the property of any service. It is related directly to the collaboration or indirectly to the service, via functional unit. It is modeled inside any service system as a whole.
		Association relation	This element is used to model the relation between any structure or behavior element of the method. It shows the association relationships between these elements.
	<u><<relationship>></u>	Relationship relation	This element is used to model the relation between two properties. It can also contain the cardinalities, such as 0..1, 0..*, 1..*, *, etc.
R E L A T I O N S		Refinement relation	This element is used to model the refinement relations between system as a whole and system as a composite, service as a whole and service as a composite, collaboration as a whole and collaboration as a composite, or between processes as a whole and process as a composite.

The proposed method inherits some of the principles for representing a diagram used in SEAM: explicit context, explicit hierarchy, explicit roles and explicit mapping to reality.

Explicit context: The context is always made explicit in a diagram. In many notations, such as UML, the context is implicit. For example, in the UML use case diagram [15], it is possible to hide the IT system boundary [14]. The underlying principle that explains why UML allows hiding the IT system boundary is called the Occam-razor principle. This principle expresses that a succinct theory is better than a verbose one. The boundary is not considered as an important concept, so it can be hidden. In a systemic model, the boundary is probably one of the most important concepts, so it has to be visible.

Explicit hierarchy: The organizational hierarchy captures systems' construction. The functional hierarchy captures behaviors' structure (services or processes). We represent hierarchies as boxes inside boxes. We do not have, as in UML, composition relationships. With this we put an emphasis on which concepts are hierarchical and which ones are not. The hierarchy also makes the context in which the element is defined explicit.

Explicit roles: The key actors are represented with their explicit roles. It is possible to have the “same” actor with different roles (in different systems). This is extremely useful to analyze conflicts of interests and resource issues.

Explicit mapping to reality: We show concrete projects, people, services, departments, etc. Very often, in workshops, if possible, we use the picture of the real people, companies, products, etc. This helps make the model concrete. If we make a business model, we analyze one representative customer with a name.

5 BASS Service Design Spirals: Theory

In this chapter, we describe BASS spirals: service specification spiral and validation and verification spiral. The spirals are the result of several attempts to define the service design process. We were improving the process based on our understanding of the requirements and goals of the process, feedback we got from practitioners and different projects on which we have applied it. This has resulted in defining two spirals for service design: service specification spiral and validation and verification spiral.

Service specification spiral is the main spiral used in any project done with BASS. Following the steps of the spiral, a business/IT analyst defines an overview of the whole business case, from definition of services that the customer needs to analysis of internal organization and responsibilities needed to implement the services. In each step, the design decisions are captured and the model is refined into a more detailed one. As a result, the business/IT analyst creates the set of BASS models corresponding to different viewpoints of the modeled business case. For each of these models, validation and verification spiral can be applied to create more precise models when needed. Following the steps of validation and verification spiral, business/IT analysts together with the formal specialists can simulate the model, detect and resolve anomalies or find missing business rules. The initial ideas for the work in this chapter are described in [119].

5.1 Service Specification Spiral: Theory

Figure 29 shows an overview of one cycle of the service specification spiral (Figure 1). It shows 6 different models: 1 model as a result of the initial phase, 5 models as results of steps of the spiral (and fifth optional step). In each step we refine the model by capturing the design decisions and possibly adding the concepts to the model. The result of each step is a new model. Once we do one cycle of the spiral in which we analyze the responsibilities inside one organization, we can apply the same steps on any of its internal organizations, until we reach the needed level of abstraction. In steps 1-4, we use the “identify and allocate” (or conceptualize and design) pattern, i.e. in each step we identify the new elements and then allocate the existing elements to the newly identified one. Sometimes, it can happen that when we identify new elements, they can lead to the creation of additional elements, too. For instance, when we define the structure of one service system, we might observe some new properties that we did not have a chance to observe before. The last step is used to split the service systems, i.e. to create systems that can be observed independently, containing all the necessary information about the other service systems. The whole service specification spiral is focused on the service identified in the conceptualization phase. At the beginning the external services needed by the customer are identified and then the details of its implementation are defined. Therefore, it corresponds to the service provider perspective. We explain the steps of

the spiral: what is done in each step, what is the value of that step and viewpoint to which it corresponds. By convention, the elements that are crucial for each step, such as the newly identified elements, we mark in bold.

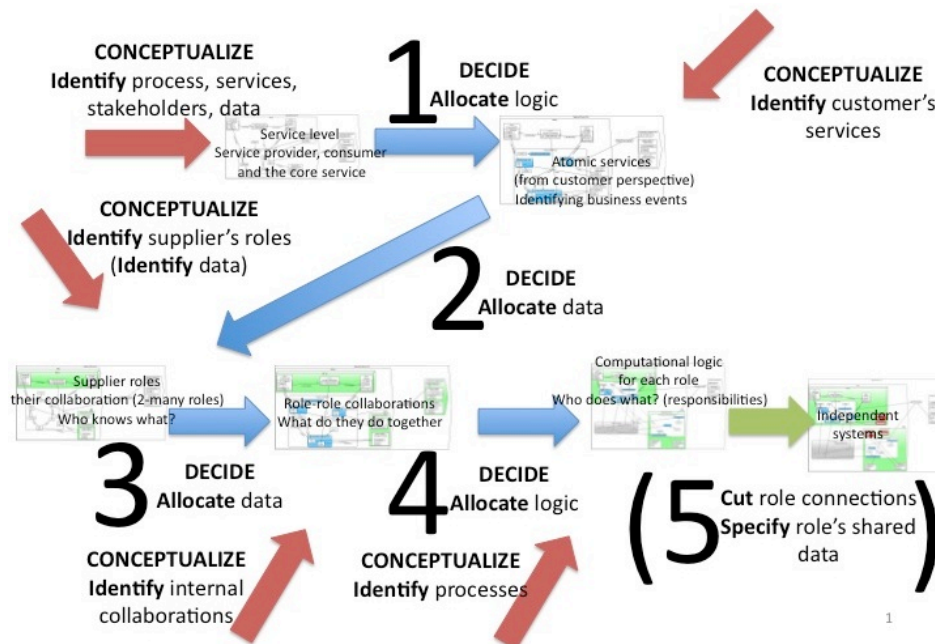


Figure 29: One cycle of BASS service specification spiral

5.1.1 Initial Model Design

The first phase of the process is to create initial model by building a set of concepts from universe of discourse. In this phase, we **identify the main service systems** participating in creating the value (stakeholders), based on the template in Figure 15. We identify service provider, service consumer and potentially other participants, such as service customer and service regulator. As in the further phases of the spiral we will refine service provider more, we refer to service consumer, service customer and service regulator, as the external service systems. We also **identify the main process** involving the **main external service** of service provider and **services** of other stakeholders. Then, we **identify the services** of each service system that are used to implement the identified process. These services can be identified with or without functional units, depending on the fact whether simulation is relevant for the project. Finally, we **identify service properties** in each of the service systems based on the requirements coming from external service systems. The result of conceptualization matches the template shown in Figure 15 for service specification, showing only the core services and service systems.

The value of this phase is in capturing the main stakeholders participating in core service implementation and the service properties relevant for them. This way, it frees the business and IT analysts of thinking about the implementation details. Instead, it enforces them to think about the high-level requirements, which, when jumped directly to implementation, are very often misunderstood or omitted.

The resulting diagram corresponds to the **management's perspective** of the service provider organization, as it analyzes the service provider organization and its

environment. It captures the core service and the main requirements in form of service properties that come from external stakeholders. In this thesis we deal with the data properties, in this case meaning the inputs and outputs of the service exchanged with the external stakeholders.

5.1.2 Step 1: External services

In the initial phase, we have identified the core service and the service systems participating in its implementation and captured main requirements in form of service properties. In order to be sure that the service implementation will match the needs of the external stakeholders, besides the requirements captured in form of properties, we need to identify externally observable services of the service provider who is in charge of service implementation. This means, we need to identify the sub-services of the service in service provider. These services are identified based on the business events relevant to the external stakeholders, such as consumer. They can be related to the deliverables the external service systems would expect to have, or to the main functions they would expect to observe. Once we have identified the externally visible services, we need to allocate the logic of the service in form of functional units to the newly defined services. In case we model without functional units, we need to allocate the properties, as they are related to the service directly in this case.

The value of this step is to show an integrated view for the external service systems, such as consumers. By identifying externally visible services, it complements conceptualization phase with service properties, and creates the diagram with fully identified requirements for the service to be implemented. Again, it enforces the business and IT analysts to think about the externally visible services, instead of jumping immediately to the implementation process.

The resulting diagram corresponds to the **management's perspective**. It captures the business needs based on the interactions with the external service systems, such as consumer. It creates an integrated view for the external service systems, which captures the externally visible services and their properties.

5.1.3 Step 2: Internal organization

In step 1, we have analyzed the external service systems and identified the services that need to be provided to them, and their properties. In step 2, we need to decide on how the service provider is going to implement the identified services. Therefore, we need to identify the service systems inside the service provider participating in the service implementation. This means, we need to **identify provider's roles**, which correspond to the teams with certain roles in providing the service. They do not necessarily need to correspond to organizational chart and to correspond to organization's departments. Instead, it is possible to have people from different departments in the same team, if they have the same role in providing the service, such as supporting the service. Also, it is possible to have external organizations as a part of the provider, if they are involved in providing the service. Once we have identified who participates in implementing the service, we need to

decide who is in charge of what property. Therefore, we **allocate the properties** to the provider's roles. Note that once we have shown the internal organization of the service provider, there might be some new properties as well that are visible now, and that were not visible when we were thinking about the service provider generally without its internal structure. Therefore, it is also possible to define the new properties in this step and allocate them to one of the provider's roles.

The value of this step is to think about the internal structure of the service provider, based on the service they are implementing. It enforces thinking about the service provider organization from the service perspective, instead of in a traditional way, in terms of organizational hierarchy. It enforces finding the new way of internal structure, based on the service the organization wants to provide.

The resulting diagram corresponds to the **service provider's perspective**, as it captures the internal structure of service provider organization.

5.1.4 Step 3: Internal services

In step 2, we have identified who participates in service implementation. We also need to identify what services are to be implemented. These services can be different than the one identified in step 1, as in step 1 we have identified externally visible service, and now we want to **identify internally visible services** as well. They may include some new internal services that were not observable from the external systems. Once we have identified the new internal services, we need to define their semantics. This means we need to **allocate the service logic** in form of functional units to the newly defined services. Note that we model the internal services in form of collaboration, i.e. without explicitly stating which provider role is responsible for what internal service. We will do this in the next step.

The value of this step is to enforce thinking about the internal services needed to provide externally visible services. It enforces to make distinction between the internally and externally visible services. When we analyze the service provider and its environment, we think about the externally visible services and their requirements, and do not bother with the details of implementation. When we analyze the service provider internal organization and service implementation, we think about the internally relevant services.

The resulting diagram corresponds to the **service provider's perspective**, as it captures the internal services of service provider organization.

5.1.5 Step 4: Internal responsibilities

In step 3, we have identified who participates in service implementation and what the internal services necessary to provide externally visible services in form of collaboration are. In this step, we need to identify the responsibilities of each provider role in implementing the service that is provided. In order to do that, we **identify the process with its services** in the corresponding provider's roles. For each service identified in step 3, we decide who is in charge of it and based on that create the

services and their relations to the process. We also need to **allocate the logic of the internal services** to the newly defined services of provider's roles.

The value of this step is to enforce thinking about the responsibilities of each of the provider's roles in the service implementation based on the internal services that should be implemented. This way, we distinguish between different abstractions of services, depending on the analyzed perspective: services relevant for external systems, such as consumer (step 1), services relevant for service provider (step 3) and services relevant for provider role's (step 4). Each of these services is relevant for different perspectives. Therefore, it is useful to enforce clear distinction between them, as it frees the business and IT analysts of thinking about the other aspects of the service system at the same time and they can bring full attention to the currently analyzed service.

The resulting diagram corresponds to the **provider role's perspective**, as it captures the responsibilities of service provider role's organization.

5.1.6 Step 5: Independent service systems (optional)

In step 4, we have defined the full structure and the behavior of the service provider and its roles that defines how the service should be implemented. Therefore, in the last, optional step, we can split the provider's roles to make independent service systems. This way, we can observe the systems independently of each other, because one system will contain all the necessary information about the other systems, i.e. the simulation of all the surrounding systems. In order to do that we **cut the provider's roles connections** and **specify the interfaces** for each system that contains properties that are shared with the other systems. Note that in the current method, we simulate only the static part of the surrounding systems with the interfaces representing the exchanged properties, i.e. each system knows where the properties come from. As a part of the future work, it could be extended with simulation of the behavior of the surrounding systems.

This step is important if we want to be able to observe one service system without its environment and still have all the necessary information.

The resulting diagram corresponds to the **provider role's perspective**, as it defines the independent provider role's service system.

Once we finish one cycle of the spiral, we have defined the internal organization and their responsibilities for the modeled service system. In the next cycle, we can apply the same steps on any of internal organizations to define organization and responsibilities needed to provide necessary services. Once we have reached the necessary level of details for the given project, we have finished the service specification spiral.

5.2 Validation and Verification Spiral: Theory

Validation and verification spiral is used optionally when the analysts require more precision in the model and would like to validate it by observing instances of its

behavior. Following the steps of the spiral, business/IT analysts together with the formal specialists can simulate the model, get a visual feedback about the behavior of the modeled system, detect and resolve possible anomalies. We explain the three steps of the spiral.

5.2.1 Initial Model Design

We create initial model specified in Alloy. It can be created manually or automatically, using the tool BASS2Alloy that transforms BASS model to Alloy specification. Initial model includes the service system of interest: the main stakeholders, properties and services with initial assumptions about our model based on the business needs. This model can be any model of the service specification spiral, but it is mostly used in the first step, to detect anomalies at early stages.

5.2.2 Step 1: Generate Samples

We simulate our initial model by using the Alloy Analyzer tool. Technically, a model written in Alloy represents a logical formula; model simulation means searching for a model instance that satisfies this formula. If it exists, it indicates that the formula is consistent (i.e., no contradictory constraints are specified). In our spiral, we validate with stakeholders if the model corresponds to the business needs and if there are some anomalies by studying the random set of model instances generated by Alloy Analyzer. Model instances reveal the issues in the system behavior, possible issues with the existing business rules and indicate the missing or implicit rules. Once a new business rule is discovered, the business analyst specifies this rule in a natural language. Then this rule is added to the Alloy model for further simulations. An example of business rule discovery process is explained in [122].

We test if the model corresponds to the business needs and if there are some anomalies by studying the behavior of the service system observing the generated samples. In addition, we can verify if the model conforms to the meta-model.

There are two types of anomalies that can be observed: anomalies due to underspecification and anomalies due to overspecification.

Underspecification means that some behavior prohibited by the specification still appears during the simulation. Overspecification means the opposite: some expected behavior is not observed during the simulation.

5.2.3 Step 2: Correct Model Based on Samples

In case of underspecification anomaly, we restrict the model by adding new constraints. In case of overspecification anomaly, we need to relax the constraint. Business/IT analysts do this with the help of a formal specialist. We can add the changes to the generated simulation code available in the simulation tool, or to the model and then re-generate the code. In case of making the changes in code, we need to make sure that all the additional constraints that we detected in this cycle are also added to the model, so that they are not lost.

5.2.4 Step 3: Check Assertions

We verify if all desirable properties and business rules hold using assertions. This way, we check if there are any counterexamples that would show that the assertion does not hold.

5.2.5 Step 4: Correct Model Based on Assertions

In case we detect that the assertions are not valid, we correct model as explained in step 2.

Once we have finished one cycle of the spiral, we can continue to check the model to make sure that with the changes we did not introduce some new errors in the model. Once we do not detect any anomalies by observing samples (validation with stakeholder) and check that all business rules and constraints are valid (verification), we have finished the validation and verification spiral.

In addition to validation using Alloy simulations, as explained in this loop, we use the prototyping for validation. The result of prototyping is executable application in the given target language, in this case Java, reflecting the behavior of modeled services. Application prototypes are then used to obtain feedback from the business/IT analysts, customers and other stakeholders and in this way to validate the models.

Simulation-based validation is useful for static validation, to verify the constraints of the model. For example, it can be used to verify the model against the meta-model, to detect the inconsistencies in the model or to validate business rules and discover hidden ones. Prototyping-based validation is useful for dynamic validation, in which the user can interact with the system, observe the system from different roles' perspectives, analyze the services of each role, their semantics and the effect on the service system, as well as simultaneous service executions.

6 BASS Service Design Spirals: Case of Order Creation at Générale Ressorts

In this chapter, we illustrate BASS spirals with the working example. We first describe the working example. This example is based on the consulting project we have conducted in the company Générale Ressorts. Générale Ressorts SA is the market leader in watch barrel springs and a first-class manufacturer of tension springs, coil springs, shaped springs and industry components [120]. Générale Ressorts SA works with thousands of customers and strives to ensure the highest quality both for its products and for its customer services.

All models in this chapter are created using the simple notation explained in Chapter 4. We use the same notation for all models in the spiral. The advantage of this is much easier and quicker understanding of the used concepts by both business and IT analysts. We have tested this on participants of workshops who were able to create their models using given notation after just a short introduction of the method. Although the notation is the same on different levels of abstraction, the service systems and properties itself are different depending on which level of abstraction we model.

6.1 Working Example: The Case of Order Processing at Générale Ressorts

We illustrate the method applying it to the design of the “Order Creation” service for Générale Ressorts SA (GR). “Order Creation” is a part of an “Order Processing”: it is followed by “Order Delivery” and “Accounting”. The whole process, from the moment the customer makes an order to the delivery and the accounting is known as order-to-cash cycle. It can vary from company to company. A typical one is shown in Figure 30 [121]. To illustrate the method, we show the simplified “Order Creation” service that covers the process from the moment the company receives the order to the moment the order is booked. It is then followed by the delivery and payment of the ordered product.

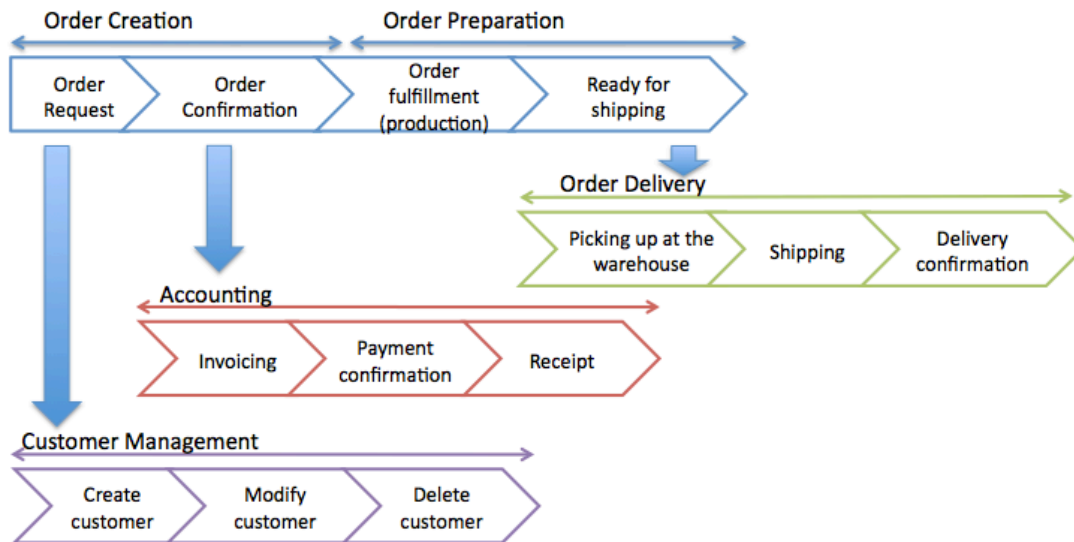


Figure 30: Order-to-cash cycle - Adapted from [121]

The use cases for "Order Creation" and "Order Delivery" are formalized following the recommendations from [14] and presented in Table V, Table VI and Table VII, Table VIII respectively. By convention, information in italics represents the corresponding names of the elements in the model.

Table V: **Order Creation: Section Main**

Use Case:	Order Creation
Purpose:	Capture the customer's orders for parts.
Actors:	Customer, OrderEntryPerson
Type:	Primary and real
Overview:	The company gets a request from a customer (<i>OrderRequest</i> ¹) for manufacturing of a specific watch component identified by its ID (<i>partID</i>). The resulting <i>OrderRequest</i> contains a customer <i>name</i> , <i>address</i> , <i>partID</i> and <i>partInfo</i> . A company agent (<i>OrderEntryPerson</i>) identifies the customer and the part to manufacture by entering the customer's <i>name</i> and the <i>partID</i> to the enterprise information system (<i>EIS</i>). The process terminates with a creation and confirmation of a customer order (<i>OrderConfirmed</i>) in the enterprise information system.
Cross References:	Business rules: BR1, BR2 and BR3.

Table VI: **Order Creation: Typical Course of Events**

Actor Action	IT System Response
1. This use case begins when the company (<i>GR</i>) gets a request (<i>OrderRequest</i>) for manufacturing a specific watch component identified by its ID (<i>partID</i>).	
2. A company agent (<i>OrderEntryPerson</i>) receives the <i>OrderRequest</i> from the <i>Customer</i> containing all the necessary information for order processing: customer's <i>name</i> , customer's <i>address</i> , <i>partID</i> and <i>partInfo</i> .	
3. <i>OrderEntryPerson</i> enters the customer's <i>name</i> and the <i>partID</i> to the enterprise information system (<i>EIS</i>).	4. Finds the customer and the part by <i>name</i> and <i>partID</i> , respectively.
5. <i>OrderEntryPerson</i> identifies the customer and the part to manufacture.	
6. Using the identified customer and part to manufacture, <i>OrderEntryPerson</i> creates <i>Order- Confirmed</i> in <i>EIS</i> .	7. Adds <i>OrderConfirmed</i> to the database and present it.
8. <i>OrderEntryPerson</i> confirms the order to the <i>Customer</i> and prepares the delivery of the given part to the given customer.	
Extensions:	
5a. In case the customer is not found, <i>OrderEntryPerson</i> enters necessary data for creating the customer in the system.	

Table VII: **Order Delivery: Section Main**

Use Case:	Order Delivery
Purpose:	Capture the order delivery to the customer.
Actors:	Customer, DeliveryPerson, Engineer, Clerk
Type:	Primary and real
Overview:	Based on the confirmed order (<i>OrderConfirmed</i>), containing information about the <i>customer</i> and the <i>part</i> , a <i>DeliveryPerson</i> requests the part to be prepared. An <i>Engineer</i> prepares the part to be delivered. The process terminates with a <i>Clerk</i> delivering the part to the <i>customer</i> , based on information in <i>OrderConfirmed</i> .
Cross References:	Business rules: BR4.

Table VIII: **Order Delivery: Typical Course of Events**

Actor Action	IT System Response
1. This use case begins when the customer's order is confirmed, i.e. booked. All information is stored in <i>OrderConfirmed</i> (what <i>part</i> should be delivered to which <i>customer</i>).	
2. Based on information in <i>OrderConfirmed</i> , a person in charge of delivery (<i>DeliveryPerson</i>) requests the <i>part</i> to be prepared.	3. Receives the request for part and sends notification to the <i>Engineer</i> .
4. An <i>Engineer</i> prepares the package with the part, based on <i>partID</i> and <i>partInfo</i> . He enters in the system that the part is ready to be delivered.	5. Sends notification to the <i>DeliveryPerson</i> that the part is ready to be delivered.
6. <i>DeliveryPerson</i> checks the package and confirms that the part is ready to be delivered.	
7. <i>Clerk</i> delivers the package with the part to the customer, based on information in <i>OrderConfirmed</i> (customer's <i>name</i> and <i>address</i>).	

We specify the following business rules for our process:

- BR1: The created order must include the complete part specification (to be used for the order fulfillment) and the complete customer details (to be used for product delivery);
- BR2: The order can be confirmed only when the customer exists in the system;
- BR3: The order can be placed for the existing parts only;
- BR4: The company has to guarantee "no faulty delivery".

The working example of the Order Creation and Order Delivery will be used to illustrate the proposed method. First we use the example to model service specification for the Order Creation in company Générale Resorts following the steps of service specification spiral. Then, we illustrate how these models for Order Creation can be simulated to detect and resolve anomalies following the steps of validation and verification spiral.

6.2 Service Specification Spiral: Case of Order Creation at Générale Resorts

We illustrate the service specification spiral on the working example of company Générale Resorts, as explained in the previous chapter. To illustrate the spiral, we do not consider business rules and Order Delivery service. Instead, we focus on Order Creation service only. By convention, information from the diagram is shown in italics.

6.2.1 Initial Model Design

In the initial phase, the analyst builds a set of concepts by interacting with his universe of discourse. He conceptualizes the service systems (stakeholders), main process and services (with functional units and events), and properties relevant for this level of abstraction. In this case, we have only two relevant service systems: *Générale Resorts[w]* as the service providers and *Customer Company[w]* as the service customer and consumer. The main process we model is *Order Creation[w]* with the services it uses: *Make Order[w]* in *Customer Company[w]* and *Create Order[w]* in *Générale Resorts[w]*. He defines the semantics of the services with functional units, as well as the properties relevant for the modeled level of abstraction.

The result of the initial phase is shown in Figure 31. We model *Segment Resorts[c]* with the two identified stakeholders: *Générale Resorts[w]* and *Customer Company[w]*. The core service that is modeled is Order Creation. In this diagram, we have already shown the implementation of that service with the *Order Creation[w]* process and the services it uses *Make Order[w]* and *Create Order[w]*. As we also use this example to simulate and prototype the models, we show the semantics of services with functional units. In this case, Customer Company only sends *OrderRequest* and receives *OrderConfirmed* as the confirmation, which is captured by the system interfaces in form of data with stereotypes *send* and *receive*. Therefore, service *Make Order[w]* in *Customer Company[w]* has no functional units. In service *Create Order[w]* we model functional units representing the semantics of this service: finding the customer in *CustomerSet* based on the *CustomerName* in *OrderRequest*, finding the part in *PartSet* based on the *PartId* in *OrderRequest*, creating the *OrderConfirmed* and adding *OrderConfirmed* to the *OrderConfirmedSet*. This service also has the event in *Générale Resorts[w]* related to it, meaning that the company Générale Resorts is in charge of operating this service. In addition to that, the model contains invariant stating that the customer with the given name is always in a customer set and the part with the given part ID is always in a part set. With this constraint we model only the successful scenario. If we would like to model what happens in case the customer or the part are not found, we would remove the invariant and add functional units for creating the customer and/or part. Finally, we define the properties in each of the service systems. We show the properties shared between the service systems: *OrderRequest* and *OrderConfirmed*; we also show the properties necessary to parameterize functional units: *CustomerSet*, *PartSet*, *OrderConfirmedSet*.

To sum up, as the result of initial phase, we have modeled *Générale Ressorts* as the service provider and its environment: *Customer Company* as the service customer and consumer. We have modeled the core services *Make Order* and *Create Order* participating in the process *Order Creation*. Based on the *Customer Company* needs, we have modeled the relevant service properties. In this phase, we enforce thinking about the customer's needs and their requirements, without thinking about the details of implementation.

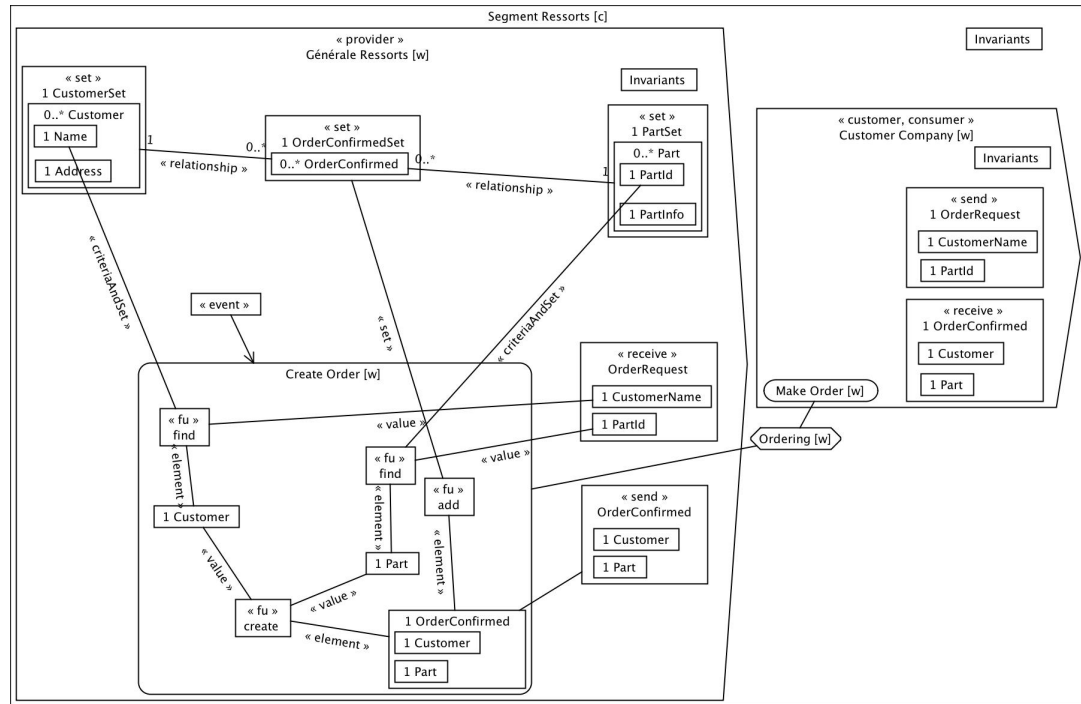


Figure 31: The result of initial model design

6.2.2 Step 1: External Services

In this step, we add detailed external services of *Générale Ressorts* to accommodate expected interactions with the customer. In the case of simplified order creation, this service is the same as already modeled service *Create Order[w]*. In case there would be some other externally visible services except *Create Order*, we would need to allocate functional units to them. As there are none, we can continue with step 2.

6.2.3 Step 2: Internal organization

Once we have identified the service systems participating in service implementation and the externally visible services of the *Générale Ressorts* as the service provider, we need to decide on the details on how *Générale Ressorts* will implement the service *Create Order[w]*. First, we need to decide on internal structure of the service provider, in this case *Générale Ressorts*. We define the internal structure based on the service they provide. Therefore, we do not use the traditional organization structure, showing the hierarchy of departments. Instead, we define all

the organizations and persons participating in providing the service. One organization can include people from different departments. In the case of simplified order creation at Générale Ressorts, we identify two participating service systems: Enterprise Resource Planning system (*ERP[w]*) and the person in charge for processing the order (*OrderEntryPerson[w]*). Then, we need to allocate existing properties of Générale Ressorts to the new identified service systems. This allocation is done based on the fact who knows what properties, i.e. for whom those given properties are important. Note that, as we reveal the internal structure of Générale Ressorts, it is possible to add new properties that were not visible when we have discussed Générale Ressorts as a whole, in its environment. In this case, we have no such properties. The identification of new service systems and allocation of the properties to them is shown in the Figure 32 and Figure 33.

Roles
ERP
OrderEntryPerson

Figure 32: Step 2: Conceptualize - Identifying provider's roles

	ERP	OrderEntryPerson
OrderRequest		X
OrderConfirmed		X
CustomerSet	X	
PartSet	X	
OrderConfirmedSet	X	

Figure 33: Step 2: Decide - Allocating properties to roles

The result of the step 2 is shown in Figure 34. *Segment Ressorts[c]* and *Customer company[w]* keep the same structure, only Générale Ressorts is refined to define the necessary details needed for the implementation of the service *Process Order[w]*. Now, we show Générale Ressorts as a composite, revealing its internal structure with the service systems: *ERP[w]* and *OrderEntryPerson[w]*. The properties that were in *Générale Ressorts[w]* are now distributed to the new service systems *ERP[w]* and *OrderEntryPerson[w]* based on the decisions captured in the matrices.

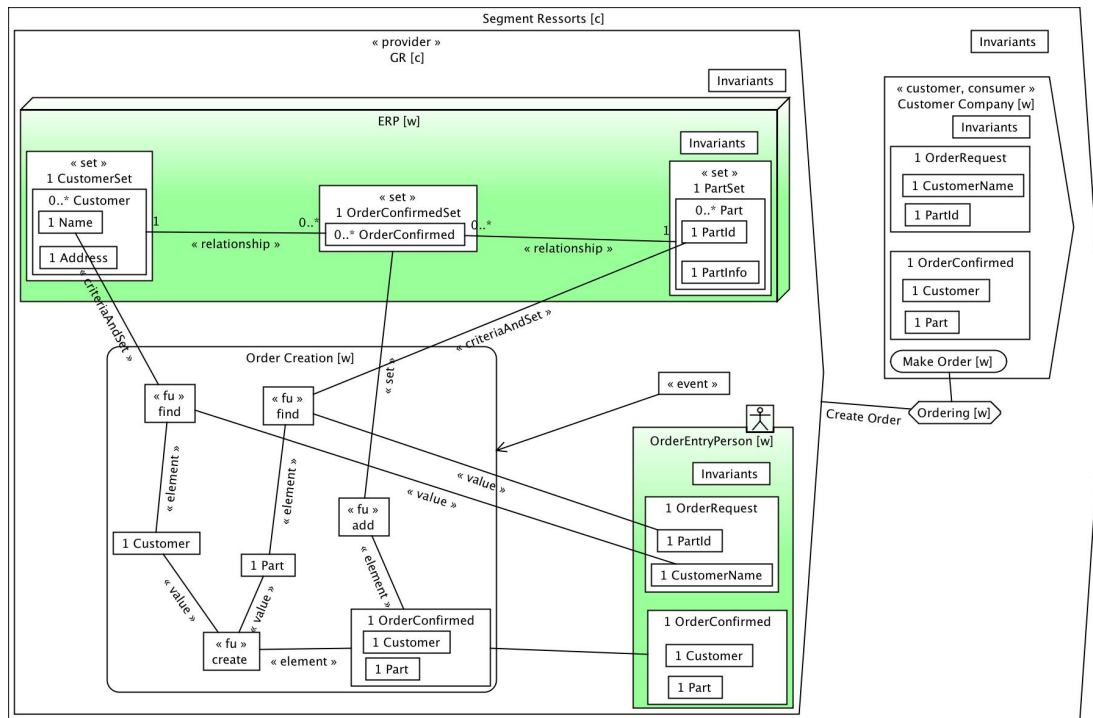


Figure 34: The result of step 2

6.2.4 Step 3: Internal services

Once we have identified the internal organization of Générale Resorts that participates in the implementation of the service *Create Order[w]*, we need to define what internal services needed in Générale Resorts are in order to provide externally visible services, in this case *Create Order[w]* only. Therefore, business and IT analysts need to identify those internal services. Also, they need to distribute the semantics of the service *Create Order[w]* in a form of functional units to the newly defined services. Similarly with refining the service systems, when we refine the service, it can be possible that some new functional units are added, which were not visible when we modeled only the service as a whole. In this case, they remain the same and are allocated to the newly identified internal services as shown in Figure 35 and Figure 36.

Internal collaborations
FindCustomer
FindPart
CreateOrderConfirmed

Figure 35: Step 3: Conceptualize - Identifying internal collaborations

	FindCustomer	FindPart	CreateOrderConfirmed
find (Customer...)	X		
find (Part...)		X	
create (OrderConfirmed..)			X
add (OrderConfirmed..)			X

Figure 36: Step 3: Decide - Allocating functional units to collaborations

Figure 37 shows the result of step 3. As we can notice the service *Create Order[w]* is refined, and the rest of the diagram remains unchanged. We show new identified services: *FindCustomer[w]*, *FindPart[w]* and *CreateOrderConfirmed[w]* with their events, meaning that *ERP[w]* and *OrderEntryPerson[w]* are in charge together to collaborate and provide the identified services. Existing functional units are distributed to the new services according to the decision captured in the matrices.

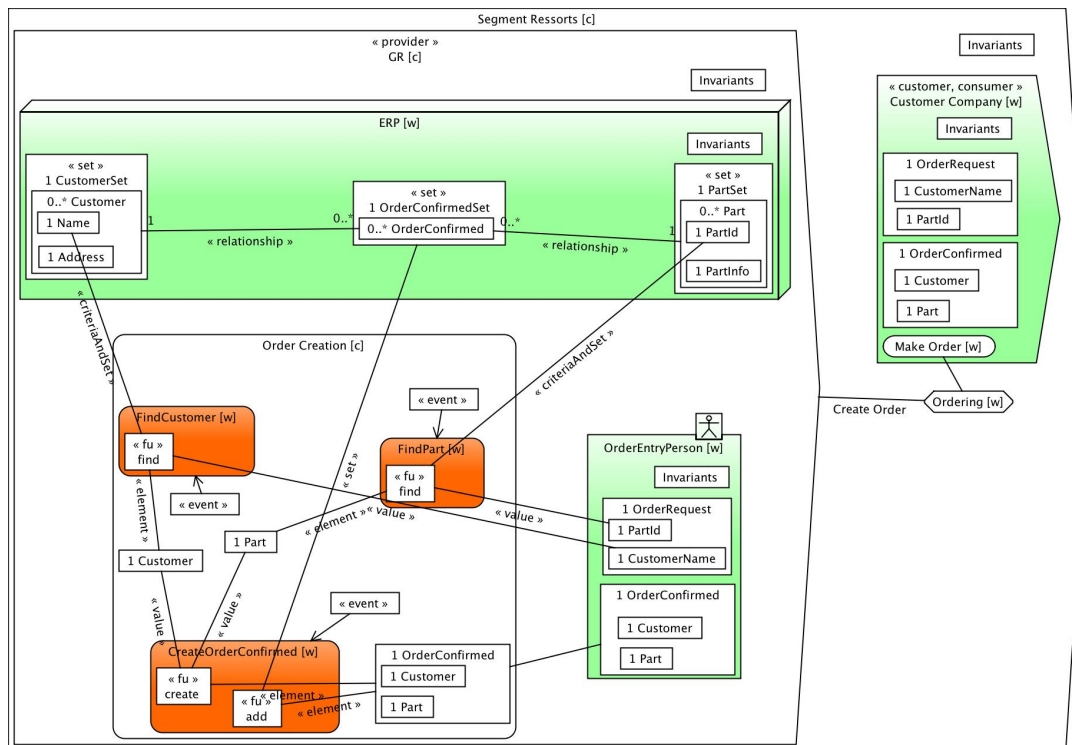


Figure 37: The result of step 3

6.2.5 Step 4: Internal responsibilities

So far, we have identified the internal organization of Générale Ressorts that participates in implementing the service *Create Order[w]* and internal services that should be provided to implement the service *Create Order[w]* for the *Customer Company[w]*. Now, we need to define the responsibility of each of organizations in Générale Ressorts in providing the internal services. Therefore, for each internal

service, we define the process and the participating provider's roles based on the RACI (responsible, accountable, consulted, informed) matrix, in this case some of *ERP[w]* and *OrderEntryPerson[w]* as well as their responsibility (who is in charge of operating the service). For each process, we have exactly one responsible organization. In the diagram this organization will have the service with the event, meaning that that person is responsible for initiating the service. This way, we define who exactly participates in providing certain service and in which role. We also need to distribute the functional units of the service to the services of the roles participating in implementing it. These decisions are captured in Figure 38 and Figure 39.

Processes	Services
FindCustomer	ERP(R), OrderEntryPerson(A)
FindPart	ERP(R), OrderEntryPerson(A)
CreateOrderConfirmed	ERP(R), OrderEntryPerson(A)

Figure 38: Conceptualize - Identifying processes and services

	ERP.CreateOrderConfirmed	ERP.FindCustomer	OrderEntryPerson.FindCustomer	...
find (Customer...)		X		
find (Part...)				
create (OrderConfirmed..)	X			
add (OrderConfirmed..)	X			

Figure 39: Decide - Allocating functional units to services

Figure 40 shows the result of step 4. As we can see, the changes in the diagram are in *Générale Ressorts[c]*. For each service in Figure 37, the new process is defined and is related to the services of the roles participating in it. These services contain functional units, based on the decision captured in matrices in Figure 39.

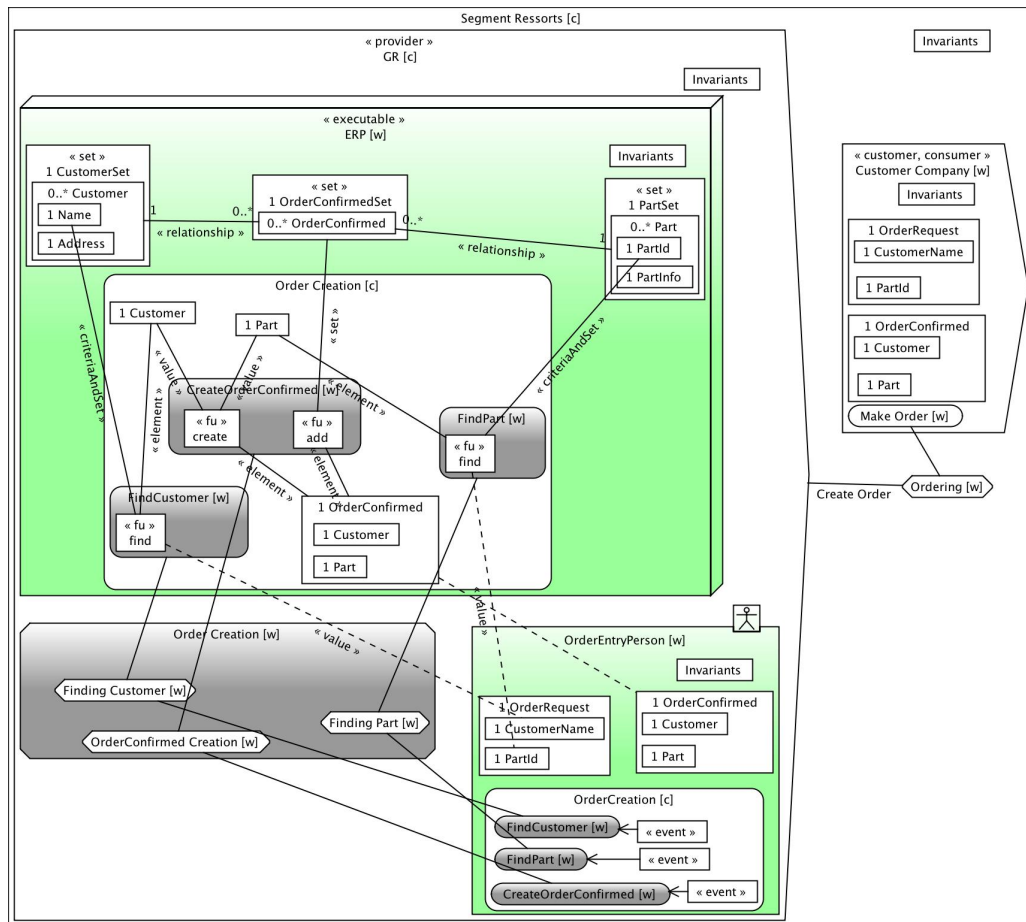


Figure 40: The result of step 4

6.2.6 Step 5: Independent service systems (optional)

After step 4, we have the necessary information for the service implementation. In some other projects, we might need to refine the organization further to reach the level of abstraction needed for that project. In the case of order creation in Générale Ressources we have all the necessary information: what externally visible services that should be provided are, what internal services required to provide the externally visible services are, what is the internal organization of Générale Ressources needed to implement the services, and what are the responsibilities of each internal organization in Générale Ressources in implementing the identified services.

In step 5, we can optionally split the internal organization in Générale Ressources to create independent systems, such that every system has all the information it needs for implementing its services. This can be useful, for example for prototyping, where we can extract IT system only and map it to the application prototype. In order to do this, we need to cut the lines connecting different systems and specify send and receive properties and information about where they come from. This way, we replace external systems with the properties coming from them. The decisions are captured in Figure 41. We can say the interfaces of the service systems are defined, by stating what properties are sent or received from/by the service system.

	Générale Ressorts Shared Data	Customer Company Shared Data
OrderRequest	X (R)	X (S)
OrderConfirmed	X (S)	X (R)

Figure 41: Step 5: Cutting connecting lines and specifying shared properties

Figure 42 shows the result of step 5. It shows the organizations in Générale Ressorts without their relations. It shows also their properties shared with the other system and the information on where they come from. Received and sent stereotypes are used for properties coming from external system and internal properties shared with the external systems, respectively. This way, it makes $ERP[w]$ and $OrderEntryPerson[w]$ independent of each other. For example, $ERP[w]$ has all the information that it needs for implementation of its services and information on where they come from.

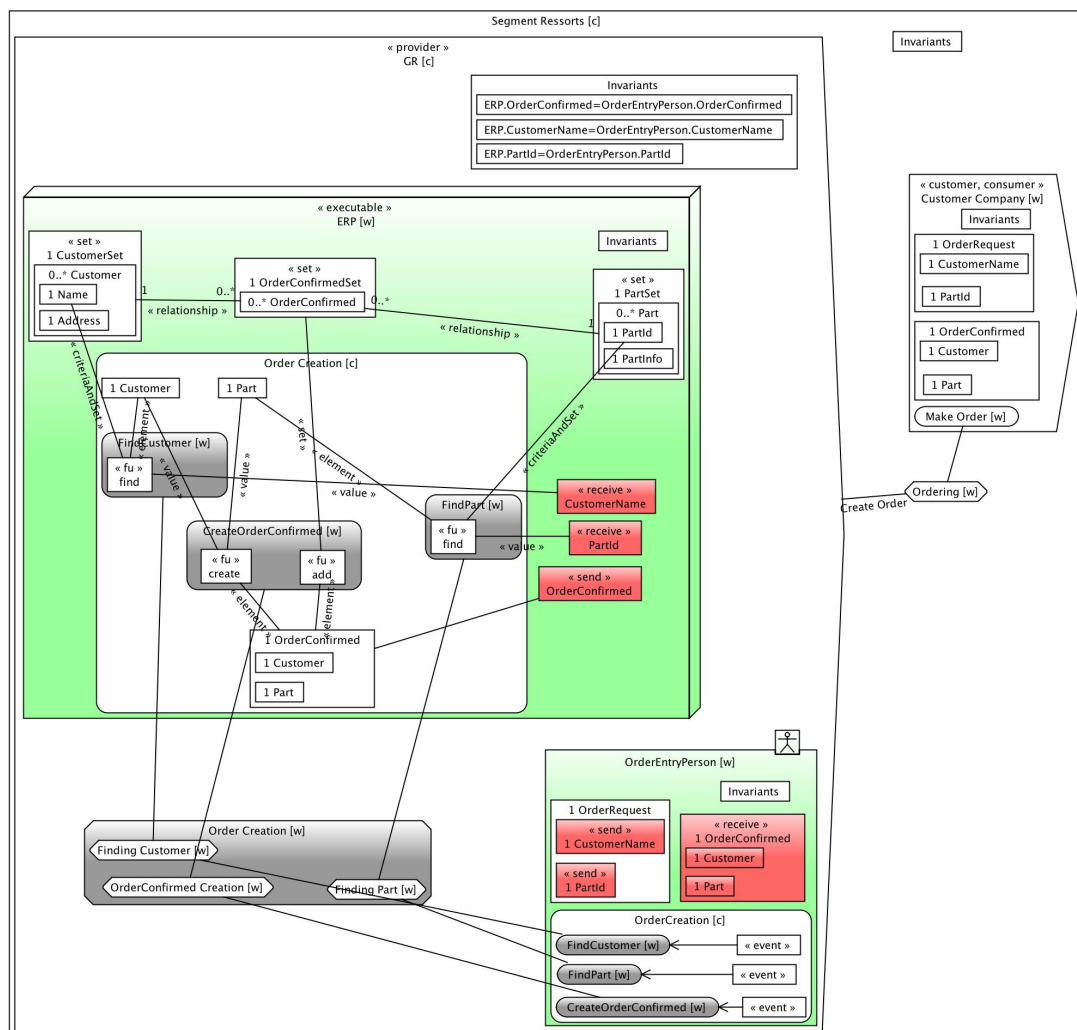


Figure 42: The result of step 5

As we can notice, in this spiral, we start with modeling very few elements of the service system. Traditionally, this is not done in the projects. We believe this can bring a lot of value, as it enforces thinking about high-level concepts and consumer needs. In the beginning of the process, business and IT analysts are enforced to think in terms of external stakeholders needs. Once that is defined, analysts decide the details of service implementation including service-based internal organization and their responsibilities. By service-based internal organization we mean that traditional organizational hierarchy is not used, but the teams are defined based on their role in providing the service. This means we can have people from different departments in the same team, or external organizations as well. This way, the process enforces business and IT analysts to think about the “big picture” and therefore involves more business experts in the service specification. The idea is that business and IT analysts can work together using this spiral to create service specification.

6.3 Validation and Verification Spiral: Case of Order Creation at Générale Ressorts

We illustrate the service validation approach with the example of Order Creation at Générale Ressorts. To illustrate the validation approach, we include as well the order delivery and order payment, as described in Chapter 6.1.

6.3.1 Initial Model Design

We define a model of a service system in Alloy: we specify its data structures, the initial predicate, business rules and make initial assumptions about our model defining model invariants. These invariants replace the properties required by the business specification and are used to control the model complexity. The model can be written manually in Alloy, or can be automatically generated from the model using our transformation tool. Here, we give an example of manually created code, which is easier to read.

For business rule specification, we use the following procedure: The business analyst specifies the BR in a natural language; the designer classifies the BR according to their scope and nature (see Section 3.2) and translates them to Alloy specification language. Together with the whole system of interest (a process, an activity, etc.) in Alloy, he can detect how the business rule influences the system behavior.

The data structure for the “Order Creation” service is modeled using Alloy signatures:

```
abstract sig GR {
    orderConfirmedSet: set Order,
    orderDeliveredSet: set Order,
    orderPaidSet: set Order,
    partSet: set Part,
    customerSet: set Customer
}
```

```

one sig GR_pre extends GR {
    orderRequest: one OrderRequest
}

one sig GR_post extends GR {}

```

Alloy signatures (sig) can be abstract or concrete, can have explicit cardinalities (e.g., only one OrderRequest object can be treated by the service at a time), and can contain one or multiple fields (as classes and attributes in object-oriented (OO) languages). We can also define additional constraints on the initial data structure with the invariants.

We express behavior in terms of a state transition: we define a pre-state that describes the state of a system before the service has been performed and the post-state that describes the condition that must hold for the system upon the service termination - the service result. Note that following the declarative modeling paradigm, we do not specify how the service will change the system's state. We model the "Order Creation" service as a corresponding predicate in Alloy.

```

1. pred orderCreation(aGR_pre: one GR_pre, aGR_post: one GR_post){
2. one aCustomer: Customer | one aPart: Part | one aOrderConfirmed: OrderConfirmed |
3.
4. aPart=findPartByPartID[aGR_pre.orderRequest.requestedPartID,aGR_pre.partSet] and
5. aCustomer=findCustomerByName[aGR_pre.orderRequest.name,aGR_pre.customerSet] and
6. aOrderConfirmed=createOrderConfirmed[apart,aCustomer] and
7. aGR_post.orderConfirmed=aOrderConfirmed and
8. aGR_post.orderConfirmedSet=aOrderConfirmed + aGR_pre.orderConfirmedSet}

```

This predicate shows a transition between Générale Ressorts (GR) pre and GR post states; these states are indicated as predicate parameters (line 1). In this predicate, the variables are declared (line 2), the customer and the part are found in the set (lines 4-5) and the order is created (line 6), returned as outcome (line 7) and added to the set (line 8), as described in the case study.

6.3.2 Step 1: Generate Samples

We use the Alloy Analyzer tool to run the predicate and generate the samples that satisfy all the constraints of the model. By analyzing the samples, we detect "Missing Customer" anomaly. Figure 43 illustrates this anomaly: in a pre-state we have Customer0, in a post-state we have Customer1. As we show exactly one execution of the service "Order Creation", we expect both the customerSet and the partSet to remain the same in pre- and post-state. However, the generated instance suggests the opposite.

NOTE: the inputs and outputs in our diagrams (e.g., OrderRequest and OrderConfirmed in Figure 43) are depicted with black rectangles; customer data (Customer, Name, Address) and part data (Part, PartID, PartInfo) are depicted with parallelograms and diamonds, respectively. We depict the pre-state (prior to the order creation service execution) and post-state (upon the service termination) of the GR company with "houses" and the corresponding labels: GR pre, GR post.

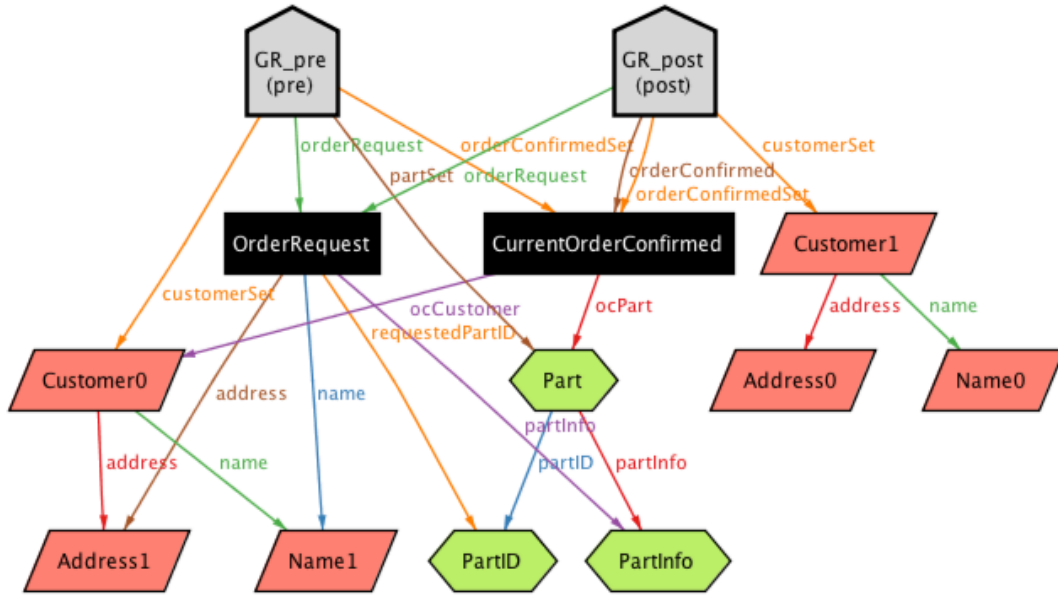


Figure 43: Anomaly due to Underspecification: “Missing Customer”

This anomaly indicates that some constraints, which should prevent the customer set and the part set from changing during the service execution, have to be specified. Thus, it is an anomaly due to the underspecified model.

6.3.3 Step 2: Correct Model Based on Samples

In fact, the declarative specification principles oblige us to explicitly state the elements that must remain “unchanged” during the state transition. Therefore, we need to add an invariant that states that the customerSet in post-state is the same as the customerSet in pre-state. The same applies to part set.

```
fact customerSetSame {
    GR_post.customerSet = GR_pre.customerSet
}
```

In order to validate that we have resolved the “Missing Customer” anomaly, we create an Alloy assertion that claims that for all Order Creation executions (i.e., model instances), the customer set will remain the same in pre- and post- states of GR.

```
customerPrePostSame: check{
    all aGR_pre: GR_pre, aGR_post: GR_post |
    orderCreation[aGR_pre,aGR_post] =>
        aGR_post.customerSet=aGR_pre.customerSet
}
```

Checking this assertion, we find no counterexamples.

```
Executing ‘‘Check customerPrePostSame’’ Solver=sat4j Bitwidth=4 MaxSeq=4
Symmetry=20 1014 vars. 109 primary vars. 1750 clauses. 32ms.
No counterexample found. Assertion may be valid. 12ms.
```

This confirms the assertion validity.

6.3.4 Step 3: Check Assertions

We make assertions about our model in order to test some desirable properties and business rules. Alloy Analyzer validates our assertion by searching for a counterexample: a model instance for which our assertion does not hold. If no such counterexample is found, then our assertion is valid within a given value domain. In the opposite case, the model has to be revised.

We check the validity of each of the business rules from Chapter 6.1, using Alloy assertions. We show an example of BR4 validation (“no faulty delivery”).

To ensure “no faulty deliveries” (BR4), we check that the customer and part data in the confirmed order are exactly the same as in the requested order. The assertion “orderConfirmedCorrect” is defined to validate this BR:

```
orderConfirmedCorrect: check{
  all aGR_pre:GR_pre, aGR_post:GR_post, oReq:OrderRequest,
  oCurrent: CurrentOrderConfirmed |

  orderCreation[aGR_pre,aGR_post]=>
  (oCurrent.ocCustomer.name=oReq.name and
  oCurrent.ocCustomer.address=oReq.address and
  oCurrent.ocPart.partID=oReq.requestedPartID and
  oCurrent.ocPart.partInfo=oReq.partInfo)
}
```

When we run the assertion, we obtain counterexamples. Figure 44 shows an example of an incorrect delivery: the order is created on the correct customer’s name, but the delivery address associated with this name does not correspond to the address provided in the OrderRequest. Therefore, the part can be delivered to the wrong address. The anomaly observed is due to model underspecification.

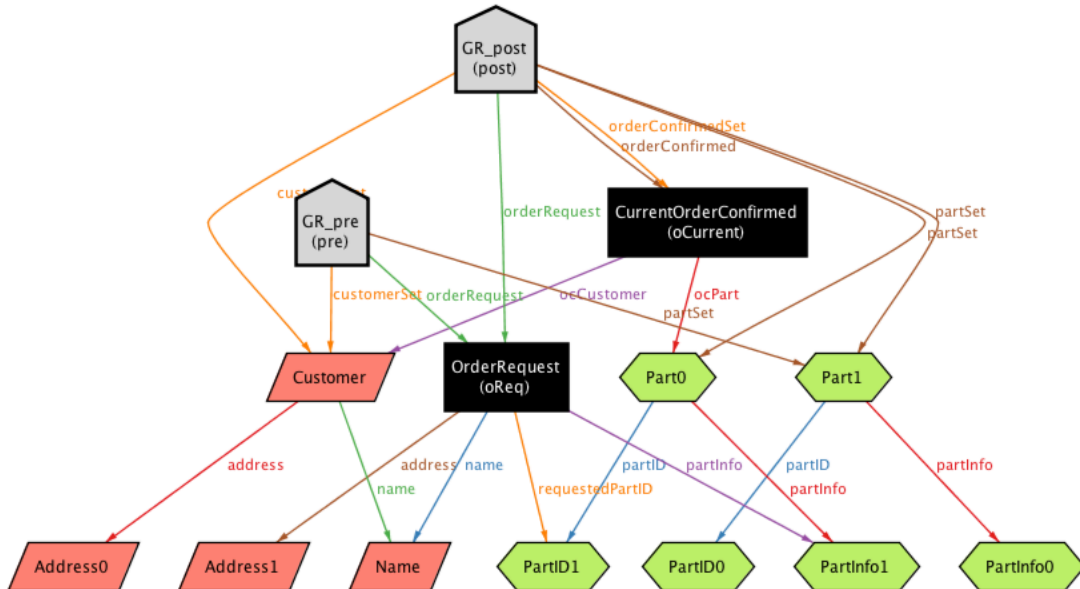


Figure 44: Anomaly due to Underspecification: “Delivery to the Wrong Address”

6.3.5 Step 4: Correct Model Based on Assertions

In order to resolve the detected anomaly, we add a new invariant "noOldAddress" that states that we cannot have a customer in the system with the name given in the requested order, but with an old/invalid address and vice versa:

```
fact noOldAddress{  
    all c:Customer |  
        c.address=OrderRequest.address<=>c.name=OrderRequest.name  
}
```

If we check now the assertion "orderConfirmedCorrect", we get the result "No counterexample found. Assertion may be valid", meaning that this assertion holds in a given domain, and all orders will be delivered to the correct customers at the correct addresses.

We continue "debugging" the model by running the simulations, checking if we have introduced some new unwilling behavior. We repeat the process for other BRs. After validating all BRs and finding no anomalies, we conclude that the designed model meets its business needs at a given level of details.

7 BASS2Alloy and Back: Transforming BASS Service Model to Alloy and Back

In this chapter, we explain how BASS service models are transformed to Alloy for the purpose of validation and verification. Then, we explain the transformation from Alloy back to BASS, making the results of simulation more understandable to people already familiar with BASS notation.

There are two steps to be done in a simulation process as shown in Figure 45:

1. Transforming model to the Alloy code.
2. Generating model instances and counterexamples to show when the properties do not hold.

The second step is done automatically using features of the Alloy Analyzer tool. Alloy Analyzer is a solver that takes Alloy model constraints as input and finds structures that satisfy them. It can be used both to explore the model by generating sample structures, and to check properties of the model by generating counterexamples. Structures are displayed graphically and can be customized. In this chapter, we explain how step one of this process is done.

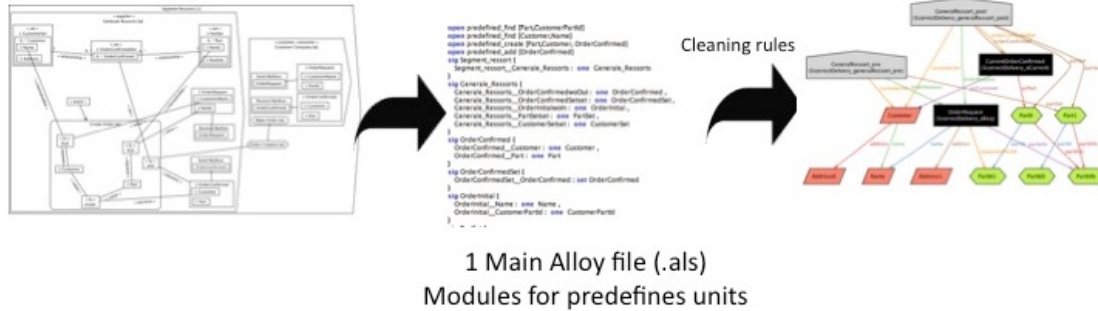


Figure 45: Simulation process overview

First we explain the input and output of the simulation. Then, we explain the transformation of BASS model to Alloy specification. Next, we explain how Alloy is used to simulate the behavior of the model. Finally, we describe how Alloy instances are transformed back to BASS, to make the results of simulation more understandable for those already familiar with BASS notation. The work in this chapter is described in [123].

7.1 Input and Output of Simulation

Input of the simulation is any model described with the meta-model in Figure 28. The output of simulation is one instance of the model satisfying the constraints of the model or the counterexample showing when checked property does not hold. The output is shown graphically with boxes and arrows showing the state of the system before and after the service execution. An example of simulation input and output for Order Creation at GR is shown in Figure 46.

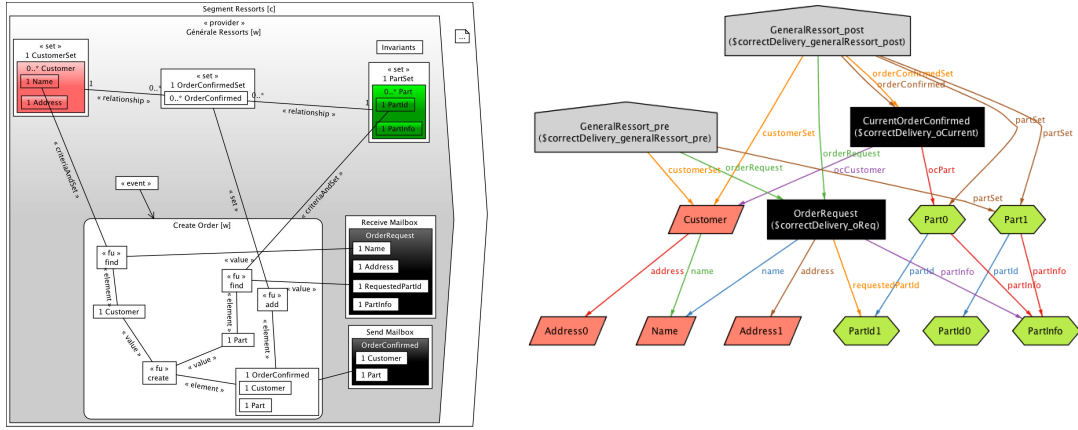


Figure 46: Input and output of simulation for Order Creation at GR

The simulated system, in this Générale Resorts is mapped to its pre- and post-service execution state. The simulated system is marked with grey houses. Instances of the properties with their attributes are shown with other boxes. The relation between elements is shown with lines. Based on the lines, it is possible to see what elements are contained in the pre-state and what in post-state of the system and to check if this corresponds to expected behavior. Mapping between input and output elements is given in Table IX.

Table IX: Mapping input and output of simulation for Order Creation at GR

Input (model)	Output (model instance)
Simulated service system with simulated action (in this case Générale Resorts)	Two boxes (in this case grey houses) showing the state of the system before (pre) and after (post) service execution
Properties and its attributes	Instances of the properties shown with boxes with names PropertyName1, PropertyName2, etc.
Relation between systems, properties and its attributes	Lines between the boxes
Behavior with its functional units	Logic of the service that affects the system and makes it change from its pre-state to post-state

Now that we have explained the input and output of the simulation, we are going to explain how this model is transformed to Alloy specification and how the simulation is done.

7.2 Transforming BASS Model to Alloy Specification

In order to transform a BASS model to the Alloy specification, we define the correspondence between the proposed method and Alloy meta-model elements. Then, we explain how functional units are used to represent service semantics in Alloy. We illustrate the transformation with the example of Order Creation at Générale Resorts. We explain only the first model from the GR service specification spiral (Figure 31).

The full Alloy specification for all models can be found in **Appendix III: Case of Order Creation at Générale Ressorts in Alloy**.

One part of Alloy meta-model showing the concepts related to modeling the signatures is shown in Figure 47 (from [65]). Besides these signature-related concepts, we also use the concepts such as predicate, function, etc. to map the concepts from our meta-model. We map the elements from the BASS meta-model (Figure 28) to the Alloy meta-model elements. The correspondence between the meta-modeling elements is shown in Table X.

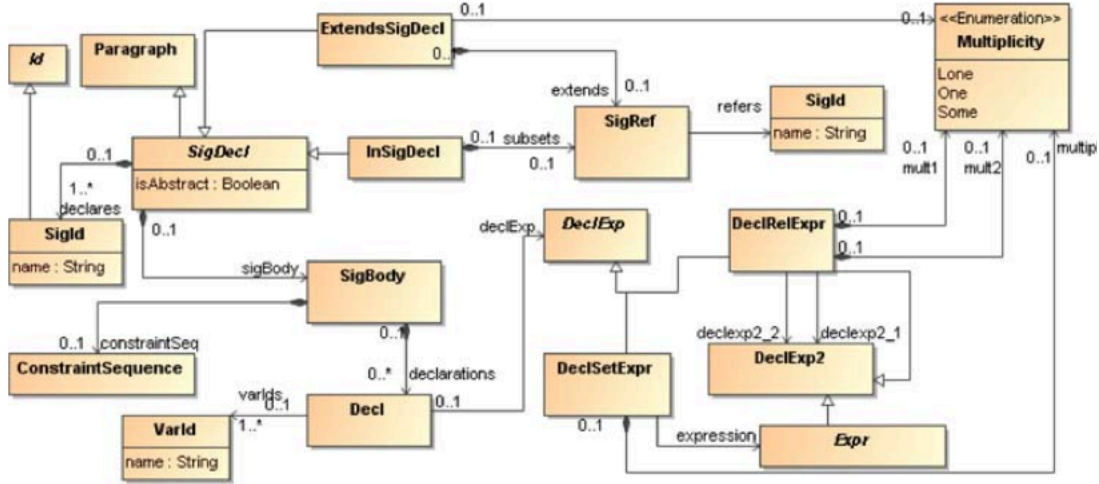


Figure 47: Alloy meta-model for signatures (from [65])

Table X: Correspondence between BASS method and Alloy meta-model elements

BASS meta-model elements	Alloy meta-model elements
Model Name	ModuleHeader
Service System	ExtendsSigDecl
Service	Predicate
Collaboration	Predicate
Process	Predicate
Functional Unit	Function
Predefined Functional Units	Module (library) with parameterized functions
FU Parameter	Decl
Property	ExtendsSigDecl
Property Value	Decl
Property Type	ExtendsSigDecl
Multiplicity	Expr
Invariant	Predicate/Function, depending on the context

As we can notice, service systems and properties are mapped to the corresponding elements of signature-related concepts in Alloy. Invariants and functional units are mapped to the predicated and functions depending on the context in which they are applied. If the constraint holds in the whole system, then an invariant becomes the fact. Otherwise, it is a contextual invariant, and holds only in a certain context. Therefore, it is expressed as predicate. Functional units are expressed with predefined functions in Alloy library (module). Whenever the new customized functional unit is added to the environment, we also need to define Alloy function representing its semantics for it. Then, we can use any of the functional units defined in the functional unit library for any other project as well.

Based on the mapping and functional unit's library, we can transform our model to the Alloy specification. The pseudocode for such transformation is:

```
for one BASS model (.json file)
    load node set and edge set
    transform systems and properties with corresponding edges to Alloy
    signatures
    transform services to Alloy predicates using the library of predefined
    functional units
    transform collaborations to Alloy predicates using the library of
    predefined functional units
    transform invariants to Alloy facts and predicates
```

The result of such transformation applied to the Order Creation at Générale Ressorts (Figure 31) looks like this:

Transformation of service systems and properties

```
sig Segment_ressort {
    Segment_ressort__Generale_Ressorts: one Generale_Ressorts
}

sig Generale_Ressorts {
    Generale_Ressorts__OrderConfirmed: one OrderConfirmed ,
    Generale_Ressorts__OrderConfirmedSetset: one OrderConfirmedSet ,
    Generale_Ressorts__OrderRequest: one OrderRequest ,
    Generale_Ressorts__PartSetset: one PartSet ,
    Generale_Ressorts__CustomerSetset: one CustomerSet
}

sig OrderConfirmed {
    OrderConfirmed__Customer: one Customer,
    OrderConfirmed__Part: one Part
}

sig OrderConfirmedSet {
    OrderConfirmedSet__OrderConfirmed: set OrderConfirmed
}

sig OrderRequest {
    OrderRequest__Name: one Name ,
    OrderRequest__PartId: one PartId
}
```

```

sig PartSet {
    PartSet__Part: set Part
}

sig Part {
    Part__PartId: one PartId,
    Part__PartInfo: one PartInfo
}

sig CustomerSet {
    CustomerSet__Customer: set Customer
}

sig Customer {
    Customer__Name: one Name,
    Customer__Address: one Address
}

sig PartId { }
sig PartInfo { }

sig Name { }
sig Address { }

```

Transformation of behavior (in this model Order Creation service)

```

pred      simulate(Segment_ressort_pre,      Segment_ressort_post: one
Segment_ressort, Part1: one Part, Customer2: one Customer, OrderConfirmed3:
one OrderConfirmed, OrderConfirmedset4: set OrderConfirmed)
{
    //find part by partId, with the value OrderRequest.partId in partSet
    Part1 = find[Part__PartId,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__Or
derRequest.OrderRequest__PartId,
    Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__PartSetset.PartSet__Part,
    Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Resso
rts__PartSetset.PartSet__Part]

    //find customer by name, with the value OrderRequest.name in customerSet
and Customer2 = find[Customer__Name,
    Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OrderRequest.OrderRequest__Name,
    Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__CustomerSetset.CustomerSet__Customer,
    Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Resso
rts__CustomerSetset.CustomerSet__Customer]

    //create orderConfirmed with the found part and set
and OrderConfirmed3 = create[Part1, Customer2, OrderConfirmed__Part,
OrderConfirmed__Customer]

    //add orderConfirmed to the orderConfirmedSet
and OrderConfirmedset4 = add1[OrderConfirmed3,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__Or
derConfirmedSetset.OrderConfirmedSet__OrderConfirmed]
And
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
rderConfirmedSetset.OrderConfirmedSet__OrderConfirmed
= OrderConfirmedset4

```

```

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
rderConfirmed
= OrderConfirmed3
and Segment_ressort_pre not= Segment_ressort_post
}

```

This represents only one part of the generated Alloy specification. The first part with signatures corresponds to the mapping of the service systems and properties. The predicate corresponds to the Order Creation service in Générale Resorts. The full Alloy code for this example and for other GR models in the service specification spiral is in **Appendix III: Case of Order Creation at Générale Resorts in Alloy**.

In addition to transforming model elements to the Alloy code, we need to add additional constraints to get clearer, more visually appealing results. As Alloy is declarative language, everything that is not explicitly stated, even if it appears as obvious, is possible to happen. For example, if we model only finding the element in the set, and do not state that the set remains unchanged, it is possible that as a result we get instance where the set has more or less elements after finding the element that before. In order to avoid such unexpected behaviors, and to get clean models, without elements not related to any other element, we need to add the following model cleaning and consistency rules:

1. If element E1 contains the element E2, then the parent of element E2 is element E1. Element can be service system, service, collaboration, process, property or any other element of the meta-model that has the aggregation relation with the other elements. For example, if service system as a composite E1 contains service system as a whole E2, then the parent of service system E1 is service system E1. Similarly, the rule applies to the relation between service system as a composite and another service system as a composite, service as a composite and its component services, service and its functional units, and all other elements related with aggregation relationship.
2. There is exactly one element that does not have a parent, and it is the root element of the model. This means, that there are no “floating” elements that are not related to the rest of the model.
3. Property P remains the same before and after the service execution, unless stated differently. This ensures that the constraints we assume to hold when we model in an imperative environment do hold, such as already mentioned example of finding element in the set.

We can also use Alloy to verify if the model conforms to the meta-model and follows the well-formedness rules. For that purpose, we have transformed our meta-model with the well-formedness rules to the Alloy specification. When this specification is included in the Alloy specification of the model, we can check if the model follows the constraints given in the meta-model specification. In case there are some contradictions, Alloy generates the message informing us about it.

To sum up, there are 4 parts to be transformed during the transformation process:

1. Model elements
2. Consistency and cleaning rules
3. Meta-model
4. Well-formedness rules.

Once we transform all of them, we can verify if the model satisfies the meta-model and well-formedness rules and validate if the generated instances correspond to the expected behavior of the service system.

7.3 Service Simulation

Once we have transformed the model into Alloy specification, it can be simulated using the features of the Alloy Analyzer tool. The Alloy Analyzer tool is using its SAT solvers to find all the instances of the model that satisfy all the constraints given in the Alloy specification. An example of such instance for the model in Figure 31 is shown in Figure 48. Note that when the model is automatically generated, the shapes and colors are not customized. For the purpose of explanation of the meaning of Alloy diagram, we show already customized model.

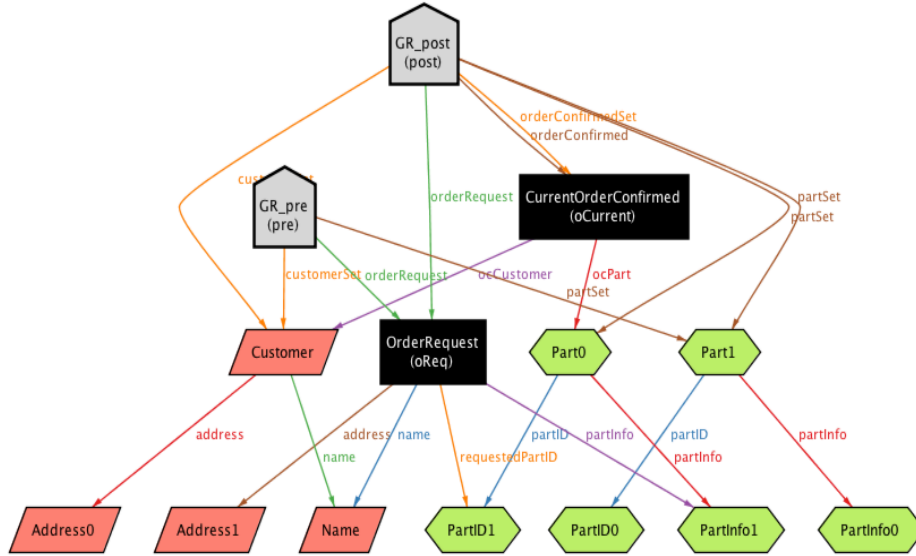


Figure 48: Order Creation at Générale Ressorts - Example of Model Instance

The result of simulation shows the state of the company before and after the Order Creation. The elements of the company can be found following the lines from *GR_pre* and *GR_post*. We can notice that the *CustomerSet* and *PartSet* remain unchanged, but the *CurrentOrderConfirmed* has been created in post-state referring to the customer and part that corresponds to the customer name and part ID from *OrderRequest*. This way we can always analyze the state of the system before and after the service has been performed and detect possible anomalies. In this case, we can see that the *OrderConfirmed* is created for the customer with the name *Name* from *OrderRequest*, and address *Address0* that does not correspond to the address from *OrderRequest Address1*. This might lead to wrong address delivery.

7.4 Transforming Alloy Instance to BASS Model

The model instances generated by the Alloy Analyzer tool are shown not to be easily readable by the business experts who are not that familiar with the Alloy language and visual notation. Therefore, to facilitate understanding of the results of simulation, we have created the tool to transform these model instances back to BASS models. The pseudo code for this transformation is:

```
for one Alloy model instance
    load instances and relations to tree
    traverse tree and generate json file
    write file
```

An example of BASS instance generated with Alloy2BASS is shown in Figure

49.

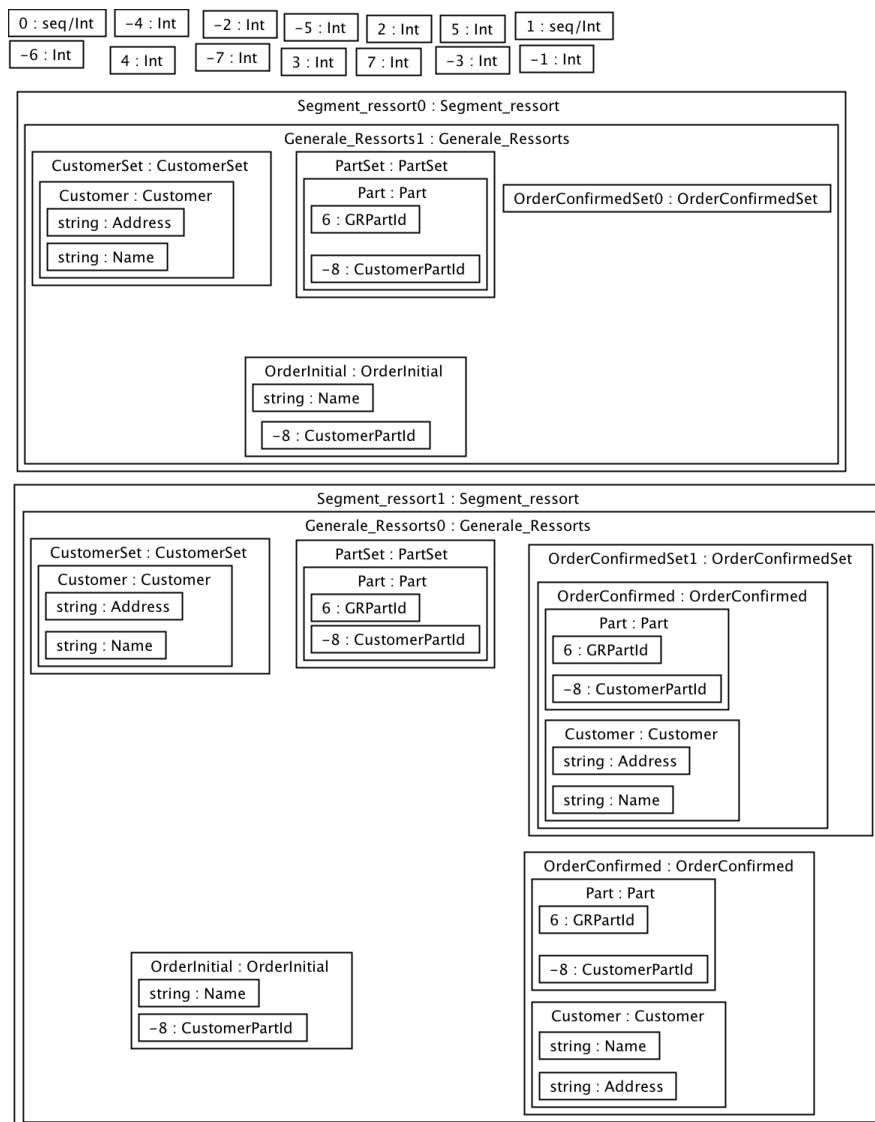


Figure 49: An example of model instance shown in BASS

8 BASS2Java: Transforming BASS Service Model to Application Prototype

In this chapter, we explain how service specification modeled in BASS can be transformed to application prototype using Arcimboldo approach [112]. We choose Arcimboldo for prototyping, because it is based on templates that enable adding another level of abstraction before going directly to code, making the transformation from model to code simpler.

Figure 50 shows an overview of BASS to Java transformation. It works in two steps:

1. Transforming service specification in an intermediate project containing Arcimboldo files in the given target language: one main object, one descriptor file, many templates and many other files that need to be copied. We refer to this project as Arcimboldo project. The whole transformation is split in three main parts that correspond to different parts of application: for generating Graphical User Interface (GUI), data-related elements and logic of application.
2. Combining templates, the main object and descriptor file and generating application in the given target language. In this thesis, we generate Java Enterprise Edition (JEE) application using Java Server Faces (JSF) for interface, MySQL database with Hibernate for data and Java files for expressing the logic.

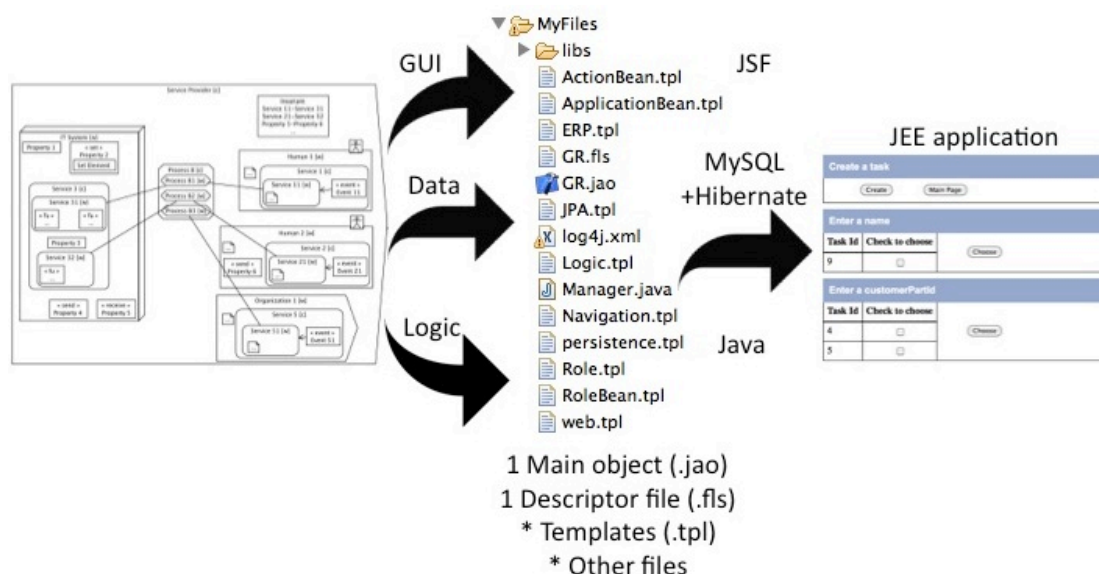


Figure 50: Overview of BASS to Java transformation

The second step is done automatically by Arcimboldo environment assuming that the necessary files are available as explained in Chapter 3.4. In this chapter, we first explain input (structure of service specification that can be prototyped) and output

(the result) of prototyping. Then, we focus on explaining the details of step one. We explain overall structure of generated Arcimboldo project explaining what files it contains and what their relation to service specification is. Next, we explain generation of GUI. Then, we explain transformation from static elements of the service specification (data) to the Arcimboldo project files. Finally, we explain transformation of dynamic elements of service specification (logic). Here, we also explain how we deal with simultaneous execution of services. We illustrate all transformations with the example of Order Creation at Générale Ressorts (Chapter 6.1).

8.1 Input (Service Specification Template) and Output (Java Application) of Prototyping

In Chapter 4.2 we show the meta-model for the proposed method. In order to prototype service specification, we need to know the responsibilities of service systems participating in providing the service. Therefore, only models at the certain level of abstraction, in specific format can be prototyped. Thus, first we explain what kind of models it is possible to do prototyping for. We show one model representing the template for prototypical specification.

Input of the prototyping is the model based on the meta-model described in Figure 28 with the following constraints: Prototyped service system is always service system as a composite (white box) containing exactly one executable service system and many non-executable service systems related with a process; all of them can be human, IT or organization service systems. Executable service system is the one that we want to prototype, i.e. the one whose services and properties we transform. In the diagram it is the service system annotated with the stereotype `<<executable>>`. Non-executable service systems are all other service systems interacting with the executable service system with the aim to provide the modeled service. From those systems we use only information about the events corresponding to the responsibilities for each service execution and shared data (send/receive properties) corresponding to the GUI fields for the given service. Note that instead of executing IT system only, like it is the case with code generation methods, we execute any service systems annotated with `<<executable>>`. This follows the systemic paradigm, on which the method is based, which treats all systems the same way, independent of their nature. Therefore, we execute equally human, organization and IT systems.

Model representing the prototypical specification template is shown in Figure 51. It contains the main segment with service provider, service consumer and other stakeholders as described in the template in Chapter 4.1.1. Prototyped service system is either service provider or one of the sub-systems of service provider that we want to prototype. It contains exactly one executable system, in this case IT system, and many non-executable systems, all as a whole. It is not possible to prototype the service

system that contains other service systems as a composite, i.e. we allow only one level of hierarchy for the prototyped service system.

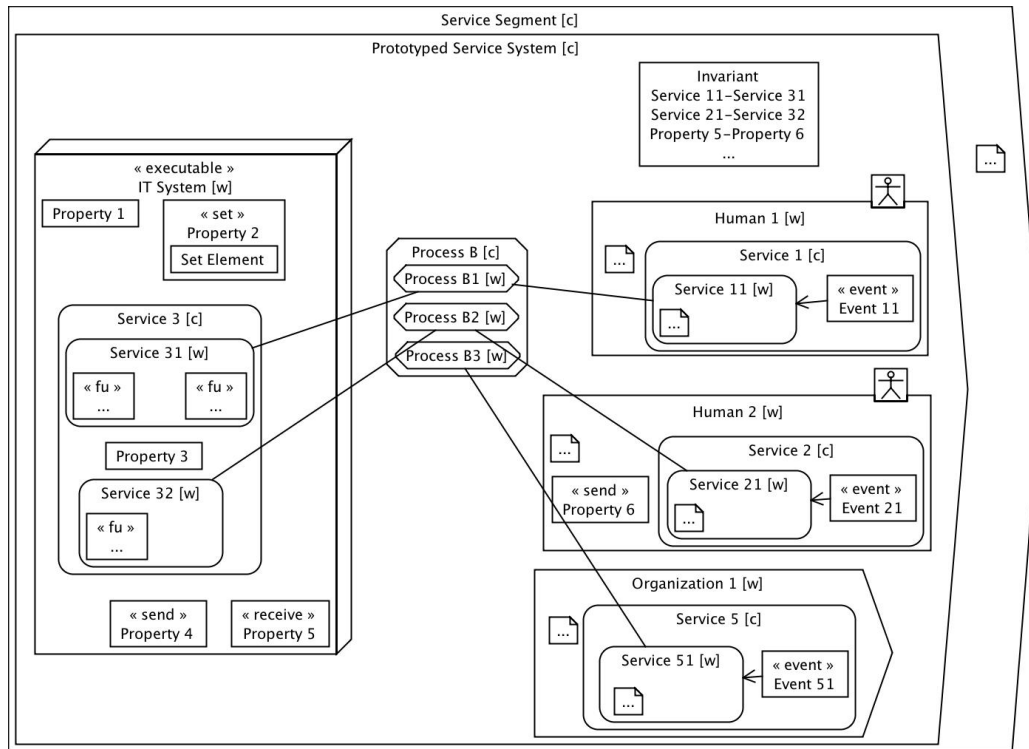


Figure 51: Input: Prototypical specification template

Component service systems of prototyped service system all contain services and properties. There are several types of properties:

- Simple property – data representing one object, such as *Property 1* and *Property 3*. Their persistence and duration depends on the context in which they are defined. Properties that are defined in executable system, such as *Property 1*, last as long as the system lasts, and are stored permanently, for instance in database. Properties defined in service as a composite, such as *Property 3*, last as long as one transaction of service and therefore are not stored permanently in IT system, but in the table capturing all transactions of *Service 3*.
- Set property – data representing set of objects, such as *Property 2*. It is marked with stereotype <<set>>. Their persistence and duration depends as well on the context in which they are defined.
- Send/receive property – data shared with other systems, such as *Property 4* and *Property 5*. They are marked with stereotypes <<receive>> and <<send>>. The relation between them is captured in invariant of the process connecting services of different service systems shown in the upper right corner of the diagram.

In addition to properties, service systems contain service as a composite corresponding to the long-running business transaction. They can contain services as

a whole or properties needed to communicate between them. Services as a whole contain functional units describing logic of the service. They are related to the events, showing who is in charge of initiating the service. For one process connecting many services, there is exactly one event related to one of the used services. The model following this template can be then used for prototyping. We use all elements of executable system and only events and shared data (send/receive properties) from non-executable systems for prototyping. This way, we replace the non-executable system only with the data that are coming from them. As a part of future work, simulation of the behavior of the other systems can also be done.

Output of the prototyping is the application in the given target language. In this thesis, we generate JEE application using JSF for interface, MySQL database with Hibernate for data and Java files for expressing the logic. As a result, we get the web application where different roles corresponding to the non-executable service systems can access different services via jsp pages. The navigation between the pages for the template given in Figure 51 is shown in Figure 52. The first page is always the main page, in which the users corresponding to different non-executable service systems can login. Then, there are pages for each user containing the list of transactions with possibility to choose which one to modify. The table containing all service transactions is used to capture simultaneous executions of the service. Finally, from these pages, there is a navigation link to the service page in which the user can enter the fields in GUI and execute the service.

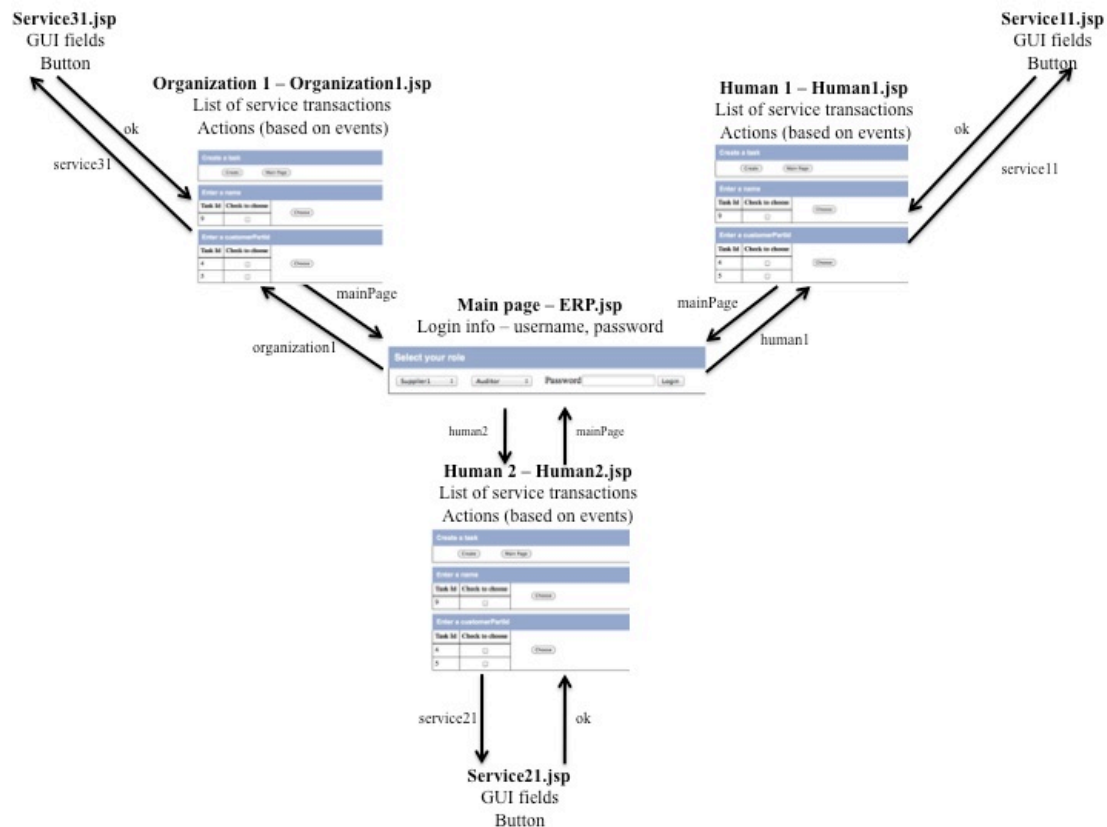


Figure 52: Output: Page navigation of prototype

Prototyped service specification and the result of prototyping for Order Creation at Générale Resorts are shown in Figure 53. The output contains only the role page, in this case OrderEntryPerson.jsp. For each of the services as a whole in OrderEntryPerson, it contains the table of OrderCreation transactions containing transactions that satisfy the precondition for that service. In addition to that, it contains the default buttons, for creating the new OrderCreation transaction and for going back to the main page.

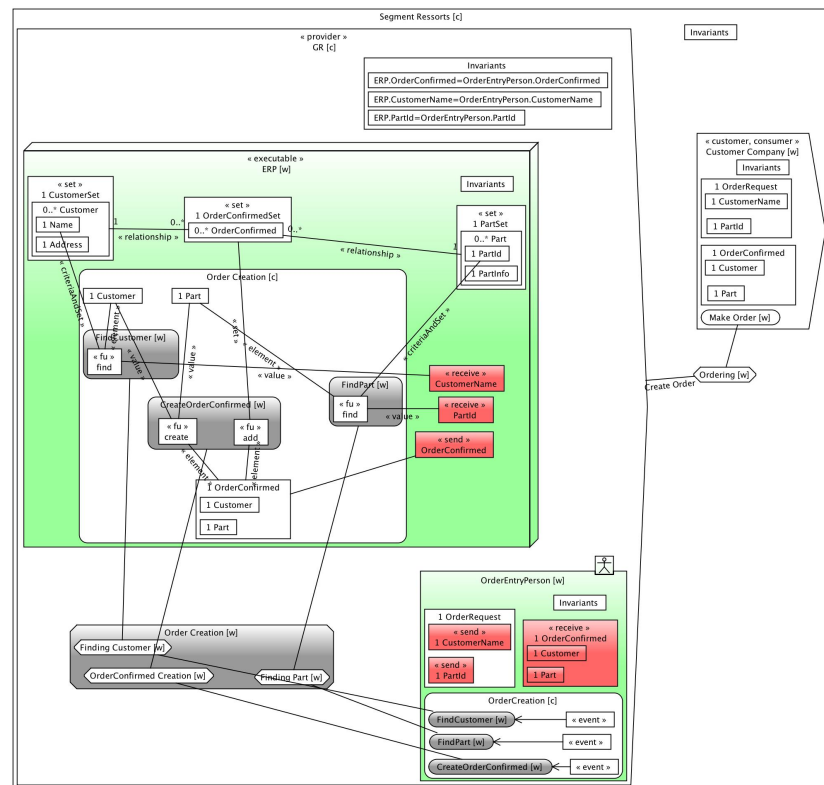


Figure 53: Input of prototyping for Order Creation at GR

OrderEntryPerson Role

OrderCreation

Create

Main Page

FindCustomer

OrderCreation Id	Customer	Part	OrderConfirmed	Check to choose
# {element.id}	# {element.customer}	# {element.part}	# {element.orderConfirmed}	<input type="checkbox"/> <div>Choose</div>

FindPart

OrderCreation Id	Customer	Part	OrderConfirmed	Check to choose
# {element.id}	# {element.customer}	# {element.part}	# {element.orderConfirmed}	<input type="checkbox"/> <div>Choose</div>

CreateOrderConfirmed

OrderCreation Id	Customer	Part	OrderConfirmed	Check to choose
# {element.id}	# {element.customer}	# {element.part}	# {element.orderConfirmed}	<input type="checkbox"/> <div>Choose</div>

Figure 54: Output of prototyping for Order Creation at GR

The mapping between model elements and application elements is shown in Table XI. Events are mapped to buttons of the service pages. Service as a composite *Create Order ERP* with intermediate data of executable system *ERP* is mapped to the table *OrderCreation* with columns *Customer*, *Part* and *OrderConfirmed*. Receive properties of executable system correspond to the GUI fields, as they are the properties coming from external systems, i.e. are entered through the interface. Set and non-set properties of executable system are stored as database tables. Service as a whole is mapped to the logic of the service on the corresponding button click.

Table XI: Mapping input and output of prototyping for Order Creation at GR

Input (IT specification)	Output (Java prototype)
Event (Create Order Creation, Find Customer, Find Part, Create Order Confirmed)	Button
Functional Unit	Logic of service on corresponding button clicks
Service as a whole from executable system (Find Customer, Find Part, Create OrderConfirmed)	Form
Service as a composite from executable system (Create Order)	Tab with a list of transaction tables for each service it contains as a whole. List of transactions for one service as a whole contains all transactions for which the precondition of that service holds.
Receive properties of executable system (Name, PartId)	GUI fields
Intermediate data in service as a composite (Customer, Part, OrderConfirmed)	Table <i>OrderCreation</i> for capturing all transactions of Order Creation with ID, customer, part and orderConfirmed field
Properties of executable system	Database tables

Once we have explained the input and output of prototyping, in the following chapters we explain how the transformation process is done, i.e. the step 1 transforming the model to the Arcimboldo project.

8.2 Mapping Prototypical Specification to Arcimboldo Project

We explain the overall structure of generated Arcimboldo project for the prototypical specification template explained in previous chapter and explain the mapping between prototypical specification elements and Arcimboldo files.

The main files required for Arcimboldo project are: the main object, descriptor file and templates. The main object contains JSON object with data, which when

combined with the template produces the given page, class or another part of application. It appears in the project with extensions .jao.

The templates correspond to the JSON templates, with some particularities. They can be used to write templates for pages, classes or other parts of application. They appear in the project with extension .tpl.

The descriptor file contains the list of all the files that must be created, and for each file the information which template and what part of the main object it is created from. It appears in the project with extension .fls.

Arcimboldo project that we generate from BASS service model contains one main object (.jao file), one descriptor file (.fls file) and many templates for describing pages, classes, logic of application, and others. In addition, it can contain files that just need to be copied and can be created without templates. There are several files to be generated:

- The main object (GR.jao) – containing data that when combined with descriptor file and templates generate pages, classes and other application elements.
- Descriptor file (GR.fls) – list of files to be generated by Arcimboldo explaining how the templates are combined and expanded to the pages, classes and other application elements.
- Other files – platform-related files that need to be copied (log4j.xml, Manager.java, folder libs) – files needed for the given platform and environment in which the application is generated. In this case, it is Java platform, enterprise edition (JEE) with MySql database and Hibernate and JSF libraries. Therefore, we need files such as log4j.xml, folder libs containing all libraries necessary for the given project and Manager.java for EntityManager used for entity beans.
- Templates for generating graphical user interface (GUI)
 - ERP.tpl – template for generating home page where different roles can login.
 - Role.tpl – template for generating pages for different roles (all service systems except IT system) with the behavior and data related to them.
 - Navigation.tpl – template containing navigation between the pages.
 - Web.tpl – template for generating web.xml containing servlet mappings for JSF.
- Templates for generating data-related elements
 - JPA.tpl – template for generating entity beans that are then mapped to database tables.
 - Persistence.tpl – template for generating persistence file needed for Hibernate to map entity beans to database tables.
 - ApplicationBean.tpl – template for generating application bean containing the global data for the whole application.

- Templates for generating logic of application
 - RoleBean.tpl – template for generating logic for different roles, such as login and logout methods.
 - ActionBean.tpl – template for generating session beans containing the logic and data related to services.
 - Logic.tpl – template for generating global logic for the application, such as loading database, etc.

In this chapter we explain the main object and descriptor file. In next chapters, we explain the other files necessary for generation of GUI, data and logic. They include templates and the parts of the main object that need to be generated. Therefore, we show the details of the main object parts relevant for generating interface, data and logic in further chapters. Figure 55 shows the descriptor file (left) and the main object (right).

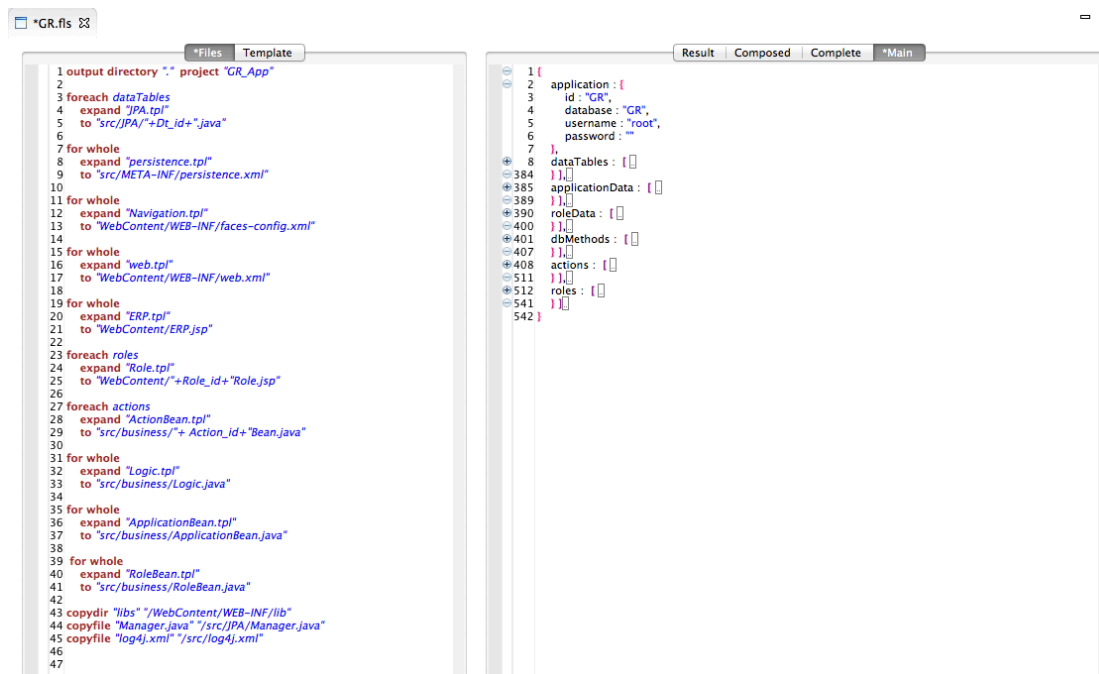


Figure 55: Descriptor file and the main object

The main object contains of several parts:

- Application – containing main data about application, such as database schema name, username and password and name of the application that will be generated by Arcimboldo.
- DataTables – contains all information about database tables, their fields, types and relations. This corresponds to the properties defined in the context of executable system that need to be stored permanently.
- ApplicationData – contains all global data, defined in the scope of whole application and stored in the application bean. This corresponds to the properties defined in the context of executable system's environment.

- RoleData – contains data necessary for roles, such as their login name and password.
- Actions – contains list of services inside modeled service system with their fields, preconditions and effect of the action expressed with functional units.
- Roles – defines list of roles for the application with the list of services they can view and list of services they can perform. This corresponds to the services of non-executable service systems. If the service is in the given role with event than it is in the list of services that can be performed. If the service is in the given role without the event, then it is in the list of service that can be viewed only.

The main object is organized based on the main modeling concepts that are used to generate different parts of application: related to GUI, data and logic. Table XII shows the parts of the main object, the corresponding modeling concepts and the part of application to which it is related (GUI, data, logic). In addition, we give the explanation of the parts of the main object that do not come from the model and model elements that are not visible in the main object.

As we can see, non-executable systems are mapped to the roles in the main object with the corresponding attributes. Executable system contains different concepts that are mapped to the parts of the main object. Services as a whole are mapped to the actions, properties to the dataTables, environment properties to the applicationData. Services as a composite are not represented in the main object, but are used for mapping to the corresponding database table for simulating state machine together with the properties inside the service. This will be explained later in Chapter 8.5.

Table XII: The main object elements and corresponding modeling concepts

Modeling Method Concepts	Main Object Elements	Application Part
Non-executable systems <ul style="list-style-type: none"> • service system name • services without event (in which the service system is participating) • services with event (that performs the service system) 	Roles <ul style="list-style-type: none"> • role name • roleViewList • roleActionList 	GUI
Services as a whole in executable system <ul style="list-style-type: none"> • service system name • receive properties for service • functional units 	Actions <ul style="list-style-type: none"> • action name • gui • units 	Logic
Properties in executable system	dataTables	Data
Properties in executable system's environment	applicaiionData	Data

In addition, *application* and *roleData* are used in the main object for showing execution-related data connected with the database schema and login for different roles. Functional units are mapped to the library of predefined functional units used to express the semantics of the service.

In further chapters, we show how each part of the main object looks like. For GUI description, we show how *roles* and *roleData* look like. For transforming static elements (data), we show *application*, *dataTables* and *applicationData*. Finally, for transforming dynamic elements (logic), we show *actions*.

On the left side of Figure 55 we see description file. It shows how the templates are expanded using elements from the main object to generate the application. First line shows the directory in which the application will be generated. Lines 3-5 describe generation of entity beans. For each database table described in the main object in *dataTables*, the template *JPA.tpl* is expanded to the file *Dt_id.java*. *Dt_id* is the element from the main object, corresponding to the name of the database table. Lines 7-9 describe generation of the file *persistence.xml*. This file is generated only once for the whole application. Therefore, we use expression *for whole* with the expression *expand* of the template *persistence.tpl*. Similarly, we describe expansion of all the other templates. For example, for each *action* described in the main object, we expand the *ActionBean.tpl* to the corresponding java class. For each role described in the main object, we expand *Role.tpl* to the corresponding jsp page. At the end, we copy the necessary files to the generated applications, such as folder *libs* containing necessary libraries and files *log4j.xml* and *Manager.java*. In the expand expression we give the name of the generated file, as well as the path in the generated project in which it will be stored. The structure of the generated application corresponds to the typical structure of the JEE application.

In this chapter we have explained the overall structure of the intermediate Arcimboldo project generated from the prototypical specification, as well as the structure of the main object and descriptor. In the following chapters we explain the other files for generating interface, data and logic. For each of them, we need to generate corresponding templates and parts of the main object. For each of these sections, we give pseudo code for transforming prototypical specification to intermediate Arcimboldo files. Then, we describe the related parts of the main object, related templates and how from these files Arcimboldo generates necessary classes and pages.

8.3 Generating GUI

In this chapter, we explain how GUI is generated from prototypical service specification. We need to generate *roles* and *roleData* part of main object and templates *ERP.tpl*, *Role.tpl*, *Navigation.tpl* and *web.tpl*. First we give pseudo code for

generating the parts of the main object and templates and then give examples of how the resulting main object's parts and templates look like.

Prototypical Service Specification to Arcimboldo files: The main object

To generate roles and roleData in the main object from prototypical specification, we do the following:

for each non-executable service system

- create an object role in roles in the main object containing role name and list of services that can be viewed only (without event) and that can be performed (with event)

- create an object roleData in roleData containing username and password for login for that role

We map non-executable service systems to the roles and services inside them to the corresponding roleViewList and roleActionList.

Prototypical Service Specification to Arcimboldo files: Templates

There are several templates we need to generate for GUI of the application. The pseudo code for this generation is:

for whole

- create template Role.tpl for generating pages for each role containing all the services it can perform and view

- create template ERP.tpl for generating home page with a list of all roles that can then login with their credentials

- create template Navigation.tpl for generating navigation between pages that is by default set between the home page and the role's pages

- create template web.tpl for generating web.xml file containing default information about servlet mapping needed for JSF

We generate the files using necessary languages and frameworks, such as JSF for generating pages, XML files for generating faces-config.xml containing navigation between the pages and web.xml for defining servlet mappings. The templates have the default format independent of the model that is being prototyped. Their format depends only on the used technologies. We use the default navigation between the pages, containing the home page (generated from ERP.tpl) and one page for each role. On each of the role pages there are tables that role can view and the list of actions that are activated when the certain precondition holds. Therefore, the navigation as well is generated by default.

Arcimboldo files to Java application

An example of the generated element for the roles is given in Figure 56. Role contains information about the name of the role coming from the name of the

corresponding non-executable service system. It also contains the list of actions roleActionList corresponding to the services of that service system. In addition, it contains the roleViewList containing the data the role is viewing, corresponding to the properties of that service system.

```

role_id : "orderEntryPerson",
roleActionList : [
{
  Saction : "actions['findCustomer']"
},
{
  Saction : "actions['findPart']"
},
{
  Saction : "actions['createOrderConfirmed']"
} ],
roleViewList : [ {} ]

```

Figure 56: The main object: roles

An example of the template is given in Figure 57. It shows one part of the Role.tpl that corresponds to the JSF page for one role (non-executable system). Using JSF tags we represent the table containing the data of the corresponding database tables of the roleViewList. This template is expanded based on the elements from the main object (roles and roleViewList in this case) to the files such as OrderEntryPerson.jsp. This is done by Arcimboldo as explained in Chapter 3.4. The resulting page is shown on the right of the Figure 57.

```

{.repeated section roleViewList}
...{.repeated section view}
...<h:form>
...<h:dataTable value="#{.meta-left}logic.{dt_id}s{.meta-right}"
...var="element" cellpadding="6" cellspacing="0"
...headerClass="table-header"
...rowClasses="table-odd-row,table-even-row">
...<h:column>
...<f:facet name="header">
...<h:outputText value="{Dt_id} Id"></h:outputText>
...</f:facet>
...<h:outputText value="#{.meta-left}element.id{.meta-right}"></h:outputText>
...</h:column>
...{.repeated section attributes}
...<h:column>
...<f:facet name="header">
...<h:outputText value="{AttName_id}"></h:outputText>
...</f:facet>
...<h:outputText value="#{.meta-left}element.{attName_id}{.meta-right}"></h:outputText>
...</h:column>
...{.repeat end}
...{.repeated section relationships}
...{.section ManyToOne}
...<h:column>
...<f:facet name="header">
...<h:outputText value="{To_id} Id"></h:outputText>
...</f:facet>
...<h:outputText value="#{.meta-left}element.{to_id}.id{.meta-right}"></h:outputText>
...</h:column>
...{.end}
...{.repeat end}
...</h:dataTable>
...</h:form>
...{.repeat end}
{.repeat end}

```

Figure 57: Role.tpl for generated page for OrderEntryPerson shown in Figure 54

8.4 Transforming Static Elements (Data)

In this chapter, we show how the data-related elements of the application are generated. We need to generate dataTables and applicationData part of the main object and then generate templates JPA.tpl, persistence.tpl and ApplicationBean.tpl. First we give pseudo code for generating the parts of the main object and templates

and then give examples of how the resulting main objects' parts and templates look like.

Prototypical Service Specification to Arcimboldo files: The main object

To generate dataTables and applicationData of the main object we do the following:

for all properties in executable system

 create an object in dataTables in the main object

for all properties in executable system's environment

 create an object in applicationData in the main object

We map the properties to the parts of the main object: properties of executable system to the dataTables and properties of executable system's environment to the applicationData.

Prototypical Service Specification to Arcimboldo files: Templates

There are several templates to be generated, as described below.

for whole

create template JPA.tpl for generating entity beans for each dataTable element in the main object

create template persistence.tpl for generating persistence.xml file needed by Hibernate for mapping entity beans to database tables

create template ApplicationBean.tpl for generating application bean containing global data relevant for the whole application

As explained for the other templates, these templates are generated independent of the model transformed. They depend on the used technologies only.

Note that in addition to the templates and changes in the main object, file Manager.java is copied that is used as EntityManager for entity beans.

Arcimboldo files to Java application

An example of the dataTables element is given in Figure 58. Element dataTable contains the name of the database table corresponding to the name of the set property and attributes corresponding to the attributes of the property in the model with their name and type. Attributes referring to the other properties are shown in the separate section relationship.

```

id: "Customer"
attributes: [
  {
    stringAttribute,
    varName: "Name"
  },
  {
    stringAttribute,
    varName: "Address"
  }
],
relationship: [
  {
    OneToMany,
    to: "OrderConfirmed"
  }
]

```

Figure 58: The main object: dataTables

An example of the template is shown in Figure 59. We show the template JPA.tpl representing the template for generating the entity beans. The template is expanded into the corresponding java file, such as Customer.java shown on the right side of the figure. The template is expanded by Arcimboldo based on the elements of the main object as explained in Chapter 3.4.

```

@Entity
public class {Id} {meta-left}
    @Id
    @GeneratedValue(strategy = GenerationType. IDENTITY)
    private Long {id}_id = null;
    {section attributes}{repeated section @}
        private {type} {id1};
    {end}{end}
    {section OneToMany} @OneToMany (mappedBy="{id}")
    private Collection<{Id1}> {id1}s = new ArrayList<{Id1}>();
    public Collection<{Id1}> get{Id1}s() {meta-left}
        return {id1}s;
    {meta-right}
    public void set{Id1}s(Collection<{Id1}> {id1}s) {meta-left}
        this.{id1}s = {id1}s;
    {meta-right}
    {end}
}

@Entity.¶
public class Customer {¶
    ...@Id.¶
    ...@GeneratedValue(strategy = GenerationType. IDENTITY).¶
    ...private Long customer_id = null;¶
    ...public String name;¶
    ...public String address;...¶
    ...private transient boolean check = false;¶
    ...¶
    ...@OneToMany (mappedBy="customer").¶
    ...private Collection<OrderConfirmed> orderConfirmed = new ArrayList<OrderConfirmed>();
    ...public Collection<OrderConfirmed> getOrderConfirmed() {¶
    ...return orderConfirmed;¶
    ...}¶
    ...public void setOrderConfirmed(Collection<OrderConfirmed> orderConfirmed) {¶
    ...this.orderConfirmed = orderConfirmed;¶
    ...}...¶

```

Figure 59: Expanding template JPA.tpl for generating entity beans (and database tables)

8.5 Transforming Dynamic Elements (Logic)

In this chapter, we show how the logic-related elements of the application are generated. We need to generate action parts of the main object and then to generate

templates RoleBean.tpl and ActionBean.tpl. First we give pseudo code for generating the parts of the main object and templates and then give examples of how the resulting main objects' parts and templates look like.

Prototypical Service Specification to Arcimboldo files: The main object

To generate actions in the main object, we need to do the following:
 for all services as whole in executable system
 create an object in actions in the main object containing service name
 name of the role in charge for it (where the event in the model is)
 for each value coming from external system GUI field with its name
 and type
 for each functional unit with its type and parameters

We map the functional units and services with their related receive properties to the parts of the main object.

Prototypical Service Specification to Arcimboldo files: Templates

There are three templates that need to be generated:

for whole
 create template RoleBean.tpl
 create template ActionBean.tpl
 create template Logic.tpl
 RoleBean.tpl contains the logic related to the roles, such as login and logout.
 ActionBean.tpl contains the data and logic related to the services of the roles.
 Logic.tpl contains the logic of the application used on the global level, such as loading database, and others.

Arcimboldo files to Java application

An example of the actions element of the main object is shown in Figure 58. Elements' actions contain the name of the corresponding service, GUI parameters corresponding to the suitable receive properties values, units corresponding to the functional units describing the logic of the service and precondition describing the condition that must hold for the service to be performed.

```

action_id : "FindCustomer",
units : [
{
  create : { },
  Stable : "dataTables['customer']",
  value : [
  {
    gui : { },
    string : { },
    name_id : "name",
    SdbField : "dataTables['customer'].attributes['name']"
  }
  ]
}
]

```

Figure 60: The main object: actions

An example of the template is shown in Figure 61. It shows one part of the template Logic.tpl showing how general logic is used in the application. The right side of the figure shows the file that is the result of the expansion of the template containing the necessary data and methods.

```

    { .repeated section.dataTables }
    public {Dt_id}.get{Dt_id}ById(long id){
        ...em = Manager.open();
        ...return em.find({Dt_id}.class, id);
    }
    { .repeat end }
    public void loadDB(){
        em = Manager.open();
        Query result;
        result = em.createQuery("SELECT t FROM MyTransaction t");
        myTransactions = (Collection<MyTransaction>).result.getResultList();
        { .repeated section.dataTables }
        result = em.createQuery("SELECT t FROM {Dt_id} t");
        {dt_id}s = (Collection<{Dt_id}>).result.getResultList();
        { .repeat end }
    }
}

public void loadDB(){
    em = Manager.open();
    Query result;
    result = em.createQuery("SELECT t FROM MyTransaction t");
    myTransactions = (Collection<MyTransaction>).result.getResultList();
    result = em.createQuery("SELECT t FROM Customer t");
    customers = (Collection<Customer>).result.getResultList();
    result = em.createQuery("SELECT t FROM Part t");
    parts = (Collection<Part>).result.getResultList();
    result = em.createQuery("SELECT t FROM OrderConfirmed t");
    orderConfirmed = (Collection<OrderConfirmed>).result.getResultList();
}

```

Figure 61: Expanding Logic.tpl to Logic.java

Synchronisous services: state machine simulation

We illustrate how we simulate service multiplicity. Figure 60 shows the specification of *Order Creation* at GR with preconditions. Order Creation corresponds to the long running transaction that include several atomic transactions for finding the customer that created the order in the system, finding the part the customer ordered and creating order confirmed with these two parameters. Therefore, we model the service *Order Creation* with its sub-services *Find Customer*, *Find Part*, *Create Order Confirmed*. Data that are created during the execution of this transaction and are therefore shown as intermediate data in the context of *Order Creation* are: *Customer*, *Part* and *OrderConfirmed*. Every service has its pre-condition, i.e. the condition that should hold for the service to have the certain effect on the system expressed with functional units. In this example, all service preconditions contain the expression that the output element is null (for example for find customer, that customer=null). This is the way to differentiate between the services that can be performed multiple times and only once. When we set this precondition, it means that the service can be performed only once, and then the customer value for example will

be set to the value different than null. Therefore, based on precondition it is not possible to perform that service once more. *Create Order Confirmed* precondition contains in addition the condition that *customer!=null* and *part!=null*. This corresponds to the fact that customer and part as input values related to the intermediate data customer and part of *Order Creation* service have to be initiated before the creation of order is confirmed. If we would show the inputs and functional units, we would see the relation between *Create Order Confirmed* functional units and customer and part intermediate data.

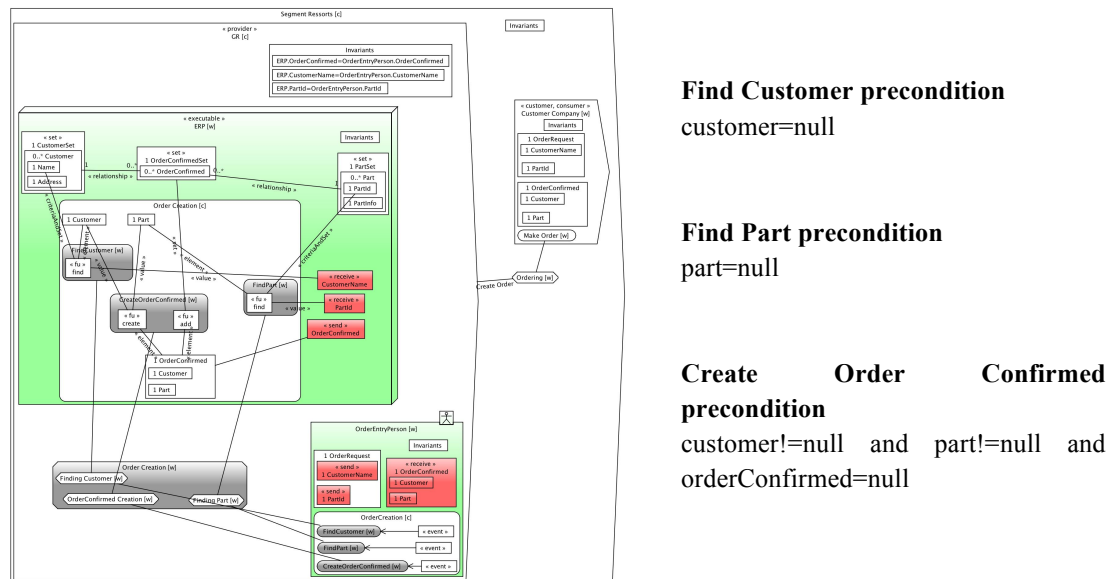


Figure 62: Service specification at Générale Ressorts with preconditions

The diagram shows one instance of the *Order Creation*. To execute the model, we need to include multiple transactions of *Order Creation*. In order to achieve that, we use one persistent table to store all transactions of *Order Creation*, which is used to simulate the state machine. Transition from one state to the other is possible depending on the preconditions of the services, i.e. the conditions set on top of data. Data outside *Order Creation* live already in their own context, either as GUI data, or data in the IT system. Therefore, the table contains intermediate data related to the sub-services: *Customer*, *Part* and *OrderConfirmed* as shown in Table XIII. The table also contains *Order Creation ID* for each transaction of *Order Creation*. This field is created automatically for each long-running transaction as id of one transaction. Based on preconditions that combine constraints on data, each of services can or cannot be performed. This way, state transitions are modeled based on *Order Creation* table and the conditions relating data.

Table XIII: Table for capturing Order Creation transactions

Order Creation ID	Customer	Part	OrderConfirmed
1

The state machine for the specification of service *Order Creation* in Figure 42 is shown in Figure 63.

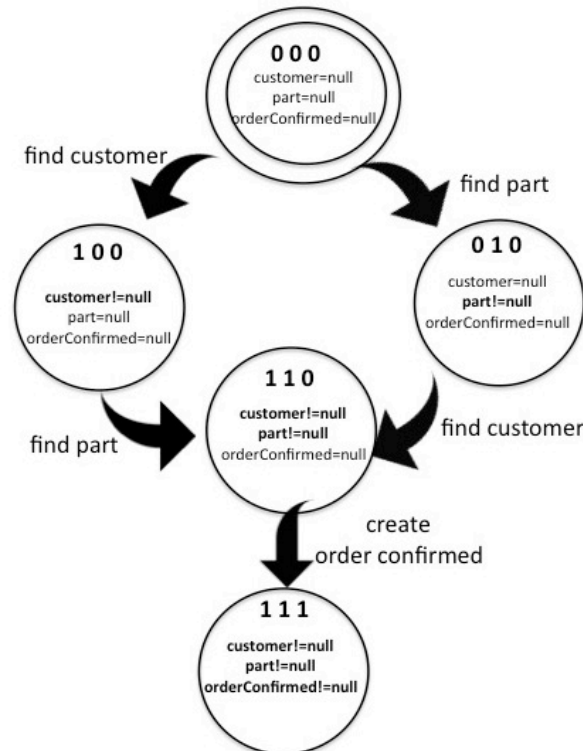


Figure 63: State machine for Order Creation business transaction

Each state is marked with 3 numbers 0 or 1, corresponding to the availability of customer, part and orderConfirmed respectively. The initial state is 000, i.e. when customer, part and orderConfirmed are null. From that state, it is possible to do find customer or find part, which leads to the states 100 or 010 respectively. In these states we can again do find customer or find part, leading to the corresponding states and finally leading to 110, meaning that both the customer and part are available. Create order confirmed can then be performed, leading to the state 111, meaning that the one transaction of *Order Creation* is finished.

This chapter has illustrated how we prototype service specification. The prototypes are then used to validate whether the service specification corresponds to business needs.

9 The Practical Impact: Evaluation of the Developed Theory in Practice

In this chapter, we show the evaluation of the proposed method. The evaluation method we use belongs to the group of observational design evaluation methods. We show how the method is applied on a case study of research project management we conducted at EPFL. In addition to that, we have conducted a survey with a group of practitioners to get their feedback on how much BASS is useful in practice.

In chapter 9.1 we describe the case of research project management based on the consulting project we have conducted at EPFL and show the benefits of the use of BASS for this case study. For this project, it was not relevant to create simulate and prototype the model. Therefore, we illustrate the service specification spiral only (Chapter 5.1).

In chapter 9.2 we position our method among other modeling methods and explain similarities, differences, and what our method brings as new. It is based on the related work described in Chapter 2.

Chapter 9.3 includes the feedback we have received from practitioners about the usefulness of the method in practice. The results of the feedback are used to describe the contributions, practical recommendations and limitations of the method (Chapter 10).

9.1 Case Study: The Case of Research Project Management at EPFL

In order to propagate research, EPFL is doing many research projects. Once the professors and researchers of EPFL define a project, they need to find funding for it and apply for it. The following actors take part in the application process for research funding:

- **Funding Organization (FO)** is an external source of funding. It can be a foundation, government department, corporation, company, private donor, etc. The reasons for funding vary from the desire for advancement in knowledge to obtaining profit. In both cases, the organization maintains close contact with its partners, because its image is very important. As different organizations have different regulations for granting funds, researchers have to adapt administration of their project (non-scientific part) to comply with the requirements of the FO. The FO makes research project acceptance decisions based on an internal competitiveness.
- **Researcher (Tom)** is a person at EPFL who is passionate about progress and wants to respond to society's concerns. With his laboratory, or together with partner laboratories, he has the competencies and ideas to carry out a research project. He applies for funds for his research project, but he does not want to

spend much time on administrative procedures for preparing the funding application. The funds he might get have certain policies and regulations to which the research project must comply.

- **EPFL Support Organization (SO)** is an organization within EPFL that promotes the quality and image of research conducted. For this purpose, the SO provides researchers with all fund-related information, assists them with finances and offers them administrative support for their application process for research funding. SO also manages the intellectual property resulting from research through evaluating new inventions, negotiating and approving research contracts with industrial partners.
- **Administration fédérale des finances (AFF)** is an external organization that audits University X to check if the financing of the projects is compliant with the financial standards, like International Public Sector Accounting Standards (IPSAS).
- **Research community** is consisted of everyone that benefits and is interested in the results of the funded project. Often, the results of the research project are published in conferences or journals, and these publications are available to the research community.

To model this case study, we start with conceptualizing the main process Propagate Research and the main service systems collaborating to propagate research using the template from Figure 15. Then we refine the implementation of the service in service provider, in two cycles of the service specification spiral explained in Chapter 5 (Figure 29), using the steps 1-4. In the first cycle we model implementation of the service Propagate research inside EPFL with its partners as the service provider. In the second cycle, we refine further the support organization of EPFL+ to show how it implements the supporting services. We explain all the steps of all cycles and the resulting diagrams. Finally, we explain the benefits of using BASS in this project. By convention, information in italics corresponds to the names in the diagram.

9.1.1 Initial Model Design

In this phase we identify the core process we model: *Propagate research [w]* with the main external service of the service provider and services of other stakeholders it uses: *Do research project[w]*, *Fund research[w]*, *Access research publications[w]* and *Check compliance with accounting standards[w]* of the corresponding service systems in which the service is creating value: *EPFL+[w]*, *SNF[w]*, *Research community[w]*, *AFF[w]*. We also define the service properties. Based on identified elements, we get the resulting diagram in Figure 64.

Figure 64 shows the result of initial model design. Service systems *EPFL+[w]*, *SNF [w]*, *Research community [w]*, *AFF [w]* are shown in the context

Research segment[c]. They participate in the process *Propagate research [w]* with services *Do research projects [w]* of *EPFL+[w]*, *Fund research [w]* of *SNF [w]*, *Access research publications [w]* of *Research community [w]* and *Check compliance with accounting standards [w]* of *AFF [w]*. Responsibilities of each of the stakeholders in implementing the service *Propagate research [w]* are shown with their services.

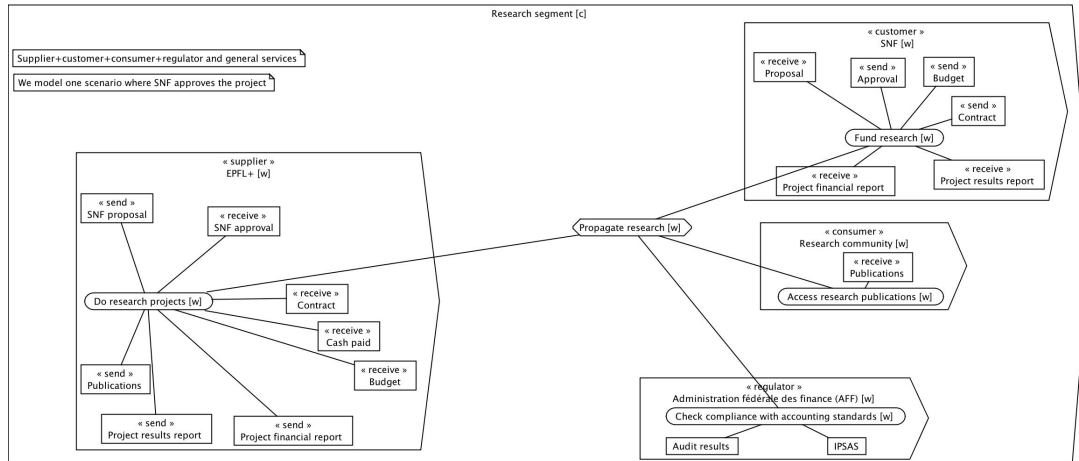


Figure 64: Result of conceptualization

We can continue with the first cycle, where we analyze how the service provider *EPFL+[w]* is organized to implement service *Do research projects [w]*.

9.1.2 Cycle 1: EPFL+

In previous iteration, we have identified the main external service *Do research projects [w]* that service provider EPFL+ needs to provide. In this cycle, we refine *EPFL+[w]* to analyze the details of implementation of the service *Do research projects [w]*. Following the step 1-4 of the service specification spiral, we analyze the specification of service *Do research projects [w]*.

Step 1: External services

In this step, we add detailed external services to accommodate expected interactions with customers and consumers. Once they are identified, properties are distributed to them. The decisions are captured in Figure 65 and Figure 66.

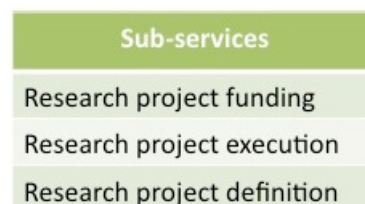


Figure 65: EPFL+: Step 1 Conceptualize

	Research project funding	Research project execution
Contract	X	
SNF Proposal	X	
SNF Approval	X	
Budget	X	
Project financial report		X
Project results report		X
Publications		X

Figure 66: EPFL+: Step 1 Decide

The result of this step is shown in Figure 67. As we can see the difference is visible in *EPFL+[w]*. Service *Do research projects [w]* is refined into *Do research projects [c]*, so that new sub-services: *Research project definition [w]*, *Research project funding [w]* and *Research project execution [w]* are shown. Properties of *EPFL+[w]* are now related to the new sub-services, based on the fact which service is using what resources, as captured in Figure 66.

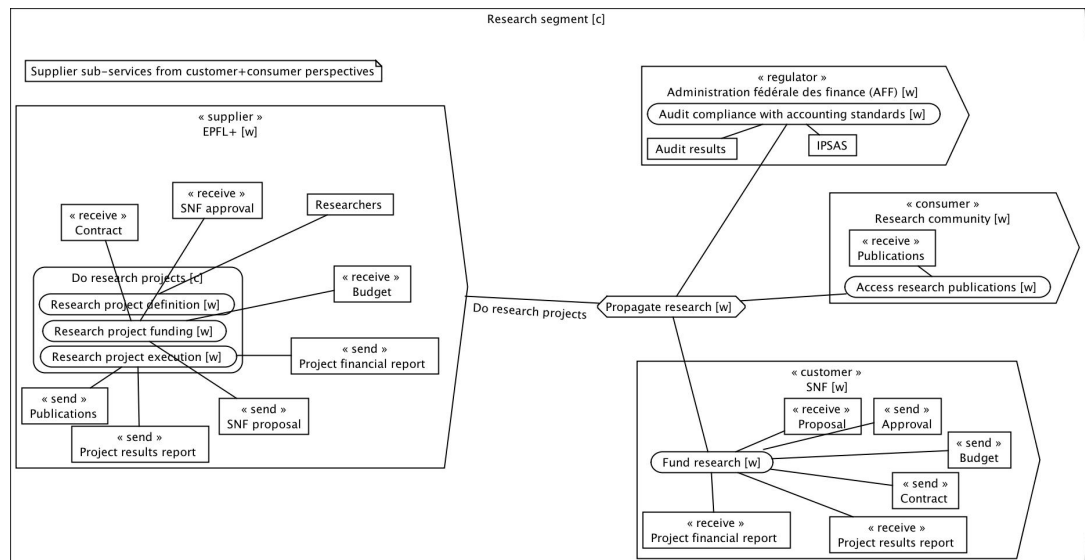


Figure 67: EPFL+: Result of step 1

Step 2: Internal structure

So far, we have identified what external services *EPFL+[w]* needs to provide in order to ensure customer's satisfaction. Now, we can decide the details necessary for implementation of this service inside *EPFL+[w]*. First, we identify service systems participating in its implementation. As mentioned, we group people, technologies and other resources based on their participation in the implementation of service *Do research projects*, i.e. based on the implementation of its sub-services. Therefore, this structure does not need to reflect the typical hierarchical organizations of companies. Instead, it reflects the organizations needed to implement the given

service. Then we allocate all the properties that were in $EPFL+[w]$ to the newly identified service systems. These decisions are captured in matrices in Figure 68 and Figure 69.

Roles
UNIL
Faculty
Laboratory
Support services
EPFL management

Figure 68: EPFL+: Step 2 Conceptualize

	UNIL	Faculty	Laboratory	Support services	EPFL management
Contract	X		X	X	
SNF Proposal			X	X	
SNF Approval			X	X	
Budget	X	X	X	X	X
Project financial report			X	X	
Project results report			X	X	
Publications	X		X		

Figure 69: EPFL+: Step 2 Decide

The result of this step is shown in Figure 70. As we can see, all the changes are inside EPFL+. It is now refined into $EPFL+[c]$, revealing its internal structure with: *UNIL [w]*, *Faculty [w]*, *EPFL management [w]*, *Support organization [w]* and *Laboratory [w]*. As we can see, it does not contain the typical departments we would see inside the company or organization. Instead it contains people and technologies grouped into organizations. Therefore, for example, the partner university UNIL is inside this service system as well. Also, as we will see, support organization can include some people hired in the laboratories or faculties. The resources are allocated to the newly identified service systems based on who is responsible and who uses and knows what resources.

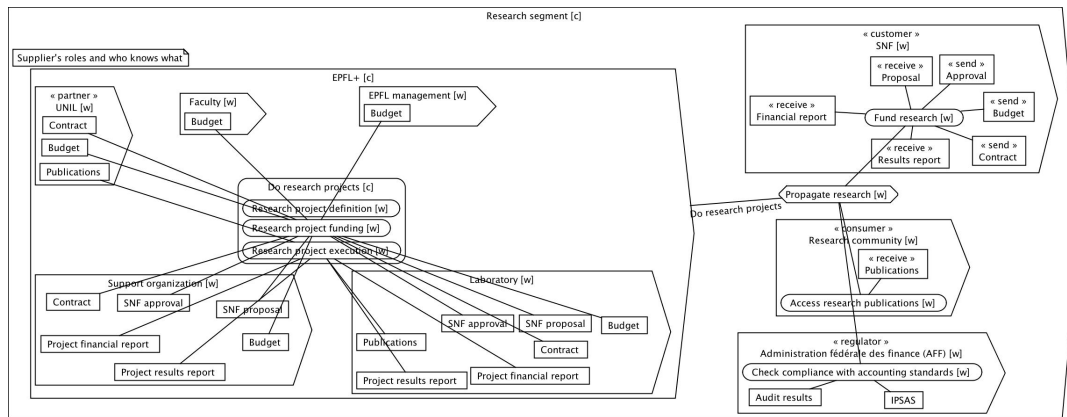


Figure 70: EPFL+: Result of step 2

Step 3: Internal services

So far, we have identified what external services should be provided by EPFL+, with what properties, how is EPFL+ organized to provide them and what properties each of internal organizations is using. Now, we should decide what internal services EPFL+ needs to provide to implement the needed external services. In this case, the services remain the same, i.e. there are no sub-services of the external visible services. Therefore, the diagram remains the same as it was. In case there would be new internal services, as the sub-services of external services, we would need to allocate the properties they use to them as well.

Step 4: Internal responsibilities

Final step of analyzing the details of implementation is to define the responsibilities of each of service systems inside *EPFL+[c]*. For each of sub-service of *Do research projects[c]*, we identify the process that implements it with the services inside the service systems this process is using. The decision is captured in Figure 71 and Figure 72. Next to service's name is one of the letters representing the role of that service system in the implementation of the service, corresponding to the RACI matrix. For each service, we also allocate service properties to the services that its implementing process is using.

Processes	
Research project definition	UNIL Define project (R), Lab Define project (A), Support services Support project definition (R)
Research project funding	UNIL Get funding (R), Faculty Manage faculty funding (I), EPFL Management Manage EPFL funding (I), Support services Support project funding (R), Laboratory Get funding (A)
Research project execution	UNIL Execute project (R), Laboratory Execute project (A), Support services Support project execution (R)

Figure 71: EPFL+: Step 4 Conceptualize

	UNIL Execute project	Lab Execute project	...
Publications	X	X	
....			

Figure 72: EPFL+: Step 4 Decide

The result of this step is shown in Figure 73. We can see the organization *EPFL+[c]* with its internal service systems and the services for which they are responsible, as well as the resources these services are using and the properties they should satisfy. These services are then used by the process that implements the higher-level services, such as: *Research project definition*, *Research project funding* and *Research project execution*.

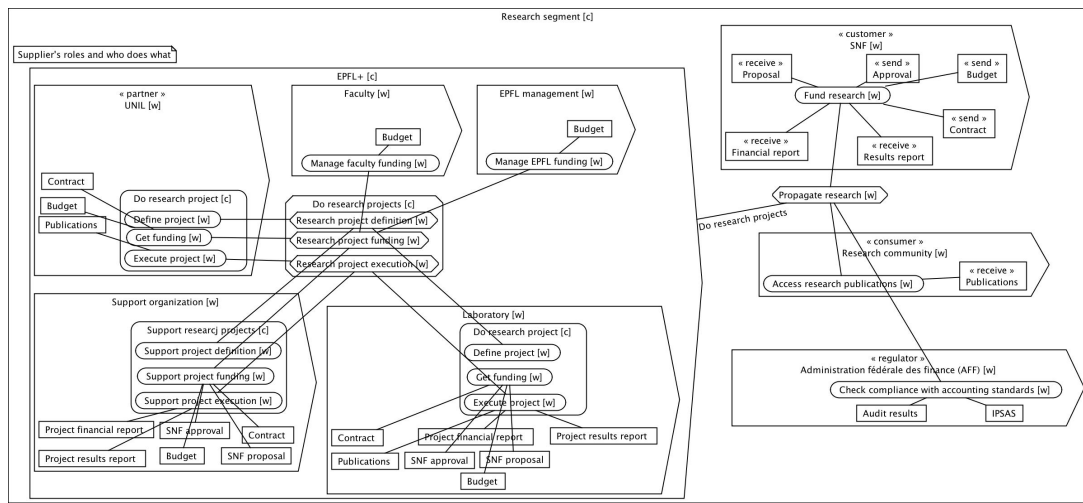


Figure 73: EPFL+: Result of step 4

Now, we have all the details of service implementation inside EPFL+. For the purpose of this project, it was important to analyze service implementation and responsibilities inside the support organization as well. Therefore, we do one more cycle of the design process.

9.1.3 Cycle 2: Support organization

Step 1: External services

The first step would be to analyze what the external services are that *Support organization [w]* should provide. In this case, they are the same services as already identified in support organization, so there is no need for this step, and the resulting diagram remains the same as it was.

Step 2: Internal structure

In this step, we analyze the details of implementation in *Support organization [w]*. We first decide the internal structure of support organization. This decision is captured in Figure 74. Based on the decision, *Support organization[c]* contains service systems: human *Lab secretary [w]*, software component *GrantsDB [w]*,

software component *SAP [w]*, organization *HR [w]* and organization *Financial services [w]*. As we can notice, laboratory secretary belongs to the support organization, even though it is hired by the laboratory that is different service system. This reflects the service-based organization, i.e. the fact that people are structured into organizations based on their role in service implementation, not the organization that is hiring them. We also allocate the properties of *Support organization [w]* to the newly identified service systems. All the decisions are captured in Figure 74 and Figure 75.

Roles	Type
HR	Business Organization
GrantsDB	Software component
Lab secretary	Human
Financial services	Business organization
SAP	Software component

Figure 74: Support organization: Step 2 Conceptualize

	HR	Grants DB	Lab secretary	Financial services	SAP
Contract	X		X	X	
SNF Proposal			X	X	
SNF Approval			X	X	
Budget		X	X	X	X
Project financial report			X	X	
Project results report			X	X	

Figure 75: Support organization: Step 2 Decide

The result of this step is shown in Figure 76. *Support organization [w]* is refined in *Service organization[c]*, revealing its internal structure. The properties are allocated to the newly identified service systems based on the decisions in Figure 74. They reflect the fact which service system knows about what properties.

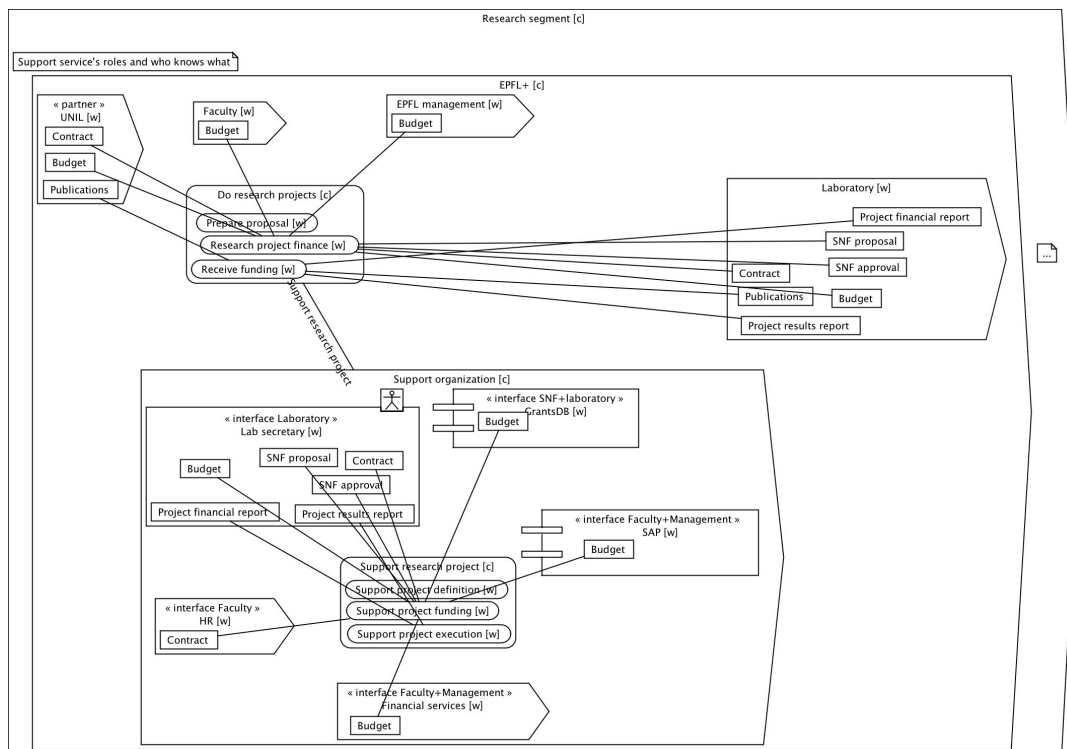


Figure 76: Support organization: Result of step 2

Step 3: Internal services

In this step, we need to identify the internal services of support organization needed to provide external services. Therefore, the external services are refined into many internal services. Also, the properties used by external services are allocated to the newly identified services. These decisions are captured in Figure 77 and Figure 78.



Figure 77: Support organization: Step 3 Conceptualize

	Prepare proposal	Research project finance	Get internal approval	...
Contract		X		
SNF Proposal		X		
SNF Approval		X		
Budget		X		
Project financial report			X	
Project results report			X	
Publications			X	

Figure 78: Support organization: Step 3 Decide

The result of this step is shown in Figure 79. We can see that support organization contains all the internal services and their relation to the properties, based on the decisions in Figure 77. The relations show which service uses what properties.

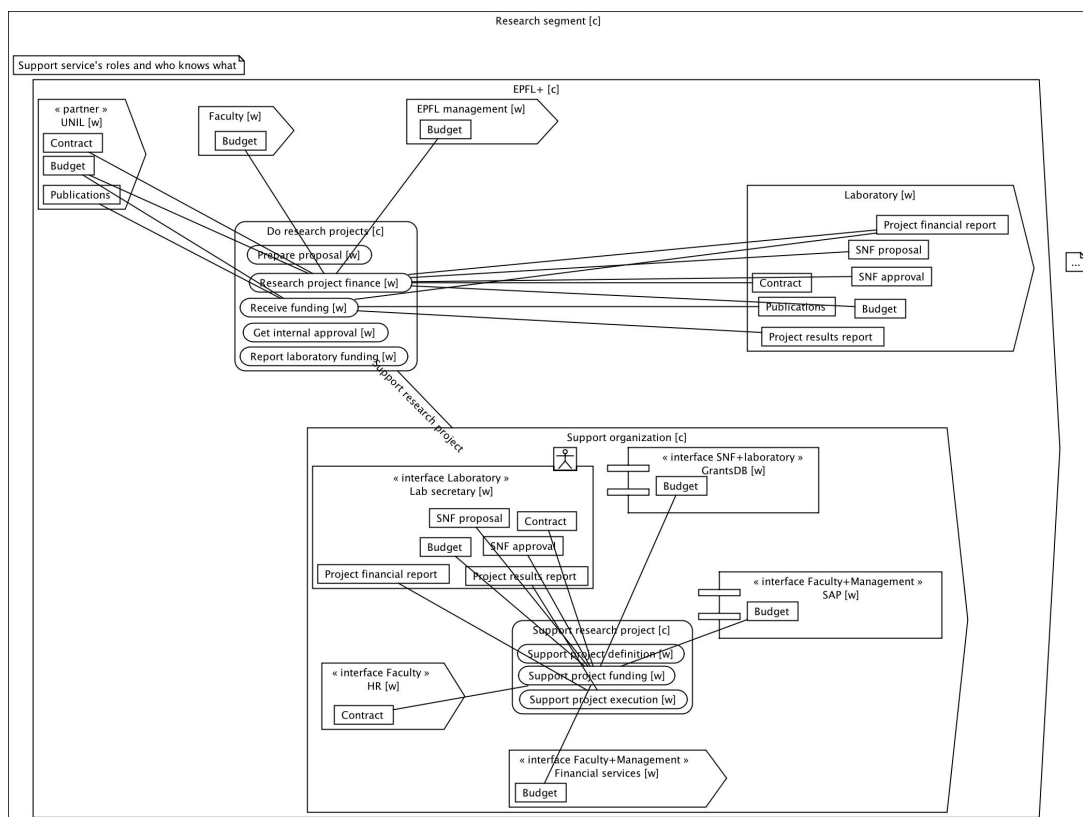


Figure 79: Support organization: Result of step 3

Step 4: Internal responsibilities

Finally, we need to define the responsibilities of all organization inside *Support organization[c]*. For each internal service, we identify the process and the services it uses to implement the higher-level service. We also allocate the service properties. The decisions are captured in Figure 80 and Figure 81.

Processes	
Prepare proposal	Lab secretary Prepare proposal (A,R)
Report project finance	...
Report project results	...
....	...

Figure 80: Support organization: Step 4 Conceptualize

	Lab secretary Prepare proposal	Lab secretary Report project finance	...
SNF Proposal	X		
....			

Figure 81: Support organization: Step 4 Decide

Figure 82 shows the result of this step. Support organization now contains all the details of service implementation: external services, internal organization, internal services and internal responsibilities of each of the service systems inside support organization. For the purpose of this project this was the level of abstraction that was enough. We started with the service Propagate research and analyzed its implementation on different levels: Research segment, EPFL+ and Support organization. In case we would like to do simulation and prototyping of the models, we would need to use functional units to describe the semantics of services. In this case, there was no need for that, so we did not model it.

relevant for service design and implementation and group them based on their role in providing the service.

As a holistic method, BASS has contributed to defining an overview of the whole process needed to provide and implement services, starting from customer's needs to definition of responsibilities of organizations needed to implement the required services. This way, we always first analyze customer's needs and then define internal organizations and their responsibilities based on services needed by customer. This helped to analyze the overall situation and agree about the responsibilities of all organizations participating in service delivery. It provided a common platform for customers, managers and employees and improved a common understanding of modeled services. It helped people and organizations to appreciate their role in the success of the service, by making them aware of their contributions to the final service and customer's satisfaction.

9.2 Comparison with Other Methods

In this chapter, we synthesize the comparison between different modeling methods explained in Chapter 2 (Table II), and position BASS in relation to these methods. Then, we focus on comparing the only modeling method intended for service design, service blueprinting and BASS.

From the table, we can conclude that simulation, model and refinement verification are applied mostly for the methods that are IT focused, show detailed view, and are used in late design phase. There are not many methods in which it is possible to simulate high-level models focusing on macro concepts. SysML and OPM are the only one providing that possibility. However, SysML is intended mostly for the specialized domains, such as aerospace, automotive, health care, and other. OPM can be applied to modeling organizations in general, but it is not intended for service design. Formal methods for model verification and simulation are applied mostly on the IT focused modeling methods, such as UML or for a specific domain.

BASS is a method used for early requirements that provides an integrated view of the process needed to define the services that need to be provided and organizations necessary for implementation. It is service focused and can be used for simulation and verification. Unlike many other methods, like for instance UML with 14 diagram types, this method uses one diagram type only including both the service system structure and its behavior. The benefits of having one diagram type are:

- Easier and quicker understanding of the method by the users
- Possibility to see the “big picture” showing both the structure and behavior of one service system
- Possibility to see the relation between the service system structure and its behavior.

For the later stages of design and for projects that include a lot low-level technical details, this can lead to cluttered and complex models. However, for the initial stages of the projects, these characteristics force people to focus on macro concepts, and that makes them very useful. BASS provides a way to create precise models on high-level of analysis, which can be simulated. In addition, BASS is also service focused,

meaning we always start by modeling customer's needs and define organizations necessary for service implementation based on their role in providing the service.

As BASS is intended for service design, we compare it in more details with service blueprinting. BASS shares many of the characteristics with service blueprinting. They both start with modeling customer's perspective of the service, continue with modeling points of interaction with the customer and support services necessary for delivering the services to the customer. They put all people involved in service design, delivery and support on the same page and force people to take cross-disciplinary and cross-functional view of a service. Customers, employees and managers all get the overview of the entire process and their role in it. Both methods always start with the customer's perspective and are therefore customer-oriented. However, there are several differences. Comparison between service blueprinting and our method is given in Table XIV. The text in right column in italics represents the differences of our method compared to service blueprinting.

Table XIV: Comparing service blueprinting and our method

Service Blueprinting	BASS
Customer-focused	Customer-focused
Common platform for customers, employees, managers	Common platform for customers, employees, managers
Provides overview of the entire process	Provides overview of the entire process
Puts everyone involved in the service design on the same page	Puts everyone involved in the service design on the same page <i>+ enforces identification of different teams relevant for customer's experience: service design, service delivery, service support and service promotion team</i>
Suitable for any level of analysis	Suitable for any level of analysis
Forces people to take cross-disciplinary and cross-functional view of a service	Forces people to take cross-disciplinary and cross-functional view of a service
Used for modeling new service and improving existing ones	Used for modeling new service and improving existing ones
Standard BPMN-like notation	<i>Richer notation capturing organizations, data, boundaries, IT system components, etc.</i>
Imperative process modeling (process oriented)	<i>Declarative process modeling (data oriented)</i>
Models classes of organizations and people	<i>Models instances of organizations and people</i>
No simulation possible	<i>Simulate the models to obtain feedback even at the early stage of design</i>

We explain the differences and when one can benefit from these different approaches. Service blueprinting is focused on modeling the service process itself. It uses the standard notation with boxes and arrows and sequence representation of the steps of the service process. It can use BPMN to represent more detailed representation of the process. BASS focuses not only on the service process modeled, but also on the organizations, sub-organizations, data and boundaries that define the context and life cycle for organizations, actions and data. Therefore, we use richer notation to represent this, such as visual representation for the IT systems and components.

Next, service blueprinting uses imperative and BASS declarative process modeling. Unlike imperative process modeling that uses predefined sequence of steps and restricts the user to perform the action in exactly given sequence of steps, declarative process modeling defines what should be done to achieve the goal without specifying the exact sequence of steps, leaving the user freedom to realize the action in the way he finds it appropriate. This can be useful for certain projects, such as for the workflows in hospital, where the doctor decides what to do based on the current situation, but still has to follow certain rules in performing it. Also, it can be very useful for initial stages of the projects, because it forces people to focus more on macro concepts, rather than on details of implementation. Depending on the nature and the project, and stage of analysis, imperative or declarative process modeling can be useful.

Unlike service blueprinting, with BASS we model only the instances of organizations, people and services. This means that we always model the specific customer, use the names of the people in the project, etc. This could lead to outdated models, when people are not in organizations any more. However, the main purpose of the models in BASS is to do analysis using the resources available at the current moment of time and to create common understanding of the situation. The process of creating the model brings new knowledge and conclusions and is therefore very important. The main purpose is not to update the model to reflect the current situation, but to analyze the existing situation or model the desired services in the given moment. Using actual names and instances helps people find their roles in the provisioning service.

Like with service blueprinting, with BASS we also put everyone involved in service design, delivery and support on the same page, helping them understand their role in the customer's experience. In addition, in one of the steps of the process for identifying service systems inside organization that participate in providing the service, we enforce identifying the main roles from service perspective, such as service design, service delivery, service support, and service promotion team. This enforces people to identify themselves as a part of the process delivering the given services, and not just as a part of traditional organization view, such as departments.

Finally, with the proposed method it is possible to simulate the models to obtain feedback on the instances of the service behavior even at early stages of design.

9.3 Conducted Survey: Practical Feedback

BASS is aiming at helping practitioners to create abstract, yet precise models. To evaluate if it is applicable in different businesses, we have conducted a survey with practitioners. We have conducted series of interviews with the practitioners having experience in different domains: management consulting, IT, healthcare and pharmacy, government, telecommunication, and others. The interview contained 30min explanation of BASS method illustrated on the case study of computer data storage we conducted at EPFL. Afterwards, the practitioners were asked to answer 12 questions related to their previous experiences with business case modeling and their opinion about applicability of BASS in their projects and companies. The evaluation document used to conduct an interview is available in **Appendix IV: Survey Questionnaire**. Note that during the interview protocol director was giving further information about the content of the document. Below, we summarize the results of the survey. Based on the answers we got, we discuss several topics: modeling methods used in practice, simulation methods used in practice, usefulness of BASS design process, usefulness of BASS simulation, main advantages and disadvantages of BASS, and suggested improvements. For each of them, we first give a synthesis of answers and then the excerpts from the questionnaire in italics.

Modeling methods and tools used in practice

House of Quality, Lean-Six Sigma, SEAM, Merise, Mega, Information Engineering (Navigator, IEM), RUP, RAD, ASAP, BPM, their own (or the one of the clients) methods and tools, created for specific problems.

- *"I usually use "House of Quality" – some of its parts or as a whole. Besides, the Lean-Six Sigma business process modeling, that I am a passionate advocate of, is all about customer centric tools and approaches."*
- *"Given the consulting role of the work we do, we typically adapt to client practices in this regard. Internally, there is not a consistent set of practices we employ, but we do come up with structures/methods that serve the purpose of a particular task at hand."*
- *"SEAM"*
- *"Yes, Merise, Mega, Information Engineering (Navigator, IEM), RUP, RAD, ASAP, SEAM, ..."*
- *"Yes, BPM or specific client methods"*.

Simulation methods and tools used in practice

Most of the interviewed practitioners do not use the simulation tools and methods. The only mentioned one was Oracle BPM. This confirms that usage of simulation tools for modeling business cases is not that common.

- *"I have never used any simulation, to be fair, due to my position that sits between business and IT."*
- *"I have personally not performed a simulation of this nature before, but was often engaged in reviewing the outputs and proposing actions to be taken based on them. I am not familiar with specific approaches/methods, as these tend to be client-specific in the context of my work."*

- *"I've never used model simulation but I saw it in some BPM tools (Oracle BPM)."*
- *"I have never used any simulation."*

The usefulness of BASS design process

All interviewed practitioners have agreed that BASS can be useful. What they have stressed the most is use of BASS for facilitating workshops and introducing structure to the design process, matching the business needs and technical requirements. Most of them stress the fact that it is useful for early stages of design.

- *"Definitely BASS can bring some new perspectives to my projects. I like the way of the presented structured way of thinking, and customer-centric approach – making sure that customer gets what customer wants. All the rest, (the back-office operations and organisation) is joining up to support the customer expectations."*
- *"BASS can definitely help to facilitate workshops with stakeholders. Depending on the stakeholders involved in the workshops, from my experience, it might be hard to keep technical people focused on the topic of "customer needs" and "capabilities" customer wants. Broadly generalising, technical people usually tend to steer the topics directly to the technical specification. However, with the good facilitation skills, I guess that this hurdle can be overcome."*
- *"In regards with the proposed approach, I think it would definitely drive some efficiencies and effectiveness during the workshop facilitation and bridging the gap between business needs and technical requirements. In regards with the simulation approach, I guess it could bring more clarity than Alloy diagrams."*
- *"I believe that the proposed method could be very valuable in the process of soliciting input and working out the optimal solution to underpin / adapt the business case. I think the method itself is supporting co-creation and is conducive to securing buy-in from the respective stakeholders."*
- *"As per above, I see facilitating workshops with BASS as one of the core strengths of the approach in this domain."*
- *"I believe that with practical improvements mentioned above, BASS could be a useful tool to bring what is otherwise a complex outcome closer to the understanding of stakeholders that are part of the process, but not necessarily versed in process design / notation, etc."*
- *"I believe that BASS, complete with the workshop approach built around the afore-mentioned visual modeling method, could be built into a close-ended solution/offering that could be 'deployed' in client situations, in particular those where new functionality/product/service/process design is the topic."*
- *"Yes, BASS can bring some new perspectives to my projects, to me it defines a framework for moving from business requirements to detailed software design."*
- *"BASS can be useful for facilitating workshops, because it gives some structure to the workshops."*
- *"BASS doesn't look like huge and having many constraints, thus it has chances to be adopted. "*

- *“Common sense would say that yes, BASS could help EPFL because it defines a framework, it looks like light and easily manageable.”*

The usefulness of BASS simulation

Most of the practitioners agree that BASS simulations can be useful for testing and improving a proposed solution. However, the suitability of visual representation is one of the biggest drawbacks. All practitioners found very difficult to interpret the results of simulation shown using Alloy diagrams. Transformation back to BASS has significantly improved their understanding of the simulation results. However, there are still suggestions for improvement.

- *“In regards with the usefulness, in line with my previous answer, I think it is applicable for the technically demanding projects. BASS instances are definitely more readable than the previous method (Alloy) that was presented to me.”*
- *“I guess it would be a useful simulation method that puts a structure around the approach.”*
- *“I believe simulating the model is relevant for testing and improving a proposed solution. I find the BASS simulation to be useful in the aforementioned context.”*
- *“Useful to detail further / validate requirements. Simulation and prototyping are always a source of confusion for business people. It still require the presence of business analyst or IT engineer to read/validate the results of simulation. Simulation is not needed always, but only when the further analysis is required to understand the “real” problem.”*

Advantages

Some of the biggest advantages of BASS that were mentioned are: structured way of thinking that BASS brings, customer-focused analysis, having a complete overview of a process, including external organizations as a part of service provider, including people from different departments, usage of declarative process modeling in initial stages of the project.

- *“I like the way of the presented structured way of thinking, and customer-centric approach – making sure that customer gets what customer wants. All the rest, (the back-office operations and organisation) is joining up to support the customer expectations.”*
- *“In my opinion, including people from different departments is one of the crucial steps in ensuring the successful closure of an initiative. More and more organisations are moving from silos-based operations that are organised around different operations departments are delivering (horizontal approach) to opposite approach that links together different, usually not connected teams on a project-to-project bases (vertical approach). This type of organisation supposes to drive a delivery time decrease and productivity increase.”*
- *“Including external organisations as a part of service provider is a common approach in large international companies. Externals can be managed differently depending on the expected outcome. The 3rd party companies can be involved as contingency workers, consultants, out-sources, smart-sources,*

or considered equally as permanent employees. Therefore, including those employees is of a high value to the project.”

- *“I find it insightful and, at times, even critical, to have a complete overview of the inputs (which can be both data and behavior of the modeled organization) when conducting an analysis or designing a process. The more one is able to consider at a time (without cluttering the picture), the better.”*
- *“I believe refocusing attention away from the sequence of steps can be useful, especially in the (early) design stages of new process creation or existing process modification. This in particular applies for the segments where key requirements are being defined by potential process users/beneficiaries.”*
- *“Declarative process modeling is definitely useful in brainstorming phase or an initial stage of the project planning when all aspects of the project should be covered. “*
- *“Being aware of the transversality of a process, involved external organizations, support, promotion and service’s delivery is always useful and can avoid surprises in the process implementation.”*

Disadvantages/Concerns

One of the main drawbacks of BASS is the suitability of the visual representation used in simulation. Another concern is how the models would look like for a more complex project. Despite liking the idea of using actual names in the models, interviewees have also stressed possible problems, such as problems with outdated models, drawing an inference that the setup is not working because a concrete person is not doing his job. Also, they have stressed that compatibility of BASS and Hermes would be very important and that this should be checked.

- *“I found the notation of the simulation results a bit challenging to interpret.”*
- *“However, I am not sure how cluttered the graph could look like if a more complex project is managed. In that case, I would still keep the approach, but organise the graphics a bit differently maybe. I can see using this approach in projects initiation phases, as well as during the problem solving and route-cause analysis sessions.”*
- *“However, in this particular case, if you want to keep the graphs for later onboarding or training purposes, there might be some problems with them being outdated. (E.g. if an employee moves on, an administrator should go through every single chart and update it with new names.) “*
- *“Depending on the setting (e.g., workshop), one would need to be careful that the effort to put actual names does not backfire (e.g., drawing an inference that the setup is not working because listed John Doe is just not doing what they are supposed to be doing).”*
- *“I think that BASS simulation can be useful but I wonder how it can be managed for complex systems.”*
- *“I also wonder if BASS is compliant with Hermès; I would say yes, but this has to be showed. If not, this could be a great disadvantage.”*
- *“Another important question is if BASS is compliant with agile methods, which tend to use as less methodology as possible.”*

- *“For high level diagrams, it is definitely useful to have one diagram type. For more details diagrams (and requirements) this can lead to messy and unreadable diagrams.”*

Suggested Improvements

Most of the suggested improvements are related to the results of simulation. Most practitioners have agreed that more user-friendly results of simulation are necessary. Transforming Alloy diagrams back to BASS has improved their readability significantly, but there is still place for improvement. Some of the suggestions are to highlight the key differences between the two states in simulation, simulating the time needed for services, detecting bottlenecks, etc. In addition to suggestions related to the results of simulation, interviewees have also suggested that leaving the possibility to use imperative process modeling is very important. Despite their appreciation of declarative process modeling, they believe that for certain stages of the project and for certain situations, it is useful to have also the possibility to model sequence of steps.

- *“I would probably prefer to see, as a result of simulation, more actionable information (all possible scenarios, bottlenecks and potential hazards) in a more user-friendly representation.”*
- *“In terms of practical improvements, I believe highlighting the key differences between the two states would go a long way. Separately, in more complex simulations, I believe it would be important to be able to trace-back/see interim steps.”*
- *“I would like to see the relationships between different types of requirements (functional, non-functional, security, data, business rules, etc.).”*
- *“If developed into a design that would be more ‘user-friendly’ for the broader set of stakeholders that (management) consultants typically interact with, BASS would be very useful.”*
- *“It could be interesting to see if the output can be graphically improved for more readability. Optimal readability is often a key success factor when humans are involved.”*
- *“If you could specify the approx. time needed for each step and the statistical law followed by events generation, you could simulate if the system will have bottlenecks, ...”*
- *“However, once when all operational items have been worked out, I strongly suggest placing every single activity, responsible person, clear deliverable and deadline in a sequence of steps.”*
- *“The main issue with declarative process modeling (excluding sequence) is that people are used to think with sequence instead of pre- and post-conditions. I think the better is not to exclude them in order to avoid people’s reluctance.”*

The feedback we got from practitioners is very valuable for this work and helps us to prioritize the directions of this research for the future.

10 Conclusion and Future Research Directions

In this thesis, we propose a visual modeling method BASS for service specification, verification and validation. It enables business/IT analysts to create abstract, yet precise models for business cases they are working on. *Abstraction* (from the Latin *abs*, meaning *away from* and *trahere*, meaning *to draw*) is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. Therefore, abstract models enable us to focus on macro concepts, which is necessary in the initial stages of the projects for which BASS is intended. To make sure that the models are abstract, but not vague, we add semantics to the modeling concepts that enable us to translate models to formal specifications, such as Alloy, or to Java language. We can then simulate and prototype the models. This way, we can validate the models with stakeholders, including their feedback even at early stages of design. Validation enables to detect and resolve anomalies making the model more precise. BASS can be used by business/IT analysts to create models of business cases or to facilitate workshops with different stakeholders. The research has been conducted following the guidelines of Design Science. Therefore, in a summary, we explain the theoretical contribution and practical recommendations for the professionals in the field of service design. We also explain limitations and drawbacks of the proposed method.

10.1 Theoretical Contribution

The contributions of our work are threefold. First, we provide the visual formalism for service specification and simulation, by adding the necessary concepts, such as functional unit, event, send and receive property, to the existing method SEAM. Second, we define a set of spirals: for service specification and service validation and verification. The service specification spiral enables us to keep the relation between several service specifications. It includes the steps with explicit design decisions on how to refine high-level specifications to include all details necessary for providing the identified services. The validation and verification spiral is used to validate and verify specifications on any level of abstraction. Finally, it provides an environment that enables simulation and prototyping of service specifications that are then used for their own validation and verification.

10.2 Practical Recommendations

In addition to the theoretical contribution to the knowledge base of service design and to SEAM, we also provide the tools and guidelines that help business and IT analysts to create and validate a service model, as confirmed by a survey conducted with practitioners.

We have conducted two consulting projects at EPFL using BASS method: Research project management and Computer data storage. In addition to the projects we conducted, we organized a series of interviews with practitioners from different fields to evaluate the applicability of the method in practice. An evaluation confirmed

that BASS could be useful in practice for facilitating workshops with stakeholders and in initial stages of the projects. It also made us aware of some limitations and possible problems with applying the method in practice. Based on the work presented in this thesis, in case studies we conducted and the data collected from practitioners in a survey, the following recommendations are proposed to professionals engaged in service specification:

- **Following the steps of the spiral, create a complete view of the service, from analysis of customer's needs to the organizations needed to implement service and their responsibilities**

In the service design process, follow the steps of the service specification spiral:

0) Initial model design

- 1) Add detailed external services to accommodate interactions with customers
- 2) Add internal organization and collaboration that implements external services together
- 3) Add internal collaborations necessary for internal processes
- 4) Define internal services and responsibilities of each internal organization.

In each step, make explicit the design decisions that need to be made by identifying the new concepts and allocating the existing ones to them. The same steps can be repeated inside each of internal organization. This way, we always first analyze the customer's needs and then define internal organizations and their responsibilities, based on services needed by the customer. This can help us to analyze the overall situation and agree about the responsibilities of all the organizations participating in service delivery. It provides a common platform for customers, managers and employees, and improves a common understanding of modeled services. It helps people and organizations to appreciate their role in service's success, by making them aware of their contributions to the final service and the customer's satisfaction.

- **Use service-based definition of internal organizations needed to implement the service**

When defining organizations needed to provide the necessary services, do not necessarily use an organizational chart. Instead, define organizations based on their role in providing the service. This means, people from different departments can be part of the same organization. External organizations can also be modeled as a part of the service provider. This way, we put together all the people and organizations relevant for providing and implementing services together. When defining organizations needed to provide the necessary services, always keep in mind the delivery, support and promotion of a service.

- **Use simulation to validate the models with stakeholders and create abstract, yet precise models**

Once the overall structure and responsibilities are defined in the models, use the simulation and prototyping tools to detect possible anomalies in the models, discover missing business rules, and others. This can help to obtain feedback from stakeholders about the behavior of modeled services. This way, we create abstract, yet precise models.

10.3 Limitations and Drawbacks

BASS is intended for the initial stages of the analysis of business cases. Therefore, it has certain characteristics that are very useful for high-level analysis, but are not efficient when applied to the low-level implementation models. One of them is the fact that BASS has one diagram type that captures both behavior and data of the modeled system. For very technical models showing all the details and data needed for implementation, this could lead to complex and cluttered models. However, this is very powerful for high-level analysis as it forces people to think about high-level concepts and not go too much into details that drag their attention from the main goals.

Similarly, declarative process modeling can be useful both for early and later stages of design. The conducted survey has shown that practitioners prefer to use sequential modeling in the later stages of the project, but that declarative process modeling can be powerful in the initial phases. Also, as BASS is intended for high-level analysis, the prototyping tool does not include requirements such as security. However, for full code generation, it would be useful to include this as well.

There are several limitations of the simulation tool. As it is based on Alloy, which is a model checker and not a theorem prover, if we do not find the counterexample for the statement we want to prove it holds, it does not mean that the statement is valid; it just proves that it holds in the given domain. Next, it is possible to model only the constraints including simple arithmetic operations. Alloy is based on relations and generates the instances of the model by analyzing all possible variable bindings in the given domain. Therefore, it uses only a limited number of integers, to avoid the state explosion.

Based on the survey conducted with practitioners, one of the main drawbacks of BASS is the suitability of the current visual representation used for simulation. Alloy diagrams are shown to be unreadable to business experts. Readability is significantly improved by the transformation of Alloy instances to BASS models. However, this can be still improved by marking explicitly possible problems in the generated BASS models, simulation of the time needed for services to be performed, detecting the bottlenecks, and others.

10.4 Future Work

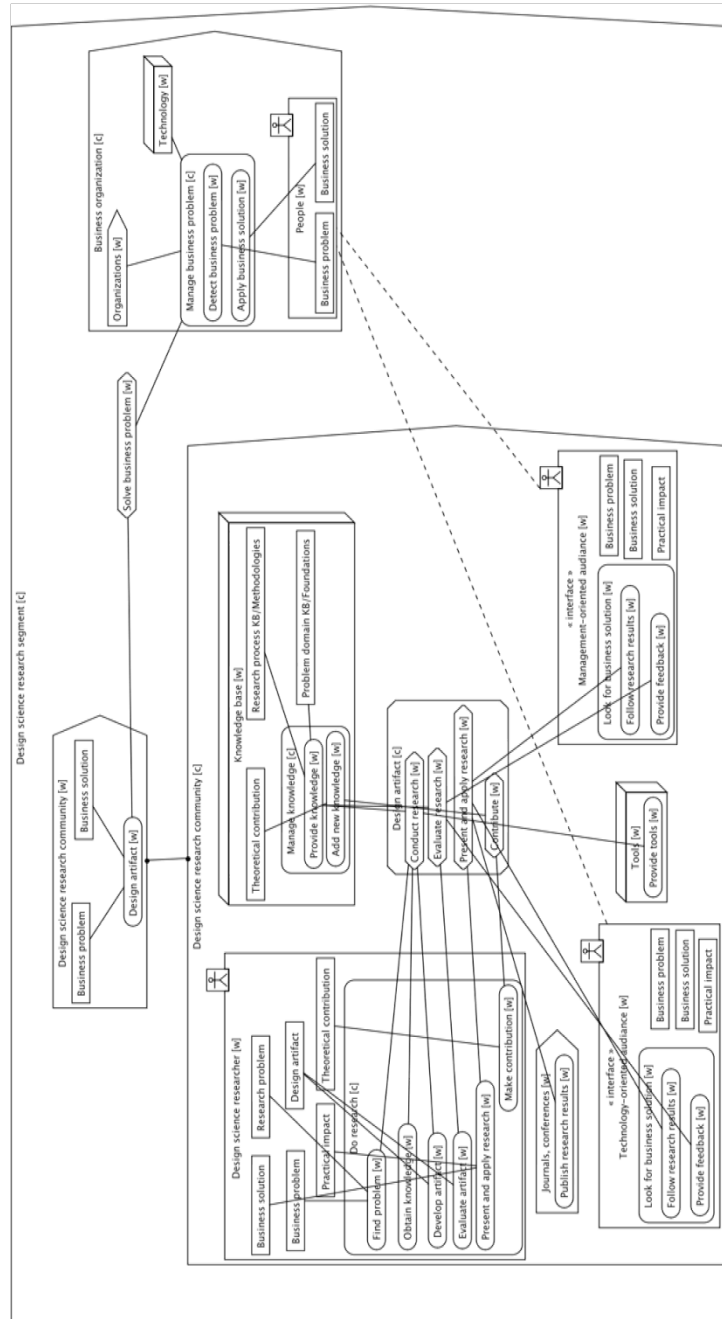
Based on the current work, the following research directions will be useful for an improved adaption of BASS method:

- Visual representation of Alloy expressions**
 To formalize BASS models, we use the semantics that can be mapped to Alloy specification. For example, service is expressed with its precondition (Alloy expression) and the effect (Alloy expression). To simplify usage of Alloy expressions in modeling, we have introduced functional units, as atomic units of logic that have their visual representation. Therefore, it is possible to show the effect of the service using visual representation and the logic defined in the library of functional units, which once defined can be used in any project. Currently, preconditions still need to be expressed using Alloy expressions in the model. Therefore, it would be useful to create an environment in which it would be possible to define these constraints in a more user-friendly form, such as when defining special conditions in BPMN diagrams. They could be expressed as a combination of data in the diagram and arithmetical and logical operations.
- Combining declarative and imperative process modeling**
 Currently, we use declarative process modeling in BASS. This means, we do not define the sequence of steps in which the process is being executed. Instead, we define just the overall responsibilities. As confirmed by the survey conducted with practitioners, this is very useful at the beginning of analysis. However, practitioners prefer to use as well sequential modeling for organizing their projects better. Therefore, it would be useful to do deeper analysis of what would be useful for practitioners and, based on that, to make possible to use both imperative and declarative way of process modeling. For instance, to create a language that could be a combination of declarative and imperative modeling concepts.
- Synthesis of Java code from Alloy expressions**
 To simulate models with BASS, we use functional units as the main units of logic to express the effect of the service. During modeling, it is possible to use functional units defined in the special library of the functional units. Once it is defined in the library, it can be used in any other project. Currently, when the designer wants to add new functional unit in BASS environment, he needs to define its semantics both in Alloy and Java. This could be improved by creating the tool that would transform Alloy predicates and functions to Java methods. The designer would need to define the semantics only in one of the languages.
- Translating business rules from natural language to Alloy specification**
 In order to simplify the translation of business rules to formal methods expressions, such as Alloy, it would be useful to create a knowledge base for the business rules that could be used and translated to the formal expressions.

- **Make explicit the problems detected with simulation**

Currently, with BASS simulation we can observe the state of the system before and after the service execution. Sometimes, it can be difficult to detect potential problem by just observing these two states, as confirmed by the survey with practitioners. It would be useful to improve simulation by showing the errors explicitly. This could be done, for example, by checking the specific rules that are common to many projects and that could be kept in the library of rules.

Appendix I: SEAM Representation of Hevner Research Framework



Appendix III: Case of Order Creation at Générale Ressorts in Alloy

Model 1: GR[w], Order Creation[w]

```
open predefined_find [Part, CustomerPartId]
open predefined_find [Customer, Name]
open predefined_create [Part, Customer, OrderConfirmed]
open predefined_add [OrderConfirmed]

sig Segment_ressort {
  Segment_ressort__Generale_Ressorts: one Generale_Ressorts
}

sig Generale_Ressorts {
  Generale_Ressorts__OrderConfirmedwoOut: one OrderConfirmed ,
  Generale_Ressorts__OrderConfirmedSetset: one OrderConfirmedSet ,
  Generale_Ressorts__OrderInitialwoIn: one OrderInitial ,
  Generale_Ressorts__PartSetset: one PartSet ,
  Generale_Ressorts__CustomerSetset: one CustomerSet
}

sig OrderConfirmed {
  OrderConfirmed__Customer: one Customer,
  OrderConfirmed__Part: one Part
}

sig OrderConfirmedSet {
  OrderConfirmedSet__OrderConfirmed: set OrderConfirmed
}

sig OrderInitial {
  OrderInitial__Name: one Name ,
  OrderInitial__CustomerPartId: one CustomerPartId
}

sig PartSet {
  PartSet__Part: set Part
}

sig Part {
  Part__CustomerPartId: one CustomerPartId,
  Part__GRPartId: one GRPartId
}

sig CustomerSet {
  CustomerSet__Customer: set Customer
}

sig Customer {
  Customer__Name: one Name,
  Customer__Address: one Address
}
```

```

sig GRPartId { }

sig Name { }
sig Address { }

sig CustomerPartId { }

pred      simulate(Segment_ressort_pre,      Segment_ressort_post:      one
Segment_ressort, Part1: one Part, Customer2: one Customer, OrderConfirmed3:
one OrderConfirmed, OrderConfirmedset4: set OrderConfirmed)
{

Part1 = find[Part__CustomerPartId,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__Or
derInitialwoIn.OrderInitial__CustomerPartId,
      Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__PartSetset.PartSet__Part,
      Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Resso
rts__PartSetset.PartSet__Part]

and one Part1

and Customer2 = find[Customer__Name,
      Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OrderInitialwoIn.OrderInitial__Name,
      Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__CustomerSetset.CustomerSet__Customer,
      Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Resso
rts__CustomerSetset.CustomerSet__Customer]

and one Customer2

and OrderConfirmed3 = create[Part1, Customer2, OrderConfirmed__Part,
OrderConfirmed__Customer]

and one OrderConfirmed3

and OrderConfirmedset4 = add1[OrderConfirmed3,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__Or
derConfirmedSetset.OrderConfirmedSet__OrderConfirmed]

and one OrderConfirmedset4

      and
      Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Resso
rts__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed
= OrderConfirmedset4

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
rderConfirmedwoOut
= OrderConfirmed3
and Segment_ressort_pre not= Segment_ressort_post
}

assert CustomerAssertion{
Customer in CustomerSet.CustomerSet__Customer
}

check CustomerAssertion
run simulate for 2

```

Model 2: GR[c], Order Creation[w]

```
open predefined_find [Part, CustomerPartId]
open predefined_find [Customer, Name]
open predefined_create [Part, Customer, OrderConfirmed]
open predefined_add [OrderConfirmed]

sig Segment_ressort {
  Segment_ressort__Generale_Ressorts : one Generale_Ressorts
}

sig Generale_Ressorts {
  Generale_Ressorts__OEP : one OEP ,
  Generale_Ressorts__ERP : one ERP
}

sig OEP {
  OEP__OrderInitialwoIn : one OrderInitial ,
  OEP__OrderConfirmedwoOut : one OrderConfirmed
}

sig OrderInitial {
  OrderInitial__CustomerPartId : one CustomerPartId ,
  OrderInitial__Name : one Name
}

sig OrderConfirmed {
  OrderConfirmed__Customer : one Customer ,
  OrderConfirmed__Part : one Part
}

sig ERP {
  ERP__OrderConfirmedSetset : one OrderConfirmedSet ,
  ERP__CustomerSetset : one CustomerSet ,
  ERP__PartSetset : one PartSet
}

sig OrderConfirmedSet {
  OrderConfirmedSet__OrderConfirmed : set OrderConfirmed
}

sig CustomerSet {
  CustomerSet__Customer : set Customer
}

sig Customer {
  Customer__Address : one Address ,
  Customer__Name : one Name
}

sig PartSet {
  PartSet__Part : set Part
}

sig Part {
  Part__CustomerPartId : one CustomerPartId ,
  Part__GRPartId : one GRPartId
}
```

```

sig Address { }
sig CustomerPartId { }

sig Name { }

sig GRPartId { }

pred simulate( Segment_ressort_pre, Segment_ressort_post : one
Segment_ressort, Part1 : one Part, Customer2 : one Customer, OrderConfirmed3
: one OrderConfirmed, OrderConfirmedset4 : set OrderConfirmed ) {

Part1 = find[Part__CustomerPartId,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__OE
P.OEP__OrderInitialwoIn.OrderInitial__CustomerPartId,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__PartSetset.PartSet__Part,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__PartSetset.PartSet__Part]

and one Part1

and Customer2 = find[Customer__Name,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__OE
P.OEP__OrderInitialwoIn.OrderInitial__Name,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__CustomerSetset.CustomerSet__Customer,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__CustomerSetset.CustomerSet__Customer]

and one Customer2

and OrderConfirmed3 = create[Part1, Customer2, OrderConfirmed__Part,
OrderConfirmed__Customer]

and one OrderConfirmed3

and OrderConfirmedset4 = add1[OrderConfirmed3,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed ]

and one OrderConfirmedset4

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed =
OrderConfirmedset4

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__OrderConfirmedwoOut = OrderConfirmed3

and Segment_ressort_pre not= Segment_ressort_post
}
run simulate for 2

```

Model 3: GR[c], Order Creation Process[w]

```
open predefined_add [Name]
open predefined_find [Customer,Name]
open predefined_add [CustomerPartId]
open predefined_find [Part,CustomerPartId]
open predefined_create [Customer,Part, OrderConfirmed]
open predefined_add [OrderConfirmed]
open predefined_add [OrderConfirmed]

sig Segment_ressort {
  Segment_ressort__Generale_Ressorts : one Generale_Ressorts
}

sig Generale_Ressorts {
  Generale_Ressorts__OEP : one OEP ,
  Generale_Ressorts__ERP : one ERP
}

sig OEP {
  OEP__OrderInitialwoIn : one OrderInitial ,
  OEP__OrderConfirmedwoOut : one OrderConfirmed
}

sig OrderInitial {
  OrderInitial__CustomerPartId : one CustomerPartId ,
  OrderInitial__Name : one Name
}

sig OrderConfirmed {
  OrderConfirmed__Customer : one Customer ,
  OrderConfirmed__Part : one Part
}

sig ERP {
  ERP__OrderConfirmedSetset : one OrderConfirmedSet ,
  ERP__CustomerSetset : one CustomerSet ,
  ERP__PartSetset : one PartSet
}

sig OrderConfirmedSet {
  OrderConfirmedSet__OrderConfirmed : set OrderConfirmed
}

sig CustomerSet {
  CustomerSet__Customer : set Customer
}

sig Customer {
  Customer__Address : one Address ,
  Customer__Name : one Name
}

sig PartSet {
  PartSet__Part : set Part
}

sig Part {
  Part__CustomerPartId : one CustomerPartId ,
```



```

    Part__GRPartId : one GRPartId
}

sig Address { }

sig CustomerPartId { }

sig Name { }

sig GRPartId { }

pred simulate(    Segment_ressort_pre,    Segment_ressort_post    :    one
Segment_ressort, Name1 :    Name, Customer2 : one Customer, CustomerPartId3 :
CustomerPartId, Part4 : one Part, OrderConfirmed5 : one OrderConfirmed,
OrderConfirmedset6 : set OrderConfirmed, OrderConfirmed7 : OrderConfirmed )
{
    Name1 =
    enter[Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OEP.OEP__OrderInitialwoIn.OrderInitial__Name ]

    and one Name1

    and Customer2 = find[Customer__Name, Name1,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__CustomerSetset.CustomerSet__Customer,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__CustomerSetset.CustomerSet__Customer]

    and one Customer2

    and CustomerPartId3 =
    enter[Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OEP.OEP__OrderInitialwoIn.OrderInitial__CustomerPartId ]

    and one CustomerPartId3

    and Part4 = find[Part__CustomerPartId, CustomerPartId3,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__PartSetset.PartSet__Part,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__PartSetset.PartSet__Part]

    and one Part4

    and OrderConfirmed5 = create[Customer2, Part4, OrderConfirmed__Customer,
OrderConfirmed__Part]

    and one OrderConfirmed5

    and OrderConfirmedset6 = add1[OrderConfirmed5,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed ]

    and one OrderConfirmedset6

    and
    Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed =
    OrderConfirmedset6

    and OrderConfirmed7 = get[OrderConfirmed5 ]

```

```
and one OrderConfirmed7

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__OrderConfirmedwoOut = OrderConfirmed7

and Segment_ressort_pre not= Segment_ressort_post
}

run simulate for 2
```

Model 4: GR[c], Order Creation Process[c]

```
open predefined_add [CustomerPartId]
open predefined_add [Name]
open predefined_add [OrderConfirmed]
open predefined_find [Customer,Name]
open predefined_find [Part,CustomerPartId]
open predefined_create [Customer,Part, OrderConfirmed]
open predefined_add [OrderConfirmed]

sig Segment_ressort {
  Segment_ressort__Generale_Ressorts : one Generale_Ressorts
}

sig Generale_Ressorts {
  Generale_Ressorts__ERP : one ERP ,
  Generale_Ressorts__OEP : one OEP
}

sig ERP {
  ERP__PartSetset : one PartSet ,
  ERP__OrderConfirmedwoOut : one OrderConfirmed ,
  ERP__OrderConfirmedSetset : one OrderConfirmedSet ,
  ERP__CustomerSetset : one CustomerSet ,
  ERP__CustomerPartIdwoIn : one CustomerPartId ,
  ERP__NamewoIn : one Name
}

sig PartSet {
  PartSet__Part : set Part
}

sig Part {
  Part__CustomerPartId : one CustomerPartId ,
  Part__GRPartId : one GRPartId
}

sig OrderConfirmedSet {
  OrderConfirmedSet__OrderConfirmed : set OrderConfirmed
}

sig CustomerSet {
  CustomerSet__Customer : set Customer
}

sig Customer {
  Customer__Address : one Address ,
  Customer__Name : one Name
}

sig OEP {
  OEP__OrderInitialwoIn : one OrderInitial ,
  OEP__CustomerPartIdwoOut : one CustomerPartId ,
  OEP__OrderConfirmedwoOut : one OrderConfirmed ,
  OEP__OrderConfirmedwoIn : one OrderConfirmed ,
  OEP__NamewoOut : one Name
}

sig OrderInitial {
```

```

    OrderInitial__CustomerPartId : one CustomerPartId ,
    OrderInitial__Name : one Name
}

sig OrderConfirmed {
    OrderConfirmed__Customer : one Customer ,
    OrderConfirmed__Part : one Part
}

sig Address { }

sig Name { }

sig CustomerPartId { }

sig GRPartId { }

pred    OEPAction(    Segment_ressort_pre,    Segment_ressort_post    :    one
Segment_ressort,    CustomerPartId1 :    CustomerPartId,    Name2 :    Name,
OrderConfirmed3 : OrderConfirmed)
{
CustomerPartId1 =
enter[Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OEP.OEP__OrderInitialwoIn.OrderInitial__CustomerPartId ]

and one CustomerPartId1

and Name2 =
enter[Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressor
ts__OEP.OEP__OrderInitialwoIn.OrderInitial__Name ]

and one Name2

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__CustomerPartIdwoOut = CustomerPartId1

and OrderConfirmed3 =
get[Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts
__OEP.OEP__OrderConfirmedwoIn ]

and one OrderConfirmed3

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__NamewoOut = Name2

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__OrderConfirmedwoOut = OrderConfirmed3

and Segment_ressort_pre not= Segment_ressort_post
}
pred    ERPAction(    Segment_ressort_pre,    Segment_ressort_post    :    one
Segment_ressort, Customer4 : one Customer, Part5 : one Part, OrderConfirmed6
: one OrderConfirmed, OrderConfirmedset7 : set OrderConfirmed )
{
Customer4 = find[Customer__Name,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__NamewoIn,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER

```

```

P.ERP__CustomerSetset.CustomerSet__Customer,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__CustomerSetset.CustomerSet__Customer]

and one Customer4

and Part5 = find[Part__CustomerPartId,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__CustomerPartIdwoIn,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__PartSetset.PartSet__Part,
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__PartSetset.PartSet__Part]

and one Part5

and OrderConfirmed6 = create[Customer4, Part5, OrderConfirmed__Customer,
OrderConfirmed__Part]

and one OrderConfirmed6

and OrderConfirmedset7 = addl[OrderConfirmed6,
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed ]

and one OrderConfirmedset7

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__OrderConfirmedSetset.OrderConfirmedSet__OrderConfirmed =
OrderConfirmedset7

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__OrderConfirmedwoOut = OrderConfirmed6

and Segment_ressort_pre not= Segment_ressort_post
}

pred simulate( Segment_ressort_pre, Segment_ressort_post : one
Segment_ressort, CustomerPartId1 : CustomerPartId, Name2 : Name,
OrderConfirmed3 : OrderConfirmed, Customer4 : one Customer, Part5 : one
Part, OrderConfirmed6 : one OrderConfirmed, OrderConfirmedset7 : set
OrderConfirmed )
{
OEPAction [ Segment_ressort_pre, Segment_ressort_post, CustomerPartId1,
Name2, OrderConfirmed3 ]

and ERPAction [ Segment_ressort_pre, Segment_ressort_post, Customer4, Part5,
OrderConfirmed6, OrderConfirmedset7 ]

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__NamewoOut =
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__NamewoIn

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__E
RP.ERP__OrderConfirmedwoOut =

```

```

Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__OE
P.OEP__OrderConfirmedwoIn

and
Segment_ressort_post.Segment_ressort__Generale_Ressorts.Generale_Ressorts__O
EP.OEP__CustomerPartIdwoOut =
Segment_ressort_pre.Segment_ressort__Generale_Ressorts.Generale_Ressorts__ER
P.ERP__CustomerPartIdwoIn
Segment_ressort_pre not = Segment_ressort_post
}

run simulate for 2

```

Appendix IV: Survey Questionnaire

STUDY TITLE: Evaluation of the Method Business Animated Service Specification (BASS)

Protocol Director: Biljana Bajić-Bizumić <biljana.bajic@epfl.ch>

DESCRIPTION: You are invited to participate in a **research study** “Evaluation of the Method Business Animated Service Specification (BASS)”. The purpose of this study is to evaluate the method named BASS used for business case modeling. The results of evaluation will be included in the PhD Thesis “Animation-Based Service Specification, Validation and Verification” done at EPFL by Biljana Bajić-Bizumić. Your cooperation will consist of participation in an interview. No preparation is required.

TIME INVOLVEMENT: Your participation will consist of 1-1.5h interview in which the protocol director will illustrate the method with a case study of computer data storage at EPFL and then ask you several questions regarding your usage of modeling methods for business cases and usefulness of BASS in practice.

RISKS AND BENEFITS: The risk associated with this study is **non-existing**. The benefit, which may reasonably be expected, is the gradual improvement of the BASS method to accommodate the needs of practitioners more.

SUBJECT'S RIGHTS: If you have read this form and have decided to participate in this project, please understand your **participation is voluntary** and you have the **right to withdraw your consent or discontinue participation at any time without penalty**. The **alternative is not to participate**. You have the right to refuse to answer particular questions. **If you refuse to participate, please note it on this form and sign.**

CONTACT INFORMATION:

Questions: If you have any questions, concerns or complaints about this research, its procedures, risks and benefits, contact the Protocol Director (the person who’s conducting the test).

Independent Contact: If you are not satisfied with how this study is being conducted, or if you have any concerns, complaints, or general questions about the research or your rights as a participant, please contact the management of LAMS Lab at EPFL to speak to the supervisor of the project Alain Wegmann <alain.wegmann@epfl.ch>.

I give consent to use the data gathered from my input and to process them for research purposes.

(please check only one)

☐ Yes

☐ No

The extra copy of this consent form is for you to keep.

SIGNATURE _____ **DATE** _____

Your feedback is extremely valuable for the purpose of research and BASS applications.

Thank you very much for taking the time to participate in this interview!

Computer Data Storage at EPFL

EPFL is university based in Lausanne, Switzerland, providing education to 6300 students employing 6800 teaching, scientific and administrative staff. All students and employees are using data storage services provided by EPFL to store their documents, multimedia files, scientific data and others. The contract for current storage system ends and EPFL needs to decide what storage system to buy as an upgrade of the existing one.

The main goal of this project is to propose the specification for the new technical solution for storage service. The expected outcomes of the project are:

- Requirements needed for the Request for Proposal (RFP) for the new storage system, such as volume of storage, technical characteristics and other.
- Definition of internal EPFL organization and responsibilities of each of the teams in providing required storage services to its students and employees.

In order to derive necessary requirements, we use BASS to model the new solution for storage services at EPFL, starting with the customer's needs through organization needed to implement the required services.

We also illustrate simulation and prototyping possibility of BASS to discover new business rules and identify possible problems with the stakeholders.

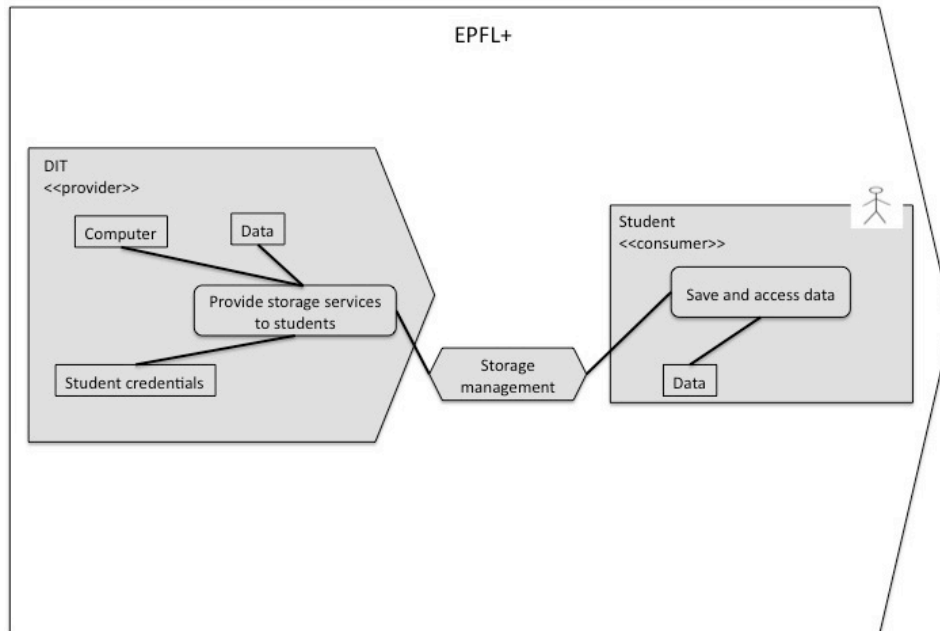
Computer Data Storage at EPFL: BASS Models

BASS can be used either to model a business case or to facilitate a workshop with stakeholders. In this project it was used for both. Here, we illustrate how we have used BASS to create a model for the new storage solution. We show only the final models, illustrating the final solution for the specific scenario. The models are results of several discussions with stakeholders through which we built our understanding of the project by improving the models. The models were used as a means of communication with necessary stakeholders.

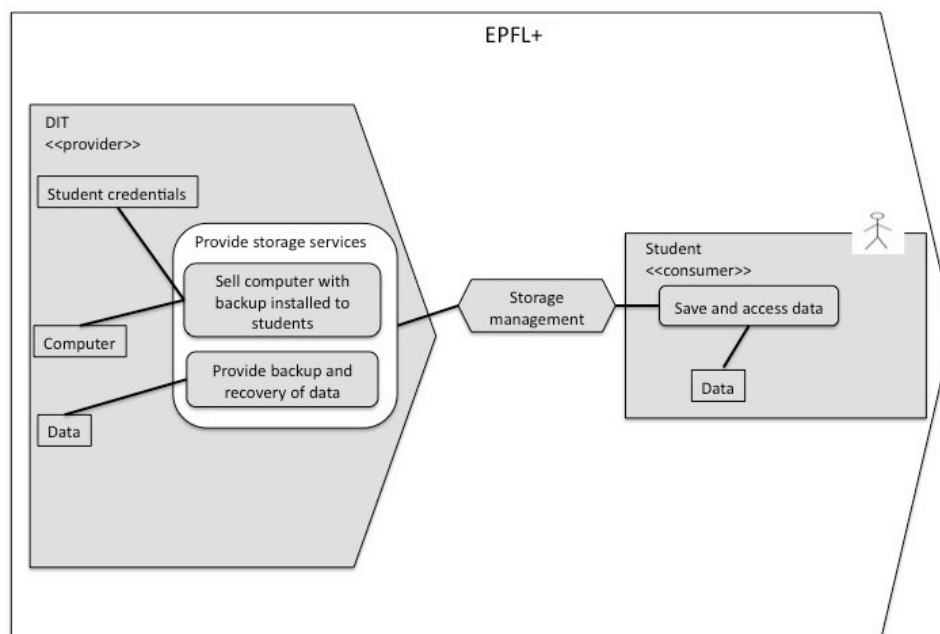
We model the new storage system for EPFL following five steps of BASS method.

CUSTOMER

Step 1: Define external service with a customer



Step 2: Add detailed external services to accommodate the expected interactions with the customer

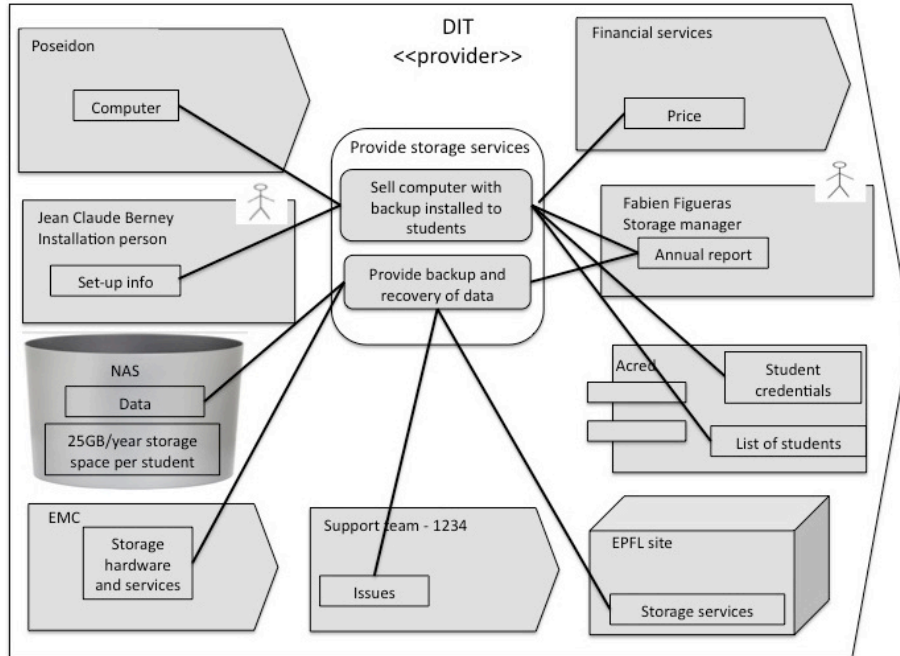


CUSTOMER-FOCUSED

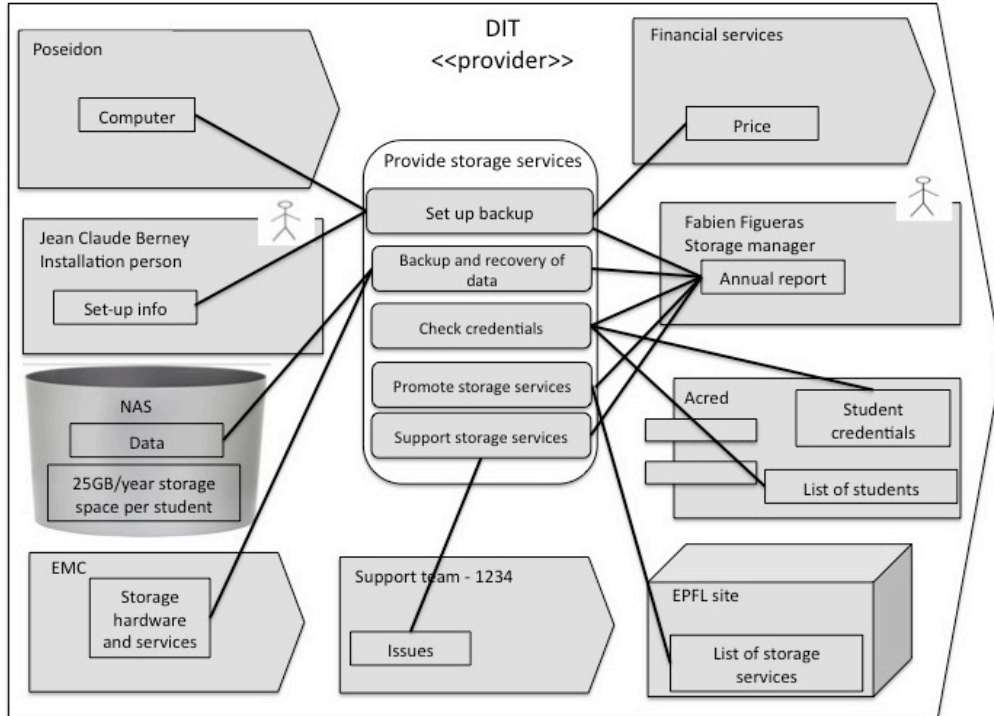
Always start by analysis of service provider, customer and possibly other external stakeholders, such as regulator. The focus of modeling the case is on providing a value to the customer.

INTERNAL ORGANIZATION

Step 3: Add internal organization and collaboration that implements external services together



Step 4: Add internal collaborations necessary to implement external services

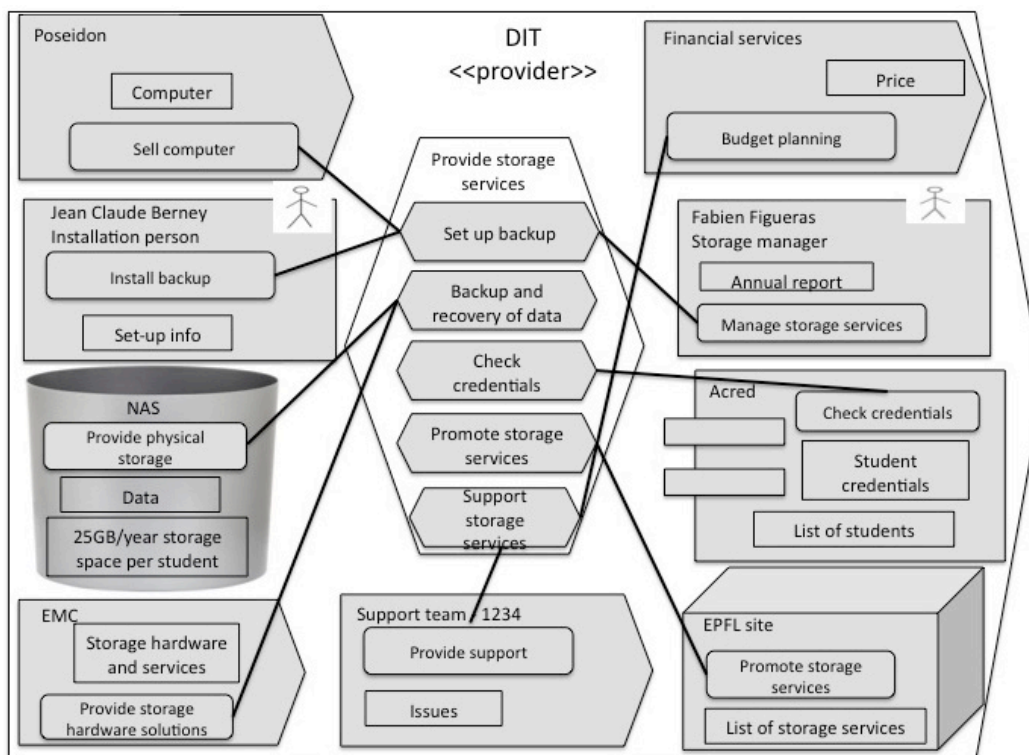


SERVICE-BASED INTERNAL ORGANIZATION

People from different departments can be members of the same team. Organizations/people from outside the company, such as suppliers, can be modeled as a part of the company providing the service. Always keep in mind delivery, support and promotion of the service.

RESPONSIBILITIES

Step 5: Define internal services of each internal organization



Each internal organization can now apply the same steps to identify necessary resources and define responsibilities among teams in order to deliver the required services.

RESPONSIBILITIES

In addition to analyzing customer needs, we model the internal organization and their responsibilities in providing the services and providing the value to the customer.

GENERAL CHARACTERISTICS

All teams and people are aware of their contribution to the final service and customer's satisfaction.

Provides an overview of the entire process.

Puts everyone involved in service design on the same page.

Provides common platform for customers, managers, and employees.

Provides cross-functional and cross-organizational view of services.

Simulation Demonstration: Alloy instances, BASS instances, Java prototype

Questionnaire

Please, answer the following questions.

First name:

Family name:

Name of the organization where you work:

Position:

Name of organization(s) in which you have worked on projects related to business and/or IT:

1. Do you use some visual modeling methods to represent business cases? If yes, which?
2. Do you believe that the proposed method could bring some new perspective for your projects?
3. Do you believe that the proposed method could help you facilitate the workshops with stakeholders?
4. Do you think that using one diagram type containing both data and behavior of modeled organization can be useful for some kind of projects and analysis? Please, explain your answer.
5. Do you think that showing the real names of people in the model can be useful? What do you see as advantages and disadvantages of such approach? Would you use it in some projects?
6. Do you think that modeling services and processes without showing sequence of steps (declaratively) can be useful? If yes, when would you use it? Please, explain your answer.
7. What do you think about service-based internal organization explained in step 3 of the process? Have you already used something similar in your projects? Would you use it? As a reminder, this includes:
 - Putting people from different departments in the same team based on their role in providing the service
 - Putting external organizations as a part of service provider
 - Always having in mind support, promotion and delivery of service?
8. Do you think that possibility to simulate the model is useful? Have you already used it? With which approaches/methods?
9. Do you find BASS simulation to be useful? Are the results of simulation readable?
10. What else would you like to see as a result of simulation?
11. What do you think about usefulness of BASS for practitioners? Please, explain your answer.
12. Do you think the proposed method can help you (your company)? If yes, how?
Please, explain advantages and disadvantages you can expect.

Thank you very much for your cooperation and for taking the time to participate in this study!

Glossary

System - a way of looking at the world (G. Weinberg). System can be of any nature (IT, human, company, and others).

Service - the application of resources (including competences, skills, and knowledge) to make changes that have value for another (system) (J. Sphorer). It consists of service offering and service implementation.

Service offerings – definition of what service is provided to the customer, without going into details of implementation. They are the starting point for configuring Service Portfolio Management. It is expressed with precondition and the effect. The precondition is expressed using properties and arithmetic and logical operations. The effect is expressed using functional units and properties.

Collaboration – one way of service implementation, which defines how component's service systems implement the main service together, showing only the net effect of their interaction.

Process – one way of service implementation, which defines the responsibilities of each of component's service systems in implementing the main service.

Service system - a configuration of people, technologies, and other resources that interact with other service systems in order to create mutual value. Many systems can be viewed as service systems, including families, cities, and companies, among many others. It can be represented as a whole showing the service offering, or as a composite showing the service implementation with a collaboration or a process.

Functional unit – atomic unit of logic used to describe the effect of a service offering or collaboration.

Property – data defined in a certain context used to parameterize functional units. The context of the property determines the visibility and how long the data is available, i.e. its lifecycle.

Precondition – a logical assertion, which when met, guarantees that the postcondition will hold if the command (effect) is executed.

Invariant – a logical assertion that is held to always be true in a certain context (during the execution of a service, inside IT system in general, etc.). From business perspective, invariants can be used to model (business) requirements of an enterprise.

Business rule - a rule that defines or constrains some aspect of business and always resolves to either true or false.

Service specification – a description of a service offering or service implementation within the service system. It is shown with a visual model of the service on any level of abstraction.

Simulation – generation of instances of the service system that satisfies the constraints of the model. They correspond to the state of the system before and after service execution. This is achieved using translation to Alloy language and simulation with the Alloy Analyzer tool.

Prototyping - generation of the prototype of Java application in which the user can test the behavior of the service system by entering input values and observing outputs. This is achieved using Arcimboldo framework.

Validation - checking that a service or system meets the needs of the customer or other stakeholders. It includes acceptance with customers and stakeholders. We use both Alloy simulation and Java prototyping to validate the models with stakeholders.

Verification - checking that a service or system conforms to the specification, regulation or imposed condition. It is often an internal process and does not include stakeholders. We use simulation to verify that the model conforms to the meta-model, well-formedness rules and other constraints.

Bibliography

- [1] P. A. Wegmann, "On the Systemic Enterprise Architecture Methodology (SEAM," in *SEAM*). Published at the International Conference on Enterprise Information Systems 2003 (ICEIS 2003, 2003, pp. 483–490.
- [2] G. M. Weinberg, *An Introduction to general systems thinking*. Wiley New York, 1975.
- [3] J. Spohrer, S. L. Vargo, N. Caswell, and P. P. Maglio, "The Service System Is the Basic Abstraction of Service Science," in *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 2008, pp. 104–104.
- [4] I. Rychkova, G. Regev, and A. Wegmann, "Declarative Specification and Alignment Verification of Services in ITIL," in *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, 2008, pp. 127–134.
- [5] A. B. Saxena, "A Situated and Embodied Approach to Service-oriented Modeling," EPFL, 2014.
- [6] A. Golnam, "Problem Structuring with the Systemic Enterprise Architecture Method," 2013.
- [7] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004.
- [8] "OMG: Unified Modeling Language - Version 2.5, OMG Document Number: ptc/2013-09-05." 2013.
- [9] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, 2008.
- [10] "OMG: Business Process Model and Notation -Version 2.0.2, OMG Document Number: formal/2013-12-09." 2014.
- [11] B. Bajić-Bizumić, C. Petitpierre, H. C. Huynh, and A. Wegmann, "A Model-Driven Environment for Service Design, Simulation and Prototyping," in *Exploring Services Science*, Springer, 2013, pp. 200–214.
- [12] B. Bajic-Bizumic, I. Rychkova, and A. Wegmann, "The Role of Invariants in the Co-evolution of Business and Technical Service Specification of an Enterprise," *POEM Latv.*, pp. 183–192, 2013.
- [13] D. Jackson, I. Schechter, and I. Shlyakhter, "ALCOA: The Alloy constraint analyzer," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 730–733.
- [14] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- [15] D. D'Souza and A. Wills, *Objects, Components and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 2001.
- [16] "OMG: Unified Modeling Language - Version 2.4.1 Superstructure, OMG Document Number: formal/2011-08-06." 2011.
- [17] "OMG: Systems Modeling Language (SysML) -Version 1.3, OMG Document Number: formal/2012-06-01." 2012.
- [18] Y. Vanderperren and W. Dehaene, "From UML/SysML to Matlab/Simulink: Current State and Future Perspectives," in *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, 3001 Leuven, Belgium, Belgium, 2006, pp. 93–93.
- [19] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and Simulation Modeling Using SysML," in *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come*, Piscataway, NJ, USA, 2007, pp. 796–803.

- [20] “The Open Group: ArchiMate 2.1 Specification.” Van Haren Publishing, 2014.
- [21] M. M. Lankhorst, H. A. Proper, and H. Jonkers, “The Architecture of the ArchiMate Language,” in *Enterprise, Business-Process and Information Systems Modeling*, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and R. Ukor, Eds. Springer Berlin Heidelberg, 2009, pp. 367–380.
- [22] J. A. Zachman, “A framework for information systems architecture,” *IBM Syst. J.*, vol. 26, no. 3, pp. 276–292, 1987.
- [23] J. F. Sowa and J. A. Zachman, “Extending and formalizing the framework for information systems architecture,” *IBM Syst. J.*, vol. 31, no. 3, pp. 590–616, 1992.
- [24] J. P. Zachman, “The Zachman framework evolution,” *EA Artic. Page Intentionally Left Blank*, 2009.
- [25] A. Wegmann, A. Kotsalainen, L. Matthey, G. Regev, and A. Giannattasio, “Augmenting the Zachman Enterprise Architecture Framework with a Systemic Conceptualization,” in *12th International IEEE Enterprise Distributed Object Computing Conference, 2008. EDOC '08*, 2008, pp. 3–13.
- [26] *Object-Process Methodology: A Holistic Systems Paradigm ; with CD-ROM*. Springer Science & Business Media, 2002.
- [27] D. Dori, I. Reinhartz-Berger, and A. Sturm, “Developing Complex Systems with Object-Process Methodology Using OPCAT,” in *Conceptual Modeling - ER 2003*, I.-Y. Song, S. W. Liddle, T.-W. Ling, and P. Scheuermann, Eds. Springer Berlin Heidelberg, 2003, pp. 570–572.
- [28] W. M. P. van der Aalst, M. Weske, and D. Grünbauer, “Case handling: a new paradigm for business process support,” *Data Knowl. Eng.*, vol. 53, no. 2, pp. 129–162, May 2005.
- [29] P. Athena, “Case handling with flower: Beyond workflow,” *Pallas Athena BV Apeldoorn Neth.*, vol. 11, 2002.
- [30] M. Pesic and W. M. P. van der Aalst, “A Declarative Approach for Flexible Business Processes Management,” in *Business Process Management Workshops*, J. Eder and S. Dustdar, Eds. Springer Berlin Heidelberg, 2006, pp. 169–180.
- [31] W. M. van der Aalst and M. Pesic, “Specifying, discovering, and monitoring service flows: Making web services process-aware,” *BPM Cent. Rep. BPM-06-09 BPMcenter Org*, 2006.
- [32] “OMG: Case Management Model and Notation (CMMN) -Version 1.0, OMG Document Number: formal/2014-05-05.” 2014.
- [33] J. L. G. Dietz, “Understanding and Modeling Business Processes with DEMO,” in *Conceptual Modeling — ER '99*, J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais, Eds. Springer Berlin Heidelberg, 1999, pp. 188–202.
- [34] G. L. Shostack, “Designing Services That Deliver,” *Harv. Bus. Rev.*, 1984.
- [35] M. J. Bitner, A. L. Ostrom, and F. N. Morgan, “Service blueprinting: a practical technique for service innovation,” *Calif. Manage. Rev.*, vol. 50, no. 3, p. 66, 2008.
- [36] L. Patricio, R. P. Fisk, J. F. e Cunha, and L. Constantine, “Multilevel Service Design: From Customer Value Constellation to Service Experience Blueprinting,” *J. Serv. Res.*, p. 1094670511401901, Mar. 2011.
- [37] J. Gordijn, Yu, Eric, and B. van der Raadt, “e-Service Design Using i* and e3value Modeling,” *IEEE Softw.*, vol. 23, no. 3, pp. 26–33, 2006.
- [38] J. Gordijn, “E3value in a Nutshell,” in *International workshop on e-business modeling, HEC Business School, Lausanne*, 2002.
- [39] E. S. K. Yu, “Towards modeling and reasoning support for early-phase requirements engineering,” in *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, 1997, 1997, pp. 226–235.

- [40] C. Rolland, S. Nurcan, and G. Grosz, "Enterprise knowledge development: the process view," *Inf. Manage.*, vol. 36, no. 3, pp. 165–184, Sep. 1999.
- [41] J. Montilva C and J. Barrios A, "BMM: A Business Modeling Method For Information Systems Development," *CLEI Electron. J.*, vol. 7, 2004.
- [42] R. W. Butler, "What is Formal Methods?" 2014.
- [43] "Alloy site - <http://alloy.mit.edu/alloy/>." 2014.
- [44] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2012.
- [45] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Computer Aided Verification*, E. Brinksma and K. G. Larsen, Eds. Springer Berlin Heidelberg, 2002, pp. 359–364.
- [46] L. C. Paulson, *Isabelle: A Generic Theorem Prover*. Springer Science & Business Media, 1994.
- [47] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer Science & Business Media, 2002.
- [48] J. Yoo, E. Jee, and S. Cha, "Formal Modeling and Verification of Safety-Critical Software," *IEEE Softw.*, vol. 26, no. 3, pp. 42–49, May 2009.
- [49] J. Kuchar and A. C. Drumm, "The traffic alert and collision avoidance system," *Linc. Lab. J.*, vol. 16, no. 2, p. 277, 2007.
- [50] T. Lecomte, T. Servat, and G. Pouzancre, "Formal methods in safety-critical railway systems," in *Proc. Brazilian Symposium on Formal Methods: SMBF*, 2007.
- [51] V. George and R. Vaughn, "Application of lightweight formal methods in requirement engineering," *CrossTALK- J. Def. Softw. Eng. Jan 2003*, 2003.
- [52] D. Jackson, "Lightweight Formal Methods," in *FME 2001: Formal Methods for Increasing Software Productivity*, J. N. Oliveira and P. Zave, Eds. Springer Berlin Heidelberg, 2001, pp. 1–1.
- [53] S. Agerholm and P. G. Larsen, "A Lightweight Approach to Formal Methods," in *Applied Formal Methods — FM-Trends 98*, D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, Eds. Springer Berlin Heidelberg, 1999, pp. 168–183.
- [54] S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, "Experiences using lightweight formal methods for requirements modeling," *IEEE Trans. Softw. Eng.*, vol. 24, no. 1, pp. 4–14, Jan. 1998.
- [55] F. M. S. Iii and C. C. Marshall, "Formality Considered Harmful: Experiences, Emerging Themes, and Directions on the Use of Formal Representations in Interactive Systems," *Comput. Support. Coop. Work CSCW*, vol. 8, no. 4, pp. 333–352, Dec. 1999.
- [56] "OMG: Object Constraint Language-Version 2.2, OMG Document Number: formal/2010-02-01." 2010.
- [57] "OMG: Meta Object Facility (MOF) -Version 2.4.1, OMG Document Number: formal/2013-06-01." 2013.
- [58] M. Vaziri and D. Jackson, "Some Shortcomings of OCL, the Object Constraint Language of UML (Response to Object Management Group's Request for Information on UML 2.0)." MIT, 1999.
- [59] C. Snook and M. Butler, "UML-B: Formal Modeling and Design Aided by UML," *ACM Trans Softw Eng Methodol*, vol. 15, no. 1, pp. 92–122, Jan. 2006.
- [60] J. R. Abrial and J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.
- [61] A. Evans, R. France, K. Lano, and B. Rumpe, "The UML as a Formal Modeling Notation," in *The Unified Modeling Language. «UML» '98: Beyond the Notation*, J. Bézivin and P.-A. Muller, Eds. Springer Berlin Heidelberg, 1999, pp. 336–348.

- [62] K. Lano, D. Clark, and K. Androutsopoulos, "UML to B: Formal Verification of Object-Oriented Models," in *Integrated Formal Methods*, E. A. Boiten, J. Derrick, and G. Smith, Eds. Springer Berlin Heidelberg, 2004, pp. 187–206.
- [63] P. Facon, R. Laleau, and H. P. Nguyen, "Mapping Object Diagrams into B Specifications," in *Proceedings of the 1996 International Conference on Methods Integration*, Swinton, UK, UK, 1996, pp. 6–6.
- [64] P. Facon, R. Laleau, H. P. Nguyen, and A. Mammar, "Combining UML with the B formal method for the specification of database applications," CEDRIC Laboratory, CNAM, Paris, 1999.
- [65] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," in *Model Driven Engineering Languages and Systems*, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds. Springer Berlin Heidelberg, 2007, pp. 436–450.
- [66] B. Beckert, R. Hähnle, and P. H. Schmitt, *Verification of Object-oriented Software: The KeY Approach*. Berlin, Heidelberg: Springer-Verlag, 2007.
- [67] A. D. Brucker and B. Wolff, "HOL-OCL: A Formal Proof Environment for uml/ocl," in *Fundamental Approaches to Software Engineering*, J. L. Fiadeiro and P. Inverardi, Eds. Springer Berlin Heidelberg, 2008, pp. 97–100.
- [68] R. Laleau, F. Semmak, A. Matoussi, D. Petit, A. Hammad, and B. Tatibouet, "A first attempt to combine SysML requirements diagrams and B," *Innov. Syst. Softw. Eng.*, vol. 6, no. 1–2, pp. 47–54, Mar. 2010.
- [69] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso, "Model checking early requirements specifications in Tropos," in *Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings*, 2001, pp. 174–181.
- [70] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Sci. Comput. Program.*, vol. 20, no. 1–2, pp. 3–50, Apr. 1993.
- [71] K. M. Cooper, "Stimulus response requirements specification notation: an empirically evaluated requirements specification notation," University of British Columbia, 2001.
- [72] P. Ogilvie, "Formal Methods in Requirements Engineering." 2013.
- [73] R. Yates, J. Andrews, and P. Gray, "Practical experience applying formal methods to air traffic management software," in *Proceedings of the 8th Annual International Symposium of the International Council on Systems Engineering*, 1998.
- [74] J. P. Bowen, "Z: A Formal Specification Notation," in *Software Specification Methods*, M. Frappier and H. Habrias, Eds. Springer London, 2001, pp. 3–19.
- [75] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-driven Requirements Elaboration," in *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, New York, NY, USA, 1996, pp. 179–190.
- [76] M. Richters, "A precise approach to validating UML models and OCL constraints," Universitat Bremen, 2002.
- [77] D. Zignale, S. Kubicki, S. Ramel, and G. Halin, "A Model-Based Method for the Design of Services in Collaborative Business Environments," in *Exploring Services Science*, M. Snene, J. Ralyté, and J.-H. Morin, Eds. Springer Berlin Heidelberg, 2011, pp. 68–82.
- [78] A. Mammar, S. Ramel, B. Grégoire, M. Schmitt, and N. Guelfi, "Efficient: A Toolset for Building Trusted B2B Transactions," in *Advanced Information Systems Engineering*, O. Pastor and J. F. e Cunha, Eds. Springer Berlin Heidelberg, 2005, pp. 430–445.
- [79] Y. Zhao, Y. Zong-yuan, and J. Xie, "Pi-calculus Based Assembly Mechanism of UML State Diagram and Validation of Model Refinement," in *2009 International Conference on Electronic Computer Technology*, 2009, pp. 604–609.

- [80] P. O. Ating'a and A. Krishna, "Verification of i* Models Using Alloy," in *Information Systems Development*, J. Pokorny, V. Repa, K. Richta, W. Wojtkowski, H. Linger, C. Barry, and M. Lang, Eds. Springer New York, 2011, pp. 63–74.
- [81] M. C. Reynolds, "Lightweight Modeling of Java Virtual Machine Security Constraints," in *Abstract State Machines, Alloy, B and Z*, M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, Eds. Springer Berlin Heidelberg, 2010, pp. 146–159.
- [82] S. Sen, B. Baudry, and H. Vangheluwe, "Towards Domain-specific Model Editors with Automatic Model Completion," *SIMULATION*, Oct. 2009.
- [83] A. Wegmann, L.-S. Le, I. Rychkova, and G. Regev, "An Example of a Hierarchical System Model Using SEAM and its Formalization in Alloy," in *Eleventh International IEEE EDOC Conference Workshop, 2007. EDOC '07*, 2007, pp. 260–268.
- [84] H. Bagheri and K. Sullivan, "Monarch: Model-Based Development of Software Architectures," in *Model Driven Engineering Languages and Systems*, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Springer Berlin Heidelberg, 2010, pp. 376–390.
- [85] A. Kattepur, S. Sen, B. Baudry, A. Benveniste, and C. Jard, "Pairwise Testing of Dynamic Composite Services," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA, 2011, pp. 138–147.
- [86] A. Kattepur, S. Sen, B. Baudry, A. Benveniste, and C. Jard, "Variability Modeling and QoS Analysis of Web Services Orchestrations," in *2010 IEEE International Conference on Web Services (ICWS)*, 2010, pp. 99–106.
- [87] B. H. Banathy and P. M. Jenlink, "Systems inquiry and its application in education," *Handb. Res. Educ. Commun. Technol.*, pp. 37–58, 2003.
- [88] "Merriam-Webster - <http://www.merriam-webster.com/>." 2014.
- [89] H. Mintzberg, B. Ahlstrand, and J. Lampel, "Strategy Safari – The complete guide through the wilds of strategic management.," 1998.
- [90] P. Checkland and S. Holwell, "Information, Systems and Information Systems: Making Sense of the Field," 1998.
- [91] G. Vickers, *Value Systems and Social Process*. Routledge, 2013.
- [92] G. Regev, O. Hayard, and A. Wegmann, "Service Systems and Value Modeling from an Appreciative System Perspective," in *Exploring Services Science*, M. Snene, J. Ralyté, and J.-H. Morin, Eds. Springer Berlin Heidelberg, 2011, pp. 146–157.
- [93] S. L. Vargo, P. P. Maglio, and M. A. Akaka, "On value and value co-creation: A service systems and service logic perspective," *Eur. Manag. J.*, vol. 26, no. 3, pp. 145–152, Jun. 2008.
- [94] H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*. Elsevier, 1997.
- [95] A. Wegmann, P. Julia, G. Regev, O. Perroud, and I. Rychkova, "Early Requirements and Business-IT Alignment with SEAM for Business," in *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, 2007, pp. 111–114.
- [96] A. Wegmann, P. Balabko, L.-S. Le, G. Regev, and I. Rychkova, "A Method and Tool for Business-IT Alignment in Enterprise Architecture," in *Proceedings of the CAiSE Forum*, vol. 5, Porto, Portugal, 2005, pp. 113–118.
- [97] A. Wegmann, G. Regev, I. Rychkova, L.-S. Le, J. de la Cruz, and P. Julia, "Business and IT alignment with SEAM for enterprise architecture," *Fac. Inform. - Pap. Arch.*, Jan. 2007.
- [98] A. Golnam, G. Regev, J. Ramboz, P. Laprade, and A. Wegmann, "Systemic Service Design: Aligning Value and Implementation," in *Exploring Services Science*, Springer, 2010, pp. 150–164.

- [99] A. Wegmann, G. Regev, D. L. Cruz, J. Diego, L.-S. Lê, and I. Rychkova, "Teaching Enterprise and Service-Oriented Architecture in Practice," *J. Enterp. Archit.*, vol. 4, no. 3, pp. 15 – 24, 2007.
- [100] G. Regev, D. C. Gause, and A. Wegmann, "Experiential learning approach for requirements engineering education," *Requir. Eng.*, vol. 14, no. 4, pp. 269–287, Dec. 2009.
- [101] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 2004.
- [102] A. Wegmann, B. Bajic-Bizumic, A. Golnam, G. Popescu, G. Tapandjieva, A. B. Saxena, M. Yassaee, and G. Regev, "Requirements modeling in SEAM: The example of a car crash management system," in *Comparing Requirements Modeling Approaches Workshop (CMA@RE), 2013 International*, 2013, pp. 25–30.
- [103] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.
- [104] J. den Haan, "Model Transformation - <http://www.theenterprisearchitect.eu/blog/2008/02/18/mda-and-model-transformation/>." 2014.
- [105] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, Mar. 2006.
- [106] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications," *ACM Trans Program Lang Syst*, vol. 8, no. 2, pp. 244–263, Apr. 1986.
- [107] M. J. C. Gordon and T. F. Melham, Eds., *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. New York, NY, USA: Cambridge University Press, 1993.
- [108] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Progress on the State Explosion Problem in Model Checking," in *Informatics*, R. Wilhelm, Ed. Springer Berlin Heidelberg, 2001, pp. 176–194.
- [109] I. Rychkova, "Formal semantics for refinement verification of enterprise models," EPFL, 2008.
- [110] J. B. Warmer and A. G. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Professional, 2003.
- [111] G. Dennis and R. Seater, "Alloy Analyzer 4 Tutorial. Session 2: Language and Analysis." 2014.
- [112] "Arcimboldo site - <http://ltiwww.epfl.ch/Arcimboldo/>." 2014.
- [113] C. Petitpierre, "Bottom up creation of a DSL using templates and JSON," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE'11, AOOPEs'11, NEAT'11, & VMIL'11*, 2011, pp. 47–52.
- [114] "JSON Template - <http://json-template.googlecode.com/svn/trunk/doc/Introducing-JSON-Template.html>." 2014.
- [115] "Information Technology Infrastructure Library (ITIL) - <http://www.itil-officialsite.com/>." 2014.
- [116] H. Kilov and I. Simmonds, "Business rules: from business specification to design," in *Object-Oriented Technologys*, J. Bosch and S. Mitchell, Eds. Springer Berlin Heidelberg, 1998, pp. 188–194.
- [117] G. Regev, I. Bider, and A. Wegmann, "Defining business process flexibility with the help of invariants," *Softw. Process Improv. Pract.*, vol. 12, no. 1, pp. 65–79, Jan. 2007.
- [118] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *IEEE Trans. Softw. Eng.*, vol. 26, no. 10, pp. 978–1005, Oct. 2000.

- [119] B. Bajic, C. Petitpierre, D. Quang Tri, and A. Wegmann, "From Business Services to IT Services by Capturing Design Decisions," *BMSD Geneva*, pp. 94–104, 2012.
- [120] "Generale Ressorts site - <http://www.generaleressorts.com/>." 2014.
- [121] "Order-to-cash cycle - <http://www.three2tango.com/general/business/order-to-cashotc.html/>." 2014.
- [122] B. Bajic-Bizumic, I. Rychkova, and A. Wegmann, "Simulation-Driven Approach for Business Rule Discovery," in *CAiSE International Workshops, Advanced Information Systems Engineering Workshops*, Springer, 2013, pp. 111–123.
- [123] H. C. Huynh, B. Bajic-Bizumic, and A. Wegmann, "A SEAM-based Environment for System Design and Validation," *ACOMP Ho Chi Minh City*, 2013.

Biljana Bajić-Bizumić

Rütistrasse 12

8072 Zollikon

Date of birth: 27th December 1985

Phone: +41 79 8334414

e-mail: biljana.bajic@epfl.ch**Synthesis**

- EPFL Phd in Computer Science (in progress)
- Professional experience in international companies
- Board member and vice-president of EESTEC LC Novi Sad (2006-2008)
- State champion of Serbia in Maths (2000)
- ITIL V3 Foundation Certificate

Education

- 2010 – current: PhD in Computer Science, Ecole Polytechnique Fédérale de Lausanne (EPFL)
Research topic: Animation-Based Service Specification, Validation and Verification
- 2004-2009: Master in Computer Science, Faculty of Technical Sciences, University of Novi Sad (GPA 9,9/10)
- 2000-2004: Gymnasium “J. J. Zmaj”, Novi Sad, Class for talented mathematicians (GPA 5/5)

Professional experience**January-June 2013: Business Analyst at EPFL, Lausanne, Switzerland**

- Specifying a new solution for computer data storage services at EPFL.
- Organizing a workshop with stakeholders.

2010 – now: Doctoral Assistant at EPFL, Lausanne, Switzerland

- Research in the field of business/IT alignment: developing a method that enables business and IT analysts to create more precise models. The method includes visual formalism for modeling business cases and simulation tools for validation of the models with stakeholders.
- Presenting at various conferences.
- Teaching the course “Enterprise Service Oriented Architecture (ESOA)”, where students learn the basics of business, business/IT, and IT issues and the relations between these three areas through running their own company.
- Teaching engineering courses, such as “Information Technology Project (ITP)” including Java development and “Real-time Systems”.
- Supervising student semester projects and bachelor and master thesis in a field of business/IT alignment, service design, business process modeling, and others.

2008-2010: Software Engineer/Team Leader at Schneider Electric DMS NS, Novi Sad, Serbia

- Developer and leader at Schneider Electric DMS NS, an IT company for research, development and engineering in the field of the electrical power engineering management working on projects with Siemens and Telvent.
- Developing a tool in C++ and C# for the project *Electrical Power System Modeling in ArcGIS and Integration with DMS Object Model*.
- Team leading, designing, developing and deploying the web based *Release Management Tool* in Java, JSF and MySQL for managing software releases inside the company.

September-December 2008: Internship at SAP, Vienna, Austria

- Developing the *Global Shift Planning Tool* in ASP.NET, C# and SQL Server, working with SCRUM methodology.

Extracurricular activities

- 2004-2008: **Junior Research Assistant at Petnica Research Center, Petnica, Serbia**
- Organizing seminars, teaching talented high school students advanced mathematics and supervising their projects in the oldest and largest independent nonprofit organization for extracurricular informal science education in South Eastern Europe.
- 2006-2008: **Board Member and Vice-President of EESTEC (Electrical Engineering Students' European association), Novi Sad, Serbia**
- Organizing student workshops and job fairs, fundraising and managing relationships with companies.
 - Participating in international student workshops.
 - Initiating and organizing the first job fair "KONTEH" at the Faculty of Technical Sciences in December 2006. Currently, it is the job fair that collects the greatest amount of CVs in the region and provides jobs and internships to hundreds of students.

Languages

Serbian: Native language, **English:** Fluent, **German:** Intermediate (B2), **French:** Intermediate (B1/B2).

Awards and Certifications

- 2013: ITIL V3 Foundation Certificate
- 2004-2013: Scholarship of Ministry of Science for talented students of Serbia
- 2009: 100 Best Students and Young Leaders of Serbia award by EFG Eurobank
- 2008: 40 Serbian Young Professionals award by the Foundation Zoran Djindjic and WUS Austria
- 2006-2007: Top 10 students of University of Novi Sad scholarship
- 2006: DAAD (Deutscher Akademischer Austausch Dienst) scholarship for summer school of German in München, Germany
- 2005: Representative of Serbia in international summer science camp, XLAB, Göttingen, Germany
- 2003-2008: Fellow of Schneider Electric DMS NS company
- 1998-2004: Winning awards on mathematical competitions, from the first place in the city to gold medal at the national competition of Serbia

Interests: Traveling, swimming (silver and bronze medals at regional competitions), biking, playing the piano

Publications

B. Bajić-Bizumić, C. Petitpierre, H. C. Huynh, and A. Wegmann, “A Model-Driven Environment for Service Design, Simulation and Prototyping,” in *Exploring Services Science*, Springer, 2013, pp. 200–214.

B. Bajic-Bizumic, I. Rychkova, and A. Wegmann, “The Role of Invariants in the Co-evolution of Business and Technical Service Specification of an Enterprise,” *POEM 2013*

B. Bajic-Bizumic, I. Rychkova, and A. Wegmann, “Simulation-Driven Approach for Business Rule Discovery,” in *CAiSE International Workshops, Advanced Information Systems Engineering Workshops*, Springer, 2013, pp. 111–123.

A. Wegmann, B. Bajic-Bizumic, A. Golnam, G. Popescu, G. Tapandjieva, A. B. Saxena, M. Yassae, and G. Regev, “Requirements modeling in SEAM: The example of a car crash management system,” in *Comparing Requirements Modeling Approaches Workshop (CMA@RE), 2013 International*, 2013, pp. 25–30.

G. Regev, B. Bajic-Bizumic, A. Golnam, G. Popescu, G. Tapandjieva, A. B. Saxena, and A. Wegmann, “A Philosophical Foundation for Business and IT Alignment in Enterprise Architecture with the Example of SEAM,” *BMSD Noordwijkerhout*, 2013.

H. C. Huynh, B. Bajic-Bizumic, and A. Wegmann, “A SEAM-based Environment for System Design and Validation,” *ACOMP Ho Chi Minh City*, 2013.

B. Bajic, C. Petitpierre, D. Quang Tri, and A. Wegmann, “From Business Services to IT Services by Capturing Design Decisions,” *BMSD Geneva*, pp. 94–104, 2012.

B. Bajic, “Electrical Power System Modeling in ArcGIS and Integration with DMS Object Model,” *FTN Zbornik radova*, 2010.

B. Bajic, V. Bojkovic, A. Ilic, and M. Kosanovic, “Banach Tarski Paradox in 4 Dimensions,” *Petnica Papers*, 2004.