



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

Plácido Antônio de Souza Neto

***A Methodology for Building Service-Oriented
Applications in the Presence of Non-Functional
Properties***

Natal-RN / Brasil

2012

Plácido Antônio de Souza Neto

***A Methodology for Building Service-Oriented
Applications in the Presence of Non-Functional
Properties***

Tese submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte (UFRN), como requisito para a obtenção do grau de Doutor em Ciência da Computação.

Advisor: Martin A. Musicante (UFRN-Brazil)

Co-Advisor: Genoveva Vargas-Solar (CNRS-France)

Natal-RN / Brasil

2012

Catálogo da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial
Especializada do Centro de Ciências Exatas e da Terra – CCET.

Souza Neto, Plácido Antônio de.

A methodology for building service-oriented applications in the presence of non-functional properties / Plácido Antônio de Souza Neto. – Natal, RN, 2012.
190 f. : il.

Orientador: Prof. Dr. Martin A. Musicante.

Co-orientadora: Profa. Dra. Genoveva Vargas-Solar

Tese (Doutorado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada. Programa de Pós-Graduação em Sistemas e Computação.

1. Engenharia de software – Tese. 2. Serviços web – Tese. 3. Desenvolvimento dirigido a modelos – Tese. 4. Metodologia – Tese. I. Musicante, Martin A. II. Vargas-Solar, Genoveva. III. Título.

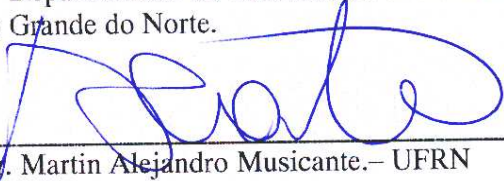
RN/UF/BSE-CCET

CDU 004.41

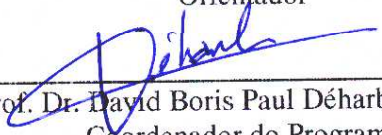
PLÁCIDO ANTONIO DE SOUZA NETO

**“A Methodology for Building Service-Oriented
Applications in the presence of Non-Functional Properties”**

Esta Tese foi julgada adequada para a obtenção do título de doutor em Ciência da Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.




Prof. Dr. Martin Alejandro Musicante.- UFRN
Orientador



Prof. Dr. David Boris Paul Déharbe – UFRN
Coordenador do Programa

Banca Examinadora




Examinador: Profª. Drª. Marcia Jacyntha Nunes Rodrigues Lucena (UFRN)



Examinador: Profª. Drª. Genevêva Vargas Solar (CNRS)



Examinador: Profª. Drª. Silvia Regina Vergílio (UFPR)



Examinador: Profª. Drª. Valeria de Castro (URJC)



Examinador: Prof. Dr. Nelson Rosa Souto (UFPE)

Dezembro, 2012

I dedicate this work to my beloved wife Ana Flávia and to my son Lucas Kauai, my daily sunshine.

ACKNOWLEDGEMENTS

When I started my Ph.D. studies back in 2008, I couldn't imagine how to write a thesis. For me it seemed to be an impossible endeavor. However, thanks to the will of God I finished the text of this thesis. So, I want to express my gratitude to God for this gift, because without him it would not be possible.

Now I have the opportunity to thank many people who helped me in some way, so, I can't forget anyone of them. I will try to do that. During my Ph.D. I had really nice and lovely people around me, and I thank for the "gifts" (mainly that ones you feel) and unforgettable moments were passed near them. I found new friends and rediscovered old ones. These things, only a God that loves you can provide.

I'm grateful to my supervisor Prof. Martin Musicante, not only for the ongoing support but also for believing in me during this time and for the opportunities he provided for my growth as a researcher. I deeply thank my co-supervisor Prof. Genoveva Vargas-Solar for the time and effort spent on my thesis and for the feedback and support when I was in Mexico and France, and even by many emails and meetings by conference when I came back to Brazil. I also would like to thank Prof. Valeria de Castro for the patience and suggestions given for my work and for the friendship during the short time we had contact. I hope to cooperate with everyone for a long time, because I know that many fruits can be harvested yet.

I also would like to thank the committee members (Prof. Nelson Souto Rosa (UFPE), Prof. Silvia Virgílio (UFPR), Prof. Márcia Lucena (UFRN) and Prof. Valeria de Casto (URJC)) who agreed to read, analyze and comment this thesis. Thank you very much for your contributions!

I was pleased to be part of the HADAS team for 9 months and meet people who I will always remember with nostalgia and affection, in particular Juan Carlos, Mohamad Othman, Epal Orleans, Carlos Manuel (Manolo), Lourdez Martinez, Christiane Kamdem, Leon Fopa and Javier Espinosa. Thanks for the friendship. I hope to have the opportunity to meet everyone again in some place of the world.

I am also grateful to Diego Caminha. A friend who I have great respect and affection and who now I consider like a brother. Thanks for had received me in France. I'm happy to have you as a friend and I miss our conversations and the good moments we spent. God bless you.

I am deeply grateful to Marinas Praia Sul Presbyterian Church (IPMPS) members for their support, prayers and love of each one. They are always by my side, I thank the church members and its amiable council. Likewise, I thank my father-in-law Luiz Machado, my mother-in-law Maria de Lourdez, my sister-in-law Ana Tereza and my brother-in-law Heliomar Dias, for being always ready to help with your prayers and actions.

My family has been my greatest supporter over the last 29 years. My mother Ana Sueli is the better and coolest mother I can imagine and I am deeply grateful for everything she did for me. I hope be like her as a father. I also thank my father Francisco Ednaldo and my sister Tatyanny Nascimento that always took care and pray for me. Thanks for being there!

Last but not least, I am very happy to have my own little family that strongly supported me in my Ph.D. studies. My beloved wife Ana Flávia always supporting me, even during the countless weekends in the office, and the 9 months I spent in Grenoble for the Ph.D. stage. I am very thankful for her love, help and support in many difficult times. My son Lucas Kauai has been the sunshine in my life for the last 2 years. I am not able to measure the love I feel for them, not only for what they have done for me, but for what they are and represent in my life. I love you two with all my heart.

Thank you so much!

Resumo

Esta tese apresenta π SOD-M (Policy-based Service Oriented Development Methodology), uma metodologia para a modelagem de aplicações orientadas a serviços a qual usa *Políticas* de qualidade. O trabalho propõe um método orientado a modelos para desenvolvimento de aplicações confiáveis. π SOD-M consiste de: (i) um conjunto de meta-modelos para representação de requisitos não-funcionais associados a serviços nos diferentes níveis de modelagem, a partir de um modelo de caso de uso até um modelo de composição de serviço, (ii) um meta-modelo de plataforma específica que representa a especificação das composições e as políticas, (iii) regras de transformação *model-to-model* e *model-to-text* para semi-automatizar a implementação de composições de serviços confiáveis, e (iv) um ambiente que implementa estes meta-modelos e regras, representando assim aspectos transversais e limitações associadas a serviços, que devem ser respeitados. Esta tese também apresenta uma classificação e nomenclatura de requisitos não-funcionais para o desenvolvimento de aplicações orientadas a serviços. Nossa abordagem visa agregar valor ao desenvolvimento de aplicações orientadas a serviços que têm necessidades de garantias de requisitos de qualidade. Este trabalho utiliza conceitos das áreas de desenvolvimento orientado a serviços, design de requisitos não-funcionais e desenvolvimento dirigido a modelos para propor uma solução que minimiza o problema de modelagem de serviços web confiáveis.

Área de Concentração: Engenharia de Software

Palavras-chave: serviços confiáveis, políticas, desenvolvimento dirigido a modelos, metodologia, π SOD-M.

Abstract

This thesis presents π SOD-M (*Policy-based Service Oriented Development Methodology*), a methodology for modeling reliable service-based applications using *policies*. It proposes a model driven method with: (i) a set of meta-models for representing non-functional constraints associated to service-based applications, starting from an use case model until a service composition model; (ii) a platform providing guidelines for expressing the composition and the policies; (iii) model-to-model and model-to-text transformation rules for semi-automatizing the implementation of reliable service-based applications; and (iv) an environment that implements these meta-models and rules, and enables the application of π SOD-M. This thesis also presents a classification and nomenclature for non-functional requirements for developing service-oriented applications. Our approach is intended to add value to the development of service-oriented applications that have quality requirements needs. This work uses concepts from the service-oriented development, non-functional requirements design and model-driven development areas to propose a solution that minimizes the problem of reliable service modeling. Some examples are developed as proof of concepts.

Area of Concentration: Software Engineering

Key words: reliable service, policy, model-driven development, methodology, π SOD-M.

Contents

List of Figures

List of Tables

1	Introduction	p. 16
1.1	Motivation and Problem Statement	p. 18
1.2	Objectives and Main Results	p. 21
1.2.1	Hypothesis	p. 21
1.2.2	Objectives	p. 22
1.2.3	Contributions	p. 23
1.3	Document Organization	p. 24
2	State of the Art	p. 26
2.1	Background	p. 28
2.1.1	Model-Driven Development	p. 28
	Computational Independent Model:	p. 28
	Platform Independent Model:	p. 29
	Computational Specific Model:	p. 29
2.1.2	Service-Oriented Applications	p. 29
2.1.3	Non-Functional Requirements	p. 30
2.1.4	Design by Contract	p. 31
2.2	Non-Functional Requirements for Service-Based Applications	p. 32
2.2.1	Concepts and Works	p. 34

2.2.2	Analysis	p. 35
2.3	Methodologies for Service Oriented Development	p. 40
2.3.1	Concepts and Works	p. 41
2.3.2	Analysis	p. 45
2.4	Classification of Non-Functional Requirements for Service-Based Applications	p. 48
2.4.1	NFR Meta-Model	p. 48
2.4.2	NFR Classification	p. 49
2.5	Conclusions	p. 53
3	πSOD-M: A Methodology for Building Reliable Service Based Applications	p. 56
3.1	π SOD-M	p. 57
3.1.1	General Overview	p. 58
3.1.2	Development Process	p. 59
3.1.3	Methodology Concepts	p. 60
3.1.4	Case Study	p. 64
3.2	Platform Independent Models	p. 66
3.2.1	π -UseCase Model	p. 66
3.2.1.1	π -UseCase Diagram, Terms and Concepts	p. 68
3.2.1.2	Meta-model	p. 70
3.2.1.3	UML Concepts Representation	p. 71
3.2.1.4	<i>To Publish Music</i> Use Case	p. 75
3.2.2	π -ServiceProcess Model	p. 76
3.2.2.1	π -ServiceProcess Diagram, Terms and Concepts	p. 77
3.2.2.2	Meta-model	p. 79
3.2.2.3	UML Concepts Representation	p. 80
3.2.2.4	<i>To Publish Music</i> Process	p. 83
3.2.3	π -ServiceComposition Model	p. 84

3.2.3.1	<i>π-ServiceComposition</i> Diagram, Terms and Concepts . . .	p. 85
3.2.3.2	Meta-model	p. 86
3.2.3.3	UML Concepts Representation	p. 88
3.2.3.4	<i>Publish Music</i> Service Composition	p. 90
3.3	π -PEWS Platform Specific Models	p. 91
3.3.1	π -PEWS Specification, Terms and Concepts	p. 92
3.3.2	Meta-model	p. 93
3.4	Model Transformations	p. 96
3.4.1	From π -UseCase to π -ServiceProcess	p. 97
3.4.2	From π -ServiceProcess to π -ServiceComposition	p. 102
3.4.3	From π -ServiceComposition to π -PEWS	p. 105
3.5	Conclusions	p. 107
4	πSOD-M Environment	p. 109
4.1	General Architecture	p. 110
4.1.1	Ecore Meta-models (<i>Models Plugin Module</i>)	p. 111
4.1.2	Model Transformation (<i>Mapping Plugin Module</i>)	p. 112
4.1.2.1	π -UseCase2 π ServiceProcess Transformation Rules	p. 114
4.1.2.2	π -ServiceProcess2 π -ServiceComposition Transformation Rules	p. 116
4.1.2.3	π -ServiceComposition2 π -PEWS Transformation Rules	p. 117
4.1.3	Code Generation (<i>Code Generation Module</i>)	p. 117
4.2	Defining Reliable Service Based Applications	p. 122
4.2.1	π -UseCase Model	p. 122
4.2.2	π -ServiceProcess Models	p. 124
4.2.3	π -ServiceComposition Models	p. 126
4.2.4	π -PEWS Models	p. 129

4.3	Extending the Environment	p. 130
4.4	Conclusion	p. 131
5	Evaluation	p. 132
5.1	Example 1: To Publish Music	p. 133
5.1.1	π -UseCase Model	p. 133
5.1.2	π -ServiceProcess Model	p. 134
5.1.3	π -ServiceComposition Model	p. 137
5.2	Example 2: Crime Map	p. 138
5.2.1	π -UseCase Model	p. 139
5.2.2	π -ServiceProcess Model	p. 142
5.2.3	π -ServiceComposition Model	p. 145
5.3	Example 3: GesIMED Application	p. 149
5.3.1	π -UseCase Model	p. 152
5.3.2	π -ServiceProcess Model	p. 153
5.3.3	π -ServiceComposition Model	p. 154
5.4	Lessons Learned	p. 155
5.4.1	Example 1: To Publish Music	p. 155
5.4.2	Example 2: Crime Map	p. 155
5.4.3	Example 3: GesIMED	p. 156
5.5	Measurement and Evaluation of the π SOD-M Methodology	p. 158
	Generate a set of goals:	p. 159
	Set of questions of interest which quantify the goals:	p. 160
	Set of data metrics and distributions:	p. 160
	Collecting the data:	p. 160
	Evaluation of the data:	p. 161
	Conclusions:	p. 162

5.5.1	Comparison of the SOD-M and π SOD-M approaches	p. 162
5.6	Conclusions	p. 165
6	Conclusions	p. 167
6.1	Main Contributions	p. 168
6.2	Future Work	p. 169
	References	p. 171
	Appendix A - Source Selection and Analysis Method: Non-Functional Requirements for Service-Based Applications	p. 180
	Appendix B - Service-Based Non-Functional Requirement Concepts	p. 183
	Appendix C - π-PEWS Language	p. 185
	Appendix D - <i>To Publish Music</i> Case Study Diagrams	p. 188

List of Figures

1	Service-Based Non-Functional Requirement Model.	p. 49
2	Relationship of the NFR Concepts.	p. 53
3	SOD-M and π SOD-M Extension Models.	p. 58
4	π SOD-M Development Process and Models.	p. 60
5	π SOD-M Concepts.	p. 61
6	Use Case Model “ <i>To Publish Music</i> ” - Spotify Music Service.	p. 65
7	Service Process Model “ <i>To Publish Music</i> ”.	p. 66
8	Traditional UML Concepts.	p. 67
9	π -UseCase model Example.	p. 68
10	Constraint Representation.	p. 68
11	Non-functional Requirement Representation.	p. 69
12	Non-functional Attribute Representation.	p. 69
13	π -UseCase Concepts (Meta-model).	p. 70
14	Business Collaborator / Package Representation.	p. 71
15	π -UseCase Concepts Represented as UML Elements.	p. 72
16	π -UseCase Model Representation.	p. 73
17	π -UseCase Model Representation (2).	p. 74
18	Use Case Model With Constraint Definition - Scenario Example Refined. . .	p. 77
19	Assertion Representation.	p. 78
20	Contract Representation.	p. 78
21	Node, Activity Edge and Contract Representation.	p. 79
22	π -ServiceProcess Concepts (Meta-model).	p. 80

23	π -ServiceProcess Model Representation.	p. 81
24	π -ServiceProcess Model Representation (2).	p. 82
25	Service Process Model with Contract Definition - Complete Example.	p. 84
26	Business Collaborator Representation.	p. 86
27	Policy Representation.	p. 87
28	π -ServiceComposition Concepts (Meta-model).	p. 88
29	π -ServiceComposition Representation Models.	p. 89
30	Service Composition Model with Policy Definition - Spotify Music Service.	p. 90
31	π -PEWS Specification Representation.	p. 93
32	π -PEWS Contract Representation.	p. 94
33	π -PEWS Meta-model.	p. 95
34	Entities' Model Transformation Rules.	p. 97
35	π -UseCase2 π -ServiceProcess Model Transformation Rules.	p. 100
36	π -UseCase2 π -ServiceProcess Model Transformation Rules (2).	p. 101
37	Extended Transformation Examples.	p. 102
38	Include Transformation Example.	p. 102
39	π -ServiceProcess2 π -ServiceComposition Model Transformation Rules.	p. 104
40	π SOD-M Development Environment.	p. 110
41	Environment Components.	p. 111
42	ATL Model to Model Transformation in π SOD-M.	p. 113
43	Acceleo Model to Text Transformation in π SOD-M.	p. 113
44	ATL Configuration for π SOD-M Transformation.	p. 114
45	Acceleo Specification for π -PEWS Code Generation.	p. 118
46	Model Transformation Process.	p. 121
47	π SOD-M Eclipse Plugin Environment.	p. 122
48	π -UseCase Model Definition in π SOD-M Eclipse Plugin.	p. 123

49	π -UseCase Properties in π SOD-M Eclipse Plugin.	p. 124
50	π -ServiceProcess Model Definition in π SOD-M Eclipse Plugin.	p. 126
51	π -ServiceProcess Properties in π SOD-M Eclipse Plugin.	p. 127
52	π -ServiceComposition Model Definition in π SOD-M Eclipse Plugin.	p. 128
53	π -PEWS Model Definition in π SOD-M Eclipse Plugin.	p. 129
54	π -PEWS Model Properties in π SOD-M Eclipse Plugin.	p. 130
55	<i>To Publish Music</i> π -UseCase.	p. 134
56	<i>To Publish Music</i> π -UseCase Environment Detail.	p. 135
57	<i>To Publish Music</i> π -UseCase Environment Detail.	p. 135
58	<i>To Publish Music</i> Services.	p. 136
59	<i>To Publish Music</i> π -ServiceProcess.	p. 136
60	<i>To Publish Music</i> π -ServiceComposition Model.	p. 138
61	Crime Map π -UseCase.	p. 140
62	<i>Search crime</i> π -UseCase Detail.	p. 141
63	<i>See crime information</i> π -UseCase Detail.	p. 142
64	Crime Map Services.	p. 143
65	<i>Crime information</i> π -ServiceProcess.	p. 143
66	<i>Crime information</i> π -ServiceProcess Environment.	p. 144
67	<i>See crime statistic and share information</i> π -ServiceProcess Detail.	p. 145
68	<i>Crime statistic and share information</i> π -ServiceComposition Environment.	p. 146
69	<i>Crime information</i> π -ServiceComposition.	p. 147
70	<i>Crime information</i> π -ServiceComposition Environment.	p. 148
71	<i>Crime statistic and share information</i> π -ServiceComposition.	p. 149
72	<i>See crime statistic and share information</i> π -ServiceComposition Environ- ment Detail.	p. 150
73	π -UseCase: Perform Image Processing.	p. 152

74	π -ServiceProcess: Perform Image Processing.	p. 152
75	π -ServiceComposition: Perform Image Processing.	p. 153
76	Use Case Model [39].	p. 157
77	Extended Use Case Model [39].	p. 157
78	Image Processing - Service Process Diagram [39].	p. 157
79	Image Processing - Service Composition Diagram [39].	p. 158
80	Publications per year.	p. 182
81	π -ServiceProcess - To Publish Music.	p. 188
82	π -UseCase - To Publish Music.	p. 189
83	π -ServiceComposition - To Publish Music.	p. 190

List of Tables

1	Research question results - RQ_1, RQ_2, RQ_3, RQ_4	p. 38
2	Research question results - RQ_5, RQ_6, RQ_7	p. 39
3	Methodologies' Analysis.	p. 46
4	Non-Functional Requirements Classification.	p. 50
5	Transformation Rules: From π -UseCase to π -ServiceProcess.	p. 99
6	Transformation Rules: From π -ServiceProcess to π -ServiceComposition.	p. 103
7	Transformation Rules: From π -ServiceComposition to π -PEWS.	p. 106
8	π SOD-M's Goals and Questions	p. 160
9	π SOD-M's Metrics/Distributions	p. 161
10	Results: π SOD-M's Metrics/Distributions	p. 161
11	Comparison: SOD-M and π SOD-M	p. 163
12	Summary of the studies selected at each step.	p. 181

1 Introduction

“The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.”

Stephen Hawking

A Web service is a logically atomic component that provides operations through a standardized interface and is accessible on the Internet. The most important features of a Web service platform is the use of the Internet high-level protocols for making software components interact among each other. Web services use three major standard technologies, namely: a communication protocol (SOAP - *Simple Object Access Protocol*) [24], an interface description language for describing services (WSDL - *Web Service Definition Language* [30]) and a specification for publishing and finding services (UDDI - *Universal Description, Discovery and Integration* [59]). These technologies are organized within a reference architecture that defines how they are used for using services as components that can exchange messages for implementing specific application logics.

Furthermore, it is possible to create new services (*i.e.*, *composite services*) by composing existing ones. Service composition is defined by a process consisting of activities implemented by other services .

Some composition processes require to define more complex interactions among services. For this, it is necessary an adequate description of the interface that can describe composite services. Since WSDL descriptions specify only static aspects, some proposals specify services and its compositions by adding behavioral characteristics, such as *PEWS* [13], *BPEL4WS* [8], *XLANG* [107], *WSCI* [9] e *OWL-S* [19].

Current standards in service composition implement functional, non-functional constraints and communication aspects by combining different languages and protocols. For example, to add a transactional behaviour to a service composition requires the implementation of code us-

ing the WS-Coordination, WS-Transaction, WS-BusinessActivity and WS-AtomicTransaction standards. The selection of the adequate protocols for adding a specific non-functional constraint to a service composition (e.g., security, transactional behaviour and adaptability) is completely left to the programmer. As a consequence, the development of an application based on a service composition is a complex and a time-consuming task. This is opposed to the philosophy of service-based software engineering that aims at facilitating the integration of distributed applications.

Non-functional properties of service oriented applications has been addressed in academic works and standards and it is based on the characteristics of standards used to describe services API, for instance WSDL [30], or in the characteristics of REST architectures where methods are part of the communication protocol, and also used to express service coordination like BPEL [89].

Non-functional properties are in general not considered when a service-oriented application is modeled. Methodologies in general focus on the application logic and then rely on programmers for implementing non-functional properties either by coding them directly (e.g., handling exceptions) or by configuring the services provided by the middleware (e.g., persistence or transactional properties).

Not respecting a non-functional property expressed in a specification in the design phase, implies an exceptional behavior of the application. These exceptions are dealt according to predefined actions also expressed in the design phase, oriented to tolerate these exceptions. In consequence, a reliable service coordination is the one that conforms to a specification and that is exception tolerant, where exceptions concern the violation of constraints and business rules defined in the specification. Different to existing works, our approach supposes autonomous services ¹ and does not associate these properties to services but to an application logic implemented as a coordination. Of course, when non-functional properties associated to a service coordination depend on the properties of participating services, the programmer should be aware of that and provide non-functional properties partially, model compensating strategies for ensuring them or at least program recovery actions that can signal an exception when a non-functional property is violated.

In general, software engineering techniques have been conceived for supporting the development of applications, and service-based applications should not be the exception. The evolution of the software engineering field has led to the proposal of the MDA (Model Driven

¹A service is autonomous when it does not share its execution state, it can evolve independently of the application that use them, and it does not provide means for controlling its availability.

Architecture) approach. MDA [80] is an important approach for the alignment between high-level information modelling, non-functional requirements and service-based development, because it provides a conceptual structure that extends from the models used by system analysts to different models used by software developers.

The challenge to the use of MDA in modeling of cross-cutting aspects of the application from the more abstract level of design is the ability to use different types of models for different purposes and understand the differences between the terminologies for the non-functional properties [120].

MDA allows the specification of a system as an abstract model, which may be realized as a concrete implementation (e.g. WSDL, BPEL or PEWS). Using a MDA specification it is possible to generate the source code for implementing the software system from an abstract model. The MDA methods provide a skeleton of the program source code, implemented as a source code template where predefined tokens are then replaced with code. MDA methods separate design from architecture. The design addresses the functional requirements while architecture provides the non-functional requirements like scalability, reliability and performance are realized.

1.1 Motivation and Problem Statement

Software construction is moving towards the use of methodologies for controlling the software development process and facilitating the specification of functional and non-functional requirements (NFRs), and its maintenance and evolution for integrating new functionalities. There exist several methodologies, methods and new approaches [40, 93, 103, 51, 21, 79] for software engineering, improving software development and modeling non-functional requirements. These methods can help to ensure the coherence between functional and non-functional properties necessary to the system, particularly when information systems include complex business processes calling web services or legacy applications exported as services. The challenge is to ensure reliability² properties for the complete application. There are still a number of problems to be solved in the development of systems, specially for service-based development, e.g., automatic composition, NFRs specification and modelling, or specific development methodologies for service-based applications.

Service oriented computing is at the origin of an evolution in the field of software devel-

² The concept of reliability of service-oriented applications used in our work is associated to the concepts of *correctness*, *robustness* and *reliability* introduced in (i) *Design by Contract* [71, 74, 73], (ii) referred by [71, 73] and (iii) defined in the database domain.

opment. An important challenge of service oriented development is to ensure the alignment between IT systems and the business logic. Thus, organizations are exploring new mechanisms to bridge the gap between the developed systems and business needs [21]. The literature stresses the need for methodologies and techniques for service oriented analysis and design, claiming that they are the cornerstone in the development of meaningful service-based applications [91]. In this context, some authors argue that the convergence of model-driven software development, service orientation and better techniques for documenting and improving business processes are the key to make real the idea of rapid, accurate development of software that serves, rather than dictates, software users' goals [115].

Service oriented development methodologies providing models, best practices, and reference architectures to build service based applications mainly address functional aspects [11, 25, 41, 93]. Non-functional aspects concerning service and application's semantics, often expressed as requirements and constraints in general purpose methodologies, are not fully considered or they are added once the application has been implemented in order to ensure some level of reliability (e.g., data privacy, exception handling, atomicity, data persistence). This leads to service based applications that are partially specified and that are thereby partially compliant with application requirements.

The use of many different platforms and APIs for the development of Web applications raises the need for an infrastructure that allows an organized execution of the analysis activities, design and implementation, enabling the design, implementation and development of reliable applications.

Even with the growing interest in developing higher-level models and design paradigms, current technology seems limited for expressing constraints as non-functional requirements (NFRs) and associating them to service-oriented applications [31, 33]. NFRs specify global constraints that must be satisfied by the software [99].

Ensuring non-functional properties has been challenging when developing applications because they are both related to the business rules of the application (e.g., defined user rights associated to the execution of applications' functions: only the manager can authorize the modification of a salary.) but also to the technical characteristics of the infrastructure where the application is executed (e.g., the messages transmitted through the network cannot be modified while transmitted). Having a methodology for specifying the application logic and its associated non-functional properties starting in the early phases of the development process is a difficult problem because:

- the designer must make the difference between requirements that concern the application logic and the non-functional requirements;
- the non-functional requirements specified in an abstract way (e.g., as business rules) must provide enough information to be translated into the specification of technical aspects implementing a concrete non-functional property. For example, saying that the payment, the authorization of delivering a package and the modification of the stock must be all together successful otherwise they must be undone, can be translated into an atomicity property associated to these operations (i.e., either the three of them are executed or none of them);
- the methodology must provide general enough concepts for modeling quite different non-functional requirements and strategies to ensure that their abstract specification will lead to the implementation of protocols that will ensure these properties while the application is executed.

In the case of service-oriented applications it is particularly difficult to add non-functional properties because in general the existing protocols and models implementing some of them assume that (i) it is possible to have a global control of the artifacts implementing the application or (ii) at least they export interfaces in order to have information about their execution state, the messages exchanged with others, the conditions in which messages are transmitted on the network.

In the case of service-oriented applications, the services used as construction units are autonomous, they do not share information about the state and data they manage and there are few engagements with respect to their availability and evolution. This aspect calls for flexible protocols and models that can accept a best effort approach for ensuring non-functional properties. For example, to provide security, all services of the composition should be safe; to ensure satisfactory response time, all services of the composition must provide the results obeying restrictions of the client. These assumptions cannot be made, and thus in some cases non-functional properties cannot be completely ensured. The methodology should take into consideration this situation in order to reflect it along the different construction phases.

As services are independent components, ensuring non-functional properties is a challenge. Different studies [15, 6, 29, 56, 116, 60, 110] try to associate non-functional requirements to services using different approaches. Associating non-functional requirements to services compositions can help to ensure that the resulting application is compliant to the user requirements and also with the characteristics of the services it uses.

Our approach combines service-based development, non-functional aspects and model driven architecture (MDA) so that reliable service-based systems may be developed by reducing time delays and increasing results, such as compliance and quality.

1.2 Objectives and Main Results

The main goal of this thesis is to provide a methodology for developing reliable web service applications. Addressing the (i) modelling of functional requirements that can be implemented by service compositions and (ii) non-functional requirements expressed as constraints associated to services and policies to ensure the enforcement of business rules that define the semantics of the application.

1.2.1 Hypothesis

We adopted the following hypothesis for defining the methodology:

1. Business rules and constraints specified as application requirements can be translated into non-functional properties that can be associated to a service composition that implements the application logic of a service-oriented application.
2. We believe that it is possible to model non-functional properties during the early stages of applications development process and that this can be done using an MDD approach. Therefore, we assume that, at least in the context of service-oriented applications, it is possible to propose meta-models that can be specialized for expressing non-functional properties and that can be associated to concepts used for modeling the application logic.
3. We use the notion of service in a general way without necessarily associating it to the Web. A service is a software element implemented by a service provider that exports an interface for calling the service and that is accessible through a network.
4. It is possible to provide a methodology for building service-oriented applications that is agnostic of the data models exchanged by the services, the interface definition language used for describing the services interfaces, the protocols and architectures used for calling services, the composition languages and the execution engines. This implies that our methodology can consider REST and SOAP services with interfaces described in WSDL, semistructured (JSON, XML), unstructured and structured data models, and compositions expressed in any language.

1.2.2 Objectives

Our work proposes the MDA based methodology π SOD-M (*Policy based Software Oriented Development Methodology*) that (i) extends the SOD-M [41] method with the notion of *Policy* [47] for representing non-functional constraints associated to service based applications; (ii) defines the π -PEWS meta-model [105] providing guidelines for expressing the composition and the Policies; and finally, (iii) defines model to model transformation rules for generating the π -PEWS model of a reliable service composition starting from the extended service composition model; and, model to text transformations to generate the corresponding implementation.

This work proposes also an environment that provides tools for using the methodology and implementing reliable services based applications. By defining the π SOD-M model a programmer can design and implement cross-cutting aspects (e.g., exception handling for describing what to do when a service is not available, recovery, persistence aspects) and constraints associated to services, that must be respected for using them (e.g., the fact that a service requires an authentication protocol for executing a method).

π -SOD-M supports the construction of service-oriented applications that implement business processes. Therefore, it proposes a development process based on the definition of models (instances of the meta-modes) and transformations for semi-automatically generating different levels of abstraction that describe a service-oriented application from abstract platform independent views (CIM and PIM level) to platform dependent views and the PSM and implementation levels. We extended the Business and Service views of the SOD-M method [40]. The Business view defines the concepts for modeling a business process, while the Service view defines the concepts for designing services compositions. Our methodology introduces concepts (e.g. NF-requirement, constraint, assertion, contract, policy) in the Policy view for describing constraints associated to services and non-functional properties associated to service processes.

The organization into different levels of abstraction of non-functional properties proposed by π -SOD-M relies on its concept of non-functional Requirement (NFR) defined as a group of semantically correlated non-functional attributes (NFA). For example, *security* is an NFR that comprises attributes such as *confidentiality* and *integrity*. NFR are used for modeling global constraints, NFAs model non-functional characteristics that can be precisely measured, such as *performance* [99]. This kind of characteristics are closer to implementation. Then, the concept non-functional property proposed by [99, 32, 33] is used for modeling different types of software constraints. Thus, it generalizes the concepts NFR and NFA of our methodology. In π SOD-M, NFR and NFA are modelled along the different modeling levels as CONSTRAINT TYPES that can be grouped and transformed into CONTRACT TYPES, that can be transformed

into POLICY TYPES. A constraint type is used for representing a non-functional property.

The objective is to provide a conceptual structure (called framework) and associated environment to:

- Capture the system requirements and specification using high-level abstraction models (represented as computation independent models, CIMs). The objective is to provide concepts for supporting the specification and representation of a business process including its application logic and business rules.
- Semi-automatize the generation of the Platform Independent Models (PIMs) from the specification given by the CIMs. The PIMs specify the business process as a service process and its associated business rules and constraints as policies. A policy expresses actions to do (recovery) if business rules and constraints are not respected.
- Transform such models into platform specific models (PSMs). The PSMs in π -SOD-M meta-model the π -PEWS language³ [105] that will serve as intermediate language for generating executable code in the final phase of the methodology process. The reason is that π -SOD-M was developed in the context of a project where it is used for declaratively implementing service compositions. We defined the π -PEWS meta-model, based on the π -PEWS syntax, and then the mapping rules between PIM and PSM.
- Serialize PSMs into the executable-code that implements the system.

1.2.3 Contributions

The main contributions of our work are:

- **Classification of non-functional requirements for web services:** This classification is based on an analysis of the key features in the development of service-based applications. Associating non-functional requirements to a service composition partially ensures that the resulting application will comply to the user requirements as well as with the characteristics of services user for implementing it.
- **The definition of non-functional requirements meta-models:** π SOD-M defines non-functional requirements meta-models for representing concepts associated to services properties and constraints, and business rules and thereby providing a framework for modelling reliable service-based applications.

³The π -PEWS meta-model was created during the research, according to the π -PEWS language syntax.

- **MDA based methodology for modelling and implementing reliable service-based applications:** π SOD-M defines a set of concepts organized according to three points of view: (i) *policy view*, focusing on the non-functional requirements expressed by policies, (ii) *business view*, focusing on the features and the requirements of the business, and (iii) the *system view*, concentrating on the functionalities and processes that need to be implemented in order to satisfy the business requirements. The modelling process of π SOD-M includes models that are in correspondence with the three different abstraction levels considered by MDA (CIM, PIM and PSM).
- **π SOD-M environment:** We the π SOD-M environment implemented as Eclipse plugins that enables the use of the proposed methodology for specifying functional and non-functional requirements of a reliable service-based application and for semi-automatizing the transformation processes from CIM to PIM and PIM to PSM levels, and then generating executable code.

1.3 Document Organization

The remainder of this document is organized as follows:

- Chapter 2 presents a systematic review of approaches associating non-functional properties to services and service-based applications. It introduces the main concepts used for describing and modeling non-functional requirements in the different phases of service-based methodologies;
- Chapter 3 introduces the π SOD-M methodology proposed in this thesis, used for building reliable service-based applications. The chapter describes the main phases of the methodology which is based on an MDA approach, and the meta-models associated to the CIM, PIM and PSM levels. Developing a service-based system consists in instantiating such meta-models and transforming them along the different levels. This chapter also describes the transformation rules among the meta-models of the π SOD-M methodology, that includes model-to-model and model-to-text transformations.
- Chapter 4 describes the π SOD-M environment for developing service-based systems using this methodology. The chapter describes the architecture of the environment, its main components and functions and the tools for implementing it;
- Chapter 5 describes the experimental scenarios that we conducted for validate the π SOD-M methodology. The chapter introduces three scenarios with different levels of complex-

ity, some focusing on complex application logics and few non-functional requirements, and others having in contrast simple application logics, but complex non-functional requirements. The π SOD-M methodology is systematically used for developing the case studies. Then, a discussion is conducted for presenting lessons learned and for comparing this methodology with a methodology of reference.

- Chapter 6 concludes this document. It enumerates the main results of this work underlining the main contributions and then it discusses future perspectives.
- Some details about our work are described in the appendices:
 - Appendix A describes the method used for source selection and analysis of related works with non-functional requirements for service-based applications;
 - Appendix B presents a list of service-based non-functional requirement concepts and definitions proposed in this thesis;
 - Appendix C presents the π -PEWS language used in this work.

2 *State of the Art*

“Computer science is no more about computers than astronomy is about telescopes.”

Edsger W. Dijkstra

Service oriented development methodologies providing models, best practices, and reference architectures to build service-based applications mainly address functional aspects [103]. Non-functional aspects concerning services and applications semantics are often expressed as requirements and constraints. It is common that these aspects are not fully considered during the development of applications. In many cases, they are considered once the application has been implemented, in order to ensure some level of reliability (e.g., data privacy, exception handling, atomicity, data persistence). This situation leads to service-based applications that are partially specified and that are partially compliant with initial application requirements.

In systems and requirements engineering, a non-functional requirement (NFR) specifies criteria about the behaviour of a system. These criteria may be not related to the results produced by the system, but to other conditions of its performance and execution. Non-functional requirements are often called qualities of a system. They are also referred as “constraints”, “quality attributes”, “quality goals”, “quality of service requirements” or “non-behavioural requirements” [106]. In the case of service-based applications, non-functional requirements concern the application itself as well as its component services.

In the case of service-based applications, non-functional requirements concern the application itself, as well as constraints imposed by the services. Associating non-functional requirements to services composition can help to ensure that the resulting application is compliant to the user requirements and also with the characteristics of the services it uses. As services are independent components, ensuring non-functional properties is a challenge.

Programming non-functional properties is not an easy task and different studies [15, 6, 29, 56, 116, 60, 110, 67] associate non-functional requirements and services using different

approaches. Although they are used as synonyms in most NFR approaches [67], we distinguish the concepts of non-functional requirements and non-functional attributes. NFRs are defined as a group of semantically correlated non-functional attributes (NFA). For example, *security* is an non-functional requirement that comprises attributes such as *confidentiality* and *integrity*. A non-functional attribute describes the characteristics of a functional requirement [99, 33].

This chapter (i) presents the background concepts of this thesis, (ii) presents the state of the art of Web service based methodologies and non-functional requirements for web service based applications, and (iii) analyses the principal methodological concepts for web service development.

Considering the analysis, it will be conducted through a systematic review¹ [62]. The result of this review can be useful to summarize the existing evidence concerning a treatment or technology, to identify gaps in current research in order to suggest areas for further investigation, and to provide a background for new research activities. The bibliography will be analyzed in order to present the different existing concepts, proposals, problems and solutions related with each area.

The analysis is organized in 2 parts to facilitate the understanding of the areas and related works and what their importance in the context of service-based development. Our goal is to identify a common nomenclature for existing NFR and web service based methodologies and propose a specification of these concepts in the context of reliable web services development.

Since the focus of this work is the development of reliable service-based applications, we analyze *what, where* and *how* non-functional requirements are modeled in the development of this type of application. Allied to this, we analyze which methodologies address development of service-based applications and if they model non-functional aspects.

This chapter is organized as follows. Section 2.1 presents the background concepts of this thesis, considering the following domains: model-driven development, service-oriented applications, non-functional requirements and design by contract. Section 2.2 presents the main concepts and works related with non-functional requirements for web service application. Section 2.3 presents the main concepts and works related with methodologies for web service development. Section 2.4 present a non-functional requirements model for service-based development and a classification of the main terms used, divided into levels. Section 2.5 concludes for this chapter.

¹A systematic review provides means of identifying, evaluating, and interpreting the literature relevant to a particular research question or topic area [62].

2.1 Background

The background of this work consists of the works addressing (i) model driven development, (ii) service-oriented applications and (iii) non-functional requirements. A number of approaches, models and solutions proposed in these domains were studied, integrated and adapted for addressing the construction of service-oriented applications in the presence of non-functional requirements. We underline in the following the key concepts that we used from each of the domains of the background of our work.

2.1.1 Model-Driven Development

Model-driven development (MDD) approaches, specially the model-driven architecture approaches provides a set of guidelines for the structuring of specifications, which are expressed as models.

The MDA approach gives the facility to understand complex and real world systems while providing an abstraction of the physical system. This abstract view of the system is represented through the OMG's modeling standards namely the Unified Modeling Language (UML) [88], the Meta-Object Facility (MOF) [80] and the XML Metadata Interchange (XMI).

The Model-driven approach does not reinvent another software development process but it regroups the knowledge of software engineering concepts to express architectures. The MDA approach is mainly based on the Model Driven Development (MDD) [115, 80] which suggests that a system must be first modeled before it can be transformed to suit the final execution platforms.

A MDA-based approach establishes a specific set of layers and transformations. The Computational Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM) are the three MDA model types. Considering this concepts, a system can be expressed as the specification of an execution environment for a set of models, the relationships between models, meta-models and platforms.

Computational Independent Model: CIM is a view of a system without any knowledge of the computation details. The aim of CIM is to link the needs of the application client with the others models that are built in the next development life cycle phases.

Platform Independent Model: PIM is a view of a system without any knowledge of the implementation details. The PIM model requires that model elements contain enough information to represent the system independent of platform.

Computational Specific Model: PSM is a view of a system with specific knowledge of the implementation platform. After the platform independent model has been accurately defined, the platform specific details can be handled in the PSM.

Another important aspect of the MDA approach is the transformation between models. Models transformation relies on a set of mapping rules between models that inform the transformer tool about the patterns it needs to follow to output the target model. Then, consistency and validation of models can be automated during this transformation process, improving the efficiency of generated applications.

2.1.2 Service-Oriented Applications

Service-Oriented Applications are defined based on the Service-Oriented Architecture (SOA) [25, 100, 21] paradigm for organizing and use of distributed computing resources that may be under the control of different domains, providing a uniform meaning for discovery, interaction and use of these resources. SOA is a set of principles and methodologies for designing and developing software in the form of interoperable services.

The use of SOA makes possible the integration of applications that are web based and make use of multiple platforms for its implementation. SOA also defines a interface in terms of protocols and functionality.

As an architecture, SOA relies on principles of service-orientation (Web services). This is motivated by the hypothesis that, if a service provides a simple interface that abstracts its complexity, users can access the services independently, without know the development platform on which it was implemented. So, service-orientation requires loose coupling of services with operating systems and other technologies that underlie applications.

Web Services can be defined as software that can be discovered and invoked through the Web. Normally independent and self descriptive, several initiatives related to web services have been implemented with the intention of providing platforms and languages that allow easy integration of heterogeneous systems. Services are usually built from the following specifications: XML, SOAP [24], WSDL [30], and UDDI [59]. Besides these, it is possible to use other technologies for services, such as ontologies DAML-S [66] that define standards for discov-

ery, description and invocation of services. An example of another initiative that uses service-oriented technology is the Business Process Execution Language for Web Service (BPEL4WS) [86, 87, 61], where a composition of web services can be described through representation of business processes.

Considering composition of services in general, two paradigms guide the compositions: the choreography and orchestration [101]. The orchestration is concerned with the flow of interactions between services (business logic and execution order of services, such as a workflow). The orchestration requires an engine that coordinates the composition execution. The choreography describes the order in which messages are ordered for using the methods exported by one or several services. The choreography is used for expressing the conversations that an application can have with a service. A conversation specifies the sequence of valid messages that a service can receive. These sequences correspond to the protocols for exchanging messages with a service.

2.1.3 Non-Functional Requirements

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Non-functional requirements are often called qualities of a system. However, for [33] there is not a formal definition or a complete list of non-functional requirements. NFRs are classified into consumer-oriented and technically-oriented attributes [85].

Based on [85], during the design process goals are decomposed design alternatives are analyzed with respect to their tradeoff design decisions are rationalized goal achievement is evaluated and a selection is made. In this approach NFR-related knowledge is codified into methods and correlation rules. This way a body of codified NFR-related knowledge offers the needed subject matter and vocabulary and is made available and reusable throughout the process [31].

In the area of software requirements, the term non-functional requirements has been used to refer to concerns not related to the functionality of the software. However, different authors characterize this difference in informal and unequal definitions, such as “*Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet*” [32].

Expressing and enforcing non-functional properties for distributed systems is a well-known problem with several associated existing solutions that have modeled thoroughly them for pro-

viding middleware services. For example in object oriented solutions like CORBA, and component oriented solutions like EJB. Models and protocols have been defined in the distributed systems and the database domains for dealing with non-functional requirements that concern for example, security, tolerance to failure (e.g. transaction monitors, synchronization protocols like 2PC), persistence. Lately service-oriented applications, call for approaches that can include these aspects trying to be loyal to the facility of construction philosophy. Some solutions leave this task to the middleware infrastructure (e.g., the service bus petals² offers some non-functional services), to the engines that execute them. Other approaches like [42] include the code that implements non-functional properties, as protocols within the coordination code that implements the application logic of service-oriented applications or within the communication protocol used for calling services methods (e.g. data encryption of SOAP messages). This later approach is particular to works stemming from the Business Process domain (BPM [87]) and also to WS-* standards.

2.1.4 Design by Contract

We used the notion of reliability in order to have a common approach for dealing with non-functional requirements (i.e., ensuring that the system respects them and defining actions when they are not respected). The concept of reliability of service-oriented applications used in our work is associated to the concepts of *correctness*, *robustness* and *reliability* introduced in (i) *Design by Contract* [71, 74, 73], (ii) referred by [71, 73] and (iii) defined in the database domain.

Design by Contract is a method for developing software. Design by Contract is the principle by which interfaces between software modules should be designed using precise specifications. The principle behind Design by Contract is a class and its client have a contract with each other. Contracts define mutual benefits and obligations (pre-conditions and post-conditions, respectively), and constraints (invariant). Contracts are defined in program code through comments of their own programming language and is translated into executable code by the compiler. The properties (predicates) of the contract are known as assertions (assertions).

Assertions are representations of states. Expressions are true at certain points of a code [71], becomes important when one wants to prove the correctness a program. To be able to prove the correctness of a given program or a piece of code, it is necessary to define logical expressions that represent desired states. If one of the conditions is not satisfied, there was something wrong with either, the defined assertion or the definition of the program.

²<http://petals.ow2.org>

Design by Contract defines reliability as a concern that integrates the ability of a software element to perform the tasks as defined in its specification (correctness) and that reacts appropriately to abnormal conditions (robustness). Furthermore, the notion of contract proposed by [71, 74, 73] enables the specification of actions to take when one of these properties is not respected by a service. [71, 73] associate the notion of reliability to the conformance between the requirements specification of a software component and its implementation with the objectives to avoid possible bugs. Thus, reliability is associated to the absence of bugs of a software component with respect to its specification (during the design phase).

In the database domain, reliability is often related to a kind of failure tolerance in the sense that a database application is reliable if it handles exception and it ensures that data consistency will be maintained despite possible physical and logic failures [5, 69].

Based on this background concepts, the challenge was to analyze approaches concerning the non-functional properties and generalize them into the meta-models that are core components for a service-oriented methodology.

2.2 Non-Functional Requirements for Service-Based Applications

This section presents a systematic review of the non-functional requirements and properties used in the context of service-oriented development. The purpose of our analysis is:

- to identify the concepts, properties and notations used in the service-based systems development;
- to find if there is any pattern or relationship between non-functional requirement concepts in different modelling levels.

The functional and non-functional requirements are refined in each phase of the development. As a result, the granularity of the requirement becomes tiner and more precise.

We propose 7 research questions (RQ_1 to RQ_7) to guide our analysis of the bibliography about non-functional requirements. For each question we define a set of possible answers in order to guide the analysis. These possible answers are defined from our knowledge on each topic. The questions are closely related to service-oriented development with a NFR focus. They are:

- **RQ₁**: How are NFR modelled by existing methodologies for developing Web services applications?
 - *Answer is specific to each proposal*
- **RQ₂**: Which are the NFR that are more frequently used by developing web services?
 - *Possible answers: security / availability / portability / ... / reliability / performance*
- **RQ₃**: What is the software development approach used in the work?
 - *Possible answers: Model driven approach (*MDD) / Ontology (*Ont) / Formal method (*FM) / Artificial intelligence (*AI) / Business Process Modeling (*BP) Traditional (*TDT)*
- **RQ₄³**: What is the discipline of the “non-functional requirements” / “non-functional properties” used in the work?
 - *Possible answers: Software architecture / QoS model / Language definition / Methodology / etc*
- **RQ₅**: Does the paper propose a (meta)model describing and analyzing NFR? Is there any relationship between the proposed non-functional requirements (meta)model and business services⁴?
 - *Possible answers: yes / no – yes / no*
- **RQ₆**: Are the non-functional aspects treated in an independent way?
 - *Possible answers: single / composition*
- **RQ₇**: Which is the publication year of the paper?
 - *Possible answers: Year of publication*

The research questions identify characteristics of the service-oriented application development, especially the modelling of non-functional requirements/properties, how they are addressed and if they are classified.

³ Since the objective of our work was to develop a methodology for supporting the service-oriented application development process, the goal of RQ₄ was to look for a “standard” or a classification of non-functional requirements for service-oriented development. As any research question, it can be too particular and can leave some works outside its scope (e.g., KAOS [111] and i-star [118] were not part of the result).

⁴ Note that the term business services is used to make the difference between the services that participate in the implementation of an application logic from the services used for implementing non-functional aspects. For instance, CORBA domain services are those equivalent to business services. BPMN [87] defines a similar term called *Service operation* that is part of the business process.

2.2.1 Concepts and Works

In order to classify the available works using the research questions, we proceed with a systematic selection of the published papers in our field of interest.

We used the approach described in Appendix A for searching, collecting and selecting some works related with non-functional requirements for developing service-based applications. This approach selects published articles from known sources, such as *IEEE*, *ACM* and *Science Direct*, to be considered for inclusion in our study. Based on these data we analyzed concepts used in each work for the description of NFRs and the differences between them. The notation used to refer non-functional requirements will be highlighted in *italic*, including the set of values that can be associated to each concept. For example, a notation used by a work can be *quality property* or *non-functional concern*, and the values associated with its notation are *security*, *reliability*, *transaction*, etc. Thus, considering each research question, we analyze a set of works in order to identify any pattern or divergence between the analyzed literature.

Babamir et al.[15] ranks services *quality properties* in three categories (business level, service level, system level). *Quality properties* are associated with *quality constraints*, that are defined as assertions or propositional logic formulas. *Non-functional attributes*, *composition model entity* and *model entity* are the notations used by Xiao et al. [116] for classifying the different concepts for non-functional requirements modeling. Non-functional attributes (NFAs) describe the non-functional aspects in the abstract process models. The framework to model NFRs proposes to annotate composition models with NFAs.

D’Ambrogio [37] uses the term *quality characteristics* to group similar characteristics into *quality categories*. Each *quality characteristic* is quantified by *quality dimensions*. *Quality characteristic* is a quantified QoS aspect, for example *latency*, *throughput*, *reliability*, *availability*, etc. *Quality characteristics* of a common subject are grouped into abstract *quality categories*, for example *performance* (for latency and throughput characteristics) and *dependability* (for reliability and availability characteristics).

The terms *category*, *sub-category* and *property* are used by Yeom et al.[117] to classify non-functional requirements. *Business*, *service* and *system* are the possible values to be associated with the proposed terms, and a *sub-category* can be *security*, *value*, *interoperability*, etc. The work in [117] defines a *web services quality model*, which considers non-functional properties in several aspects. In this model, web services qualities are classified in categories and sub-categories. Chollet et al.[29] uses only two terms to classify and relate quality proper-

ties with services, they are: *activity* and *quality property*. Each activity represents a functional property that can be divided in sub-activities, depending on its granularity. For non-functional requirements, the work in [29] describes the possibility of creating different meta-models for each *quality property*, and then relate them with activities.

Schmeling et al.[102] uses the *non-functional concerns (NFC)* notation to describe NFRs. This term encompasses two aspects: the specification of NFCs and their realization. [102] defines that a *functional concern (FC)* is a consistent piece of functionality that is part of a software system. *Non-functional concern (NFC)* is a general term describing a matter of interest or importance that does correspond to a non-functional requirement of a system, for example, *security*, *reliability*, *transactional behavior*, etc. A *non-functional action* represents some behavior that results in a *non-functional attribute*. An example of *non-functional action* is *encryption*, which realizes the *non-functional attribute*, *confidentiality*. A *non-functional activity* is also used as a term, which means to encapsulate the control flow of *non-functional action* that apply to the same subject. The term is used in analogy to activity and action in UML2 [102].

Ceri et al.[28] uses the terms *policy*, *rule*, *condition* and *action model* to specify NFRs, and in a similar way, Agarwal et al.[6] also uses the concept of *service policy* associated with the concept of service. Each service is also associated with a *service property*, which may have a specific value (*security*, *reliability*, and so on). Each service is also associated with a *unit function*, that represents one or more requirements.

Ovaska et al.[90] uses *quality attribute*, *category*, *conceptual layer* and *importance* to organize and classify the NFRs, Pastrana et al.[95] uses the term *contract* to describe non-functional requirements. In a *contract* it is possible to define pre-conditions, post-conditions and invariants. A *web service* can have many *contracts*, that defines many *assertions* and are associated with *quality properties*.

Some related papers do not have any nomenclature to classify non-functional requirements. However, some of them uses the *attribute* notation [119, 18, 60], other use *properties* [48], *factors* [81, 56], *characteristics* [45], *quality level* [44] or *values* [108, 18].

2.2.2 Analysis

Although there exist many different types of notation used for classifying non-functional requirements, in general, the values associated to them are the same, *i.e.* *security*, *performance*, *reliability*, *usability*, *availability*, etc. What distinguishes the different approaches is the adoption of different NFR hierarchies in order to prioritize or classify the quality requirements. An-

other interesting factor is that each work uses different approaches to model these requirements. Thus, there are different kinds of notations for non-functional requirements.

A number of approaches use [37, 29, 102, 18, 48, 90] MDD (Model Driven Development) for designing and developing systems. Fabra *et al.* [48] presents a complete methodology that does not focus on non-functional requirements. Fabra *et al.* [48] describes the importance of the use of MDD in the development of service-oriented applications. [108, 119] use formal methods to define a development process based on NFR for web services, [6, 95] use ontologies for the definitions and modeling of non-functional requirements and [116, 56] use Business Process Modeling (BPM) for system specification, including NFR. Most works focus on composition service modeling while others define requirements models for representing the properties they use.

In [15, 117] non-functional properties for web services are classified into three views such as *service level*, *system level* and *business level*. In the *business* level the quality properties are: *service charge*, *compensation rate*, *penalty rate* and *reputation*; at the *service* level the quality properties are: *performance* and *stability*; and at the *system* level the quality properties are classified into: *manageability*, *interoperability*, *business processing* and *security*.

In the method defined in [116], each task in the process model is annotated with the NFAs (*non-functional attributes*). During the design phase, the service composition and the definition of NFAs are separated. Then, each task in the process model is annotated with the corresponding NFA. The attributes are related with tasks or data item. For data, the NFAs are: *value* and *range*; and for tasks the NFAs are: *cost*, *time*, *resources* and *expression*.

D'Ambrogio [37] presents a WSDL extension for describing the QoS of web services. The process is based on MDA. The work presents a catalog of *QoS characteristics* for the web service domain and the Q-WSDL (Quality WSDL) meta-model for modeling QoS properties in web services. The properties presented are: *availability*, *reliability* and *access control*. [29] presents a security meta-model for web service composition. The non-functional requirements considered are *authentication*, *integrity* and *confidentiality*. Each property is related with a service activity. In [102], authors present an approach and a toolset for specifying and implementing the composition of several non-functional properties of Web services. The non-functional attributes described in [102] are: *confidentiality*, *integrity (security concern)* and *response time (performance concern)*.

The work presented in [108] describes steps to design a selection mechanism of services identified as candidates for participating in a composition, considering their quality properties. The steps are: (i) identification of relevant QoS information; (ii) identification of basic compo-

sition patterns and QoS aggregation rules for these patterns; and (iii) definition of a selection mechanism of service candidates. As QoS properties considered in [108] are: *performance*, *cost*, *reliability* and *availability*.

Among the properties presented we highlight *security* and *performance*. Users usually need to access data securely and quickly. Most studies have both properties as the most important non-functional requirements. *Reliability* is also an important non-functional requirement presented in some works and required by end users.

We can now relate the research questions presented in section 2.2, with the works described above. Tables 1 and 2 show the results.

The vocabulary used for naming and characterizing NFR is not stable, 5 works out 360 (1.6%). 19 of the total of works chosen (26.31%) propose a classification of non-functional requirements (see appendix A).

There are other works [1, 35, 34, 52, 103] that consider and propose non-functional requirements classifications. A number of software attributes are defined as requirements in [103, 1]. According to [1], the most common non-functional properties are: *performance*; interface; operational; resource; verification; acceptance; *maintainability*; documentation; *security*; *portability*; quality; *reliability*; *usability*; and *safety*. We highlight those that are used in most analyzed classifications. [1] proposes an NFR specification and a template for specifying requirements, considering both, functional and non-functional properties.

The NFRs are classified in 4 main concepts by [35]. The classification uses *quality properties*. These properties are: *performance*; *security*; *cost*; and *usability*. Pastrana [95] describes an ontology based methodology and uses a NFR classification based in some properties, most already described in other works. However that work is the only one to use *scalability*, *capacity* and *precision* properties. Considering web services development, these properties are not very frequently used, however, considering data processing, these properties can be important in cases of large data processing requests through services.

Sommerville [103] classifies requirements, either functional or non-functional, in three main blocks: *process*, *product* and *external*. [103] also defines a sub-classification considering the system domain. Software requirements are classified as: *usability*, *reliability*, *safety*, *efficiency*, *performance* and *capacity*.

Some requirements may determine the system design, for example, (i) keep certain functions in separate modules; (ii) check data integrity for critical variables; and (iii) permit only limited communication (requester / provider), are examples of restrictive requirements [1].

Reference	RQ_1 : NFR concepts	RQ_2 : NFR values	RQ_3 : Approach	RQ_4 : Domain / Scope
Babamir et al. [15]	property / category / constraint	responsiveness / availability / performance / sla properties	TDT	Software architecture
Yeom et al. [117]	category / sub-category / property	business value / performance / stability / manageability / security / business processing interoperability	TDT	QoS model
Xiao et al. [116]	NF attribute	time / cost / resource	BP	Business processes modeling
D'Ambrogio [37]	characteristics / category / dimension	availability / reliability / access control	MDD	WSDL extension
Chollet et al. [29]	activity / NF attribute	security	MDD	Orchestration Framework
Schmeling et al. [102]	NF concern / NF attribute / NF action / NF activity	security	MDD	Web service composition process
Thißen et al. [108]	NF value	performance / reliability / cost / availability	FM	Software architecture
Zhang et al. [119]	attribute / predicate	security	FM	Access control
Basin et al. [18]	attribute	security	MDD	System architecture
Ceri et al. [28]	policy / rule / condition / action	n.a.	TDT	Context-aware applications
Fabra et al. [48]	property	storage / processing (*case study)	MDD	Web service methodology
Modica et al. [44]	quality level	sla properties	TDT	Service oriented architecture
Ovaska et al. [90]	attribute / category	security / reliability	MDD	Model development
Agarwa et al. [6]	property / policy / function	<i>not explicitly defined</i>	Ont	Policy language
Jeong et al. [60]	NF attribute	operation cost / performance / availability / accessibility / security / interoperability / usability / user satisfaction	AI	Service oriented architecture
Pastrana et al. [95]	NF property / contract / assertion / NF behaviour	performance / reliability / scalability / capacity / robustness / precision / accessibility / availability / interoperability / security	Ont	Web service methodology
Diamadopoulou et al. [45]	NF characteristic	user' subjective perception	TDT	Web service selection
Gutierrez et al. [56]	NF factor / NF sub-factor	Security	BP	Web service development process
Mohanty et al. [81]	NF attribute or NF factor	reliability / performance / integrity / usability / response time / documentation	AI	Artificial intelligent / Web services classification

Table 1: Research question results - RQ_1 , RQ_2 , RQ_3 , RQ_4 .

These examples are related with some NFR concepts as: *security*, *reliability* and *availability*.

In the service-based development there is a clear difference between the business, service and system levels. Quality requirements are treated differently in each of these levels. Despite the different nomenclatures, they can be described in general as *non-functional concerns/requirements* and *non-functional attributes*.

Restrictions are usually related to use cases and functional requirements ([1] and [103]), and in most cases, a service activity can be represented as an use case. This is the way in which

Reference	RQ_5 : Service model – Business services	RQ_6 : Service type	RQ_7 : Year of publication
Babamir et al. [15]	no – yes	composition	2010
Yeom et al. [117]	yes – yes	single	2006
Xiao et al. [116]	no – no	composition	2008
D’Ambrogio [37]	yes – no	composition	2006
Chollet et al. [29]	yes – yes	composition	2009
Schmeling et al. [102]	no – no	composition	2011
Thißen et al. [108]	no – yes	composition	2006
Zhang et al. [119]	no – no	single	2005
Basin et al. [18]	yes – no	single / composition	2006
Ceri et al. [28]	no – no	single	2007
Fabra et al. [48]	yes – yes	composition	2011
Modica et al. [44]	no – no	composition	2009
Ovaska et al. [90]	yes – no	single	2010
Agarwa et al. [6]	yes – no	single / composition	2009
Jeong et al. [60]	no – no	composition	2009
Pastrana et al. [95]	yes – no	composition	2011
Diamadopoulou et al. [45]	no – no	composition	2008
Gutierrez et al. [56]	no – no	single / composition	2010
Mohanty et al. [81]	no – no	single	2010

Table 2: Research question results - RQ_5 , RQ_6 , RQ_7 .

non-functional requirements are related with web services. They do not propose a specific way for designing services constraints, they assume that a service is modeled as a use case, and it has quality requirements.

At the service level, each service activity is related in some way with the concept of contract. Contracts can be grouped into policies and their rules. Policies are directly related to concepts like quality, security, performance and availability; contracts are associated with non-functional attributes.

Due to the variety of tools and new approaches for web service development, there is not (yet) a consensus on a software process methodology for web services [93, 92, 76, 79, 49, 98, 50].

Works proposing methodologies can be classified into two types: (i) those that propose new approaches for non-functional properties guarantees; and (ii) those proposing service-oriented development methodologies for the whole development process.

[58] proposes *Design by Contract* for web services. It is possible to describe three levels for specifying contracts: *implementation level*, *XML level* and *model level*. Design by Contract applied to web services addresses the verification of web services through runtime checkers, before the deployment, such as *jmlrac* [64], by adding behavioral information to the specification of services, using JML [64].

CDL (*Contract Definition Language*) [76] is a XML-based description language, for describing contracts for services. The development with CDL offers an architecture framework,

design standards and a methodology [78, 75, 77, 79], that can be easily understood and applied to the development of applications. The greatest difficulty is that the language only represents contracts for services. Its specification is generated by refining B [4] machines that describe the services and their compositions.

2.3 Methodologies for Service Oriented Development

In this section, we present an analysis of some methodologies for service-oriented development. We defined 5 research questions to guide our analysis. The questions are closely related to service-based systems. The criteria used for evaluation and comparison are:

- **RQ₈**: What is the scope of the methodology?
 - *This item will review the major phases and the purpose of each particular phase in the context of the project and service-oriented development.*
- **RQ₉**: How does the methodology define the development process according to the adopted notation?
 - *This item describes which notation is used by the methodology for designing its models, e.g., MDA, UML, SysML, etc.*
- **RQ₁₀**: Does the methodology use a formalism? If so, What is the formalism used for specifying services?
 - *The purpose of this item is to determine whether the methodology proposes the use of a formalism for specifying the services, their compositions and iterations.*
- **RQ₁₁**: What is the approach for describing services?
 - *The purpose of this item is to determine the approach adopted by the methodology for describing the development process.*
- **RQ₁₂**: Which models does the methodology propose?
- **RQ₁₃**: Does the methodology defines service's choreography or service's orchestration?
 - *This item discusses the models proposed by methodologies for specifying the system behavior and modeling high-level business requirements.*

Along with the research questions, we also compared the MDA models proposed by different methodologies.

In general, for service-based systems, there is no specific method to conduct the development process. Our analysis, helped us to identify how these methodologies define activities and models to design reliable services, as well as service-based applications.

2.3.1 Concepts and Works

Over the last few years, a number of approaches have been proposed for the development of web services. These approaches range from the proposal of new languages for web service descriptions [8, 83, 16, 68, 101] to techniques to support phases of the development cycle of this kind of software [23, 22]. In general, the objective of these approaches is to solve specific problems, like supporting transactions or QoS, in order to improve the security and reliability service-based applications. Some proposals address service composition: workflow definition [113, 82] or semantic equivalence between services [14]. The proposed solutions come from many communities, including those of Theoretical Computer Science [101, 114, 57, 38, 27], Software Engineering [26, 113, 112, 12, 70], Programming Languages [83, 8] and Databases [96, 3]. However, in spite of the variety of tools, there is not (yet) a consensus on a software process methodology for web services.

Considering the composition of services in general, two paradigms guide the compositions: the choreography and orchestration [101]. The orchestration is concerned with the flow of interactions between services (business logic and execution order of services, such as a workflow). The orchestration requires an engine that coordinates the composition execution. The choreography describes the order in which messages are ordered for using the methods exported by one or several services. The choreography is used for expressing the conversations that an application can have with a service. A conversation specifies the sequence of valid messages that a service can receive. These sequences correspond to the protocols for exchanging messages with a service.

There are methodologies that address the service-based development towards a standard or a new way to develop reliable service-based applications. The methodologies analyzed in this work are: SOD-M [39] and SOMF [21] representing the model based development for web services; S-Cube [94] representing the business processes and service-based development; SOMA [10] that is a methodology described by IBM for SOA solutions; Extended SOA [93] merges RUP[2] and BPM[61] concepts for service modeling; DEVISE [43] is a methodology for building service-based infrastructure for collaborative enterprises. Furthermore, there are

other proposals, such as the WIED model [109] that acts as a bridge between business modeling and design models. Also, traditional approaches for software engineering [103] are being applied to service development.

SOD-M: Service-Oriented Development Method [40] proposes a service-oriented approach and model-based development for web systems. SOD-M proposes models and standards primarily focusing on the development of the systems behavioral characteristics, setting standards for the construction of business models at a high-level of abstraction. The approach describes three meta-models organized according to the MDA (*Model Driven Architecture*) [80] architecture levels: CIM (*Computational Independent Models*), PIM (*Platform Independent Models*) and PSM (*Platform Specific Models*).

At the CIM level, 2 models are defined: *value model* [53] and *BPMN model*; At the PIM and PSM levels, DSL models for service. The PIM-level models are: *use case*, *extended use case*, *service process* and *service composition*. The PSM level models are: *web service interface*, *extended composition service* and *business logic*.

The methodology provides a framework for the development of service-oriented applications with models that can express the whole development process of services-based applications. However, SOD-M has no support for describing and modelling non-functional requirements.

The basis of SOD-M is a set of concepts guiding the whole development, including transformations between models. The concepts are represented through a meta-model. This meta-model describes concepts of both the business and system views.

In SOD-M, the PIM level models the entire structure of the application flow, while, the PSM level provides transformations towards more specific platforms.

The methodology provides model transformations among: *CIM-to-PIM*, *PIM-to-PIM* and *PIM-to-PSM* transformations. Given an abstract model at the CIM level, it is possible to apply transformations for generating a model of the PSM level. In this context, it is necessary to follow the process activities described by the methodology. The model to model transformations are defined using ATL [55].

SOD-M defines a set of concepts according to two points of view: (i) *business*, focusing on the characteristics and requirements of the organization and (ii) *system requirements*, focusing on features and processes to be implemented in order application requirements. In this way, SOD-M aims to simplify the design of service-oriented applications, as well as its implementation using current technologies.

S-Cube: The S-Cube Framework [94] proposes an integrated structure for developing service-based applications. S-Cube offers three areas: *Business Process Management, Composition and Coordination of Services; and Infrastructure*. These areas are the backbone of the framework that are directly linked to three other areas for supporting systems development: *Engineering and Software Design; Monitoring; and Security and Software Quality*.

The methodology aims to guide the development of applications and describes some essential activities, such as (i) *description of business objectives*, (ii) *domain assumptions defining*, which are pre-conditions to be met for a particular application domain, (iii) *description of domains*, and (iv) *description of scenarios for each domain*.

The S-Cube methodology does not provide an exhaustive list of rules for describing services. The S-Cube framework provides activities in various service-oriented development areas, however, it is still required to apply its concepts in real case studies to give an idea of its application, given the fact that its structure is very complex and multidisciplinary.

DEVISE: [43] identifies issues to be considered in service-based design, and gives generic guidelines for addressing them. DEVISE helps the developer in the design of new applications as a collection of web services, and provides means of porting existing applications to services-based platforms.

DEVISE [43] is a methodology for defining infrastructure Web services and prescribes adherence to a thought process that entails the division of the design of a Web Services based collaborative application.

SOMA: *Service-Oriented Modeling and Architecture* [10] is a methodology by IBM for SOA solutions. SOMA defines a seven-phase development life-cycle: *business modeling and transformation; management solution, identification, specification, realization, implementation*, and *monitoring of implementation and management*.

At each stage, different tasks are carefully defined, such as the definition of business architecture, development of service model, specification of services, etc.

The SOMA conceptual model is based on the SOA architectural style it defines an interaction model with three main parts: (i) *the service provider* (which publishes a service description and provides the implementation for the service); (ii) *consumer services* (which can use the URI for the service description directly or can find the service description in a service registry for the call); and the (iii) *service broker* (which provides and maintains a record of services).

SOMA proposes the use of IBM Rational platform for SOA developing, and other different

IBM tools that are available to analysts, software architects and application developers.

SOMF: *Service-Oriented Modeling Framework* [21] is a model oriented methodology for modeling software with the best practices of software project activities and different architectural setting. SOMF can be used to describe enterprise architectures, service-oriented architecture (SOA) and cloud computing. SOMF offers a variety of modeling practices and disciplines that can contribute to developing a successful service-oriented applications. The main goals of SOMF are [21]:

1. A methodology describes SOMF modeling activities and each model transformation;
2. The diagrams are created obeying some project patterns.

The methodology's model transformations in SOMF aim to describe and refine aspects in the system development process. The models are: *discovery model, analysis model, design model, architectural model, implementation model, quality model, operations model, business model, governance model.*

Extended-SOA: [93] adopts the lifecycle of web development services focussing on the *analysis, design and development* phases, with approaches that are centered in functional requirements. According to [93], a specific methodology for developing services-based is important to specify, build, improve and customize business processes available on the Web.

Extended SOA has a hierarchical structure for the development of web services, where layers are defined as follows: *Business (Service) Domain, Business Processes, Business Services, Infrastructure Services, Component-based Service Realizations and Operational Systems.*

The methodology is partly based on other successful models and related development, such as the Rational Unified Process (RUP), Components-based development and Business Process Modeling (BPM). The difference with those models is that Extended SOA life cycle focuses specifically on service-oriented development.

Extended SOA is based on three principles for the design and development of services, they are: *Service coupling, Cohesion of service and Service granularity.*

Considering service-oriented projects, it is preferable to create high-level interfaces with coarse granularity to implement a complete business process, since multiple calls of service increases network traffic. As the services are in a cooperative environment, it is unfeasible to create services with finer granularity. It is preferable to create functions with finer granularity in

a local environment. Thus, the methodology aim is to achieve services integration and interoperability. The Extended SOA phases are: *Planning, Analysis, Design, Implementation, Testing, Provisioning, Implementation, Execution and Monitor*.

Traditional Software Engineering Approach: Sommerville [103] proposes a general approach for the development of applications that use web services. Its structure uses the activities of design, development, construction and testing of web services and their compositions. The proposal is based on the traditional concepts of software engineering, such as:

- Software reuse: standards-based web service that provides a mechanism for inter-organizational computing;
- Use process engineering services to produce reusable web services;
- Perform composition of services to facilitate and improve the development of applications;
- Show how business process models can be used for the design of service-oriented systems.

Service engineering allows the development of reliable and reusable services. Thus the entire development life cycle focusses on the reuse of independent services.

A service must be designed as a abstraction that can be reused by different systems. The main activities for service engineering are: *(i)* Identification of candidate services; *(ii)* Service design; and *(iii)* Service implementation.

2.3.2 Analysis

Our analysis is based on the research questions 8 to 12 described at the beginning of this section. Table 3 summarizes of the main features of the methodologies described above, considering the research questions.

S-Cube and SOMF provide concepts for modeling non-functional requirements, however they have specificities that hinder the development. SOMF proposes a notation for modeling and does not have a environment to assist the development process, thus hindering the design of models and their transformations.

Table 3: Methodologies' Analysis.

	Service-Based Development Methodologies						
	SOD-M [39]	S-Cube [94]	SOMA [10]	SOMF [21]	DEVISE [43]	Extended SOA [93]	Traditional Method [103]
Methodology's Scope	<i>complete</i>	<i>complete</i>	<i>complete</i>	<i>analyse, design</i>	<i>complete</i>	<i>complete</i>	<i>analyse, design implementation</i>
Methodology Models	<i>e-value, BPMN, UML</i>	<i>UML</i>	<i>UML</i>	<i>own notation</i>	<i>UML</i>	<i>BPEL, UML</i>	<i>UML</i>
Formalism Used	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>
Service Specification	<i>no</i>	<i>no</i>	<i>no</i>	<i>no</i>	<i>no</i>	<i>no</i>	<i>no</i>
Choreography / Orchestration	<i>Orch</i>	<i>Orch</i>	<i>Orch</i>	<i>Orch</i>	<i>N/A</i>	<i>Chor</i>	<i>N/A</i>
NFR	<i>no</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>	<i>no</i>	<i>no</i>
Business	<i>yes</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>
Use Case	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>
Service Composition	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>yes</i>
Abstract Level	<i>CIM, PIM, PSM</i>	<i>N/A</i>	<i>CIM, PIM</i>	<i>N/A</i>	<i>N/A</i>	<i>CIM, PIM, PSM</i>	<i>N/A</i>
Model Transformation	<i>yes</i>	<i>no</i>	<i>yes</i>	<i>no</i>	<i>no</i>	<i>yes</i>	<i>no</i>

The Extended-SOA methodology and the use of traditional methods for the development of service applications seems promising, because the developer has a vision of the whole development structure. The disadvantage of these methods is that they do not provide a way to represent the service specification, such as WSDL or formalism. They also do not provide a tool for development support. Another disadvantage of traditional methods, when applied to service-based development, is the fact that these methods do not have the focus in the development of services.

In the same way S-Cube, SOD-M and SOMA provide robust processes and concepts for service-oriented development. Both methodologies have focused on model driven development. However, these methods do not provide models for representing non-functional properties. S-Cube uses UML to represent project models, covering the full development life-cycle, describing each service and their compositions. The S-Cube structure is quite extensive and complete, however, S-Cube does not provide formalism for specifying service composition.

None of the analyzed methods uses a formalism for its description and specification of services. Most of them use UML or some UML extension for model representation, and only the SOMF approach does not address the full life-cycle of development, restricting the scope in the analysis and design phases.

Considering the methodology analysis, we conclude that there are some approaches that address modeling non-functional aspects as the methodology basis. Several methods propose particular features to improve the service-based development. Some proposals consider the whole life-cycle, however, do not consider non-functional aspects that should be designed when developing applications that provide and use web services.

Some development methodologies have worked with the MDA approach, defining models for the different levels of abstraction of this architecture and in some cases, such as SOD-M, DEVISE and SOMA, defining rules for automatic transformation between these models. SOD-M is a method for service-based development using MDA, and thus define models at CIM, PIM and PSM of the proposed architecture.

To conclude, based on our analysis of the data presented in table 3, an interesting methodology for reliable service-based development needs to address: *(i)* the definition of a service-oriented approach-oriented models for the development of applications, *(ii)* model non-functional properties, using UML for this, *(iii)* integrate the proposed method in the context of MDA, providing the representation of the various applications levels, CIM, PIM and PSM, and *(iv)* provide transformation mechanism between models, focussing on quality requirements.

2.4 Classification of Non-Functional Requirements for Service-Based Applications

According to the previous analysis, we define the main non-functional requirement concepts that are associated with service-based development. We also present a synthesis of the concepts common to the analyzed approaches used for modelling non-functional requirements of service-based applications. The NFRs have been classified into three levels, *business*, *services* and *systems* (also adopted by [102]). According to each modeling level, non-functional requirements are being refined from the business level to system level.

A classification of NFRs can be seen in [117]. However, in [117] NFRs are not applied on data, but to functions and service performance. It is important to classify the requirements of business and data (value) restrictions, because web services can be executed in different contexts.

In Sections 2.4.1 and 2.4.2 we enumerate the concepts (NFR meta-model) and values (NFR classification) that we consider important for modeling non-functional requirements for service applications.

2.4.1 NFR Meta-Model

Figure 1 shows the relationship between the concepts⁵ we consider important for modelling quality requirements used in service-based development.

A REQUIREMENT⁶, whether functional or non-functional, can be represented by one or more use cases. A use case represents a *Service Activity*. For example, a paying process can be modeled through the use cases that represent withdrawal and deposit transactions and service provider accounts. A payment process also requires the guarantee of a complete transaction and data security. Thus several use cases model a single requirement, and can be implemented as services.

Each use case has *business* or *value* constraints. *Business constraints* are restrictions on functions and how they may be implemented. The *value constraints* are restrictions on the service interface, expressing the desired values for input and output data. Each constraint is associated with NFAs.

A CONTRACT is a set of constraints for the same function. For example, a contract for

⁵The appendix B provides a description of each concept presented in figure 1.

⁶We use this TYPEFACE to refer to the concepts of the model.

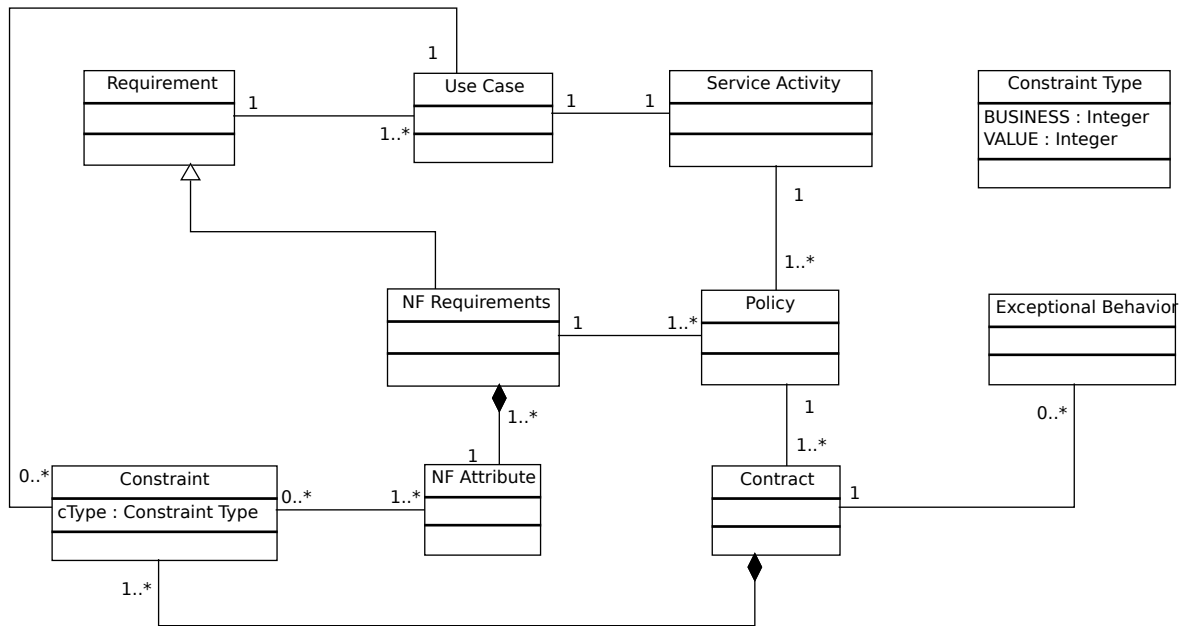


Figure 1: Service-Based Non-Functional Requirement Model.

the payment operation. The constraints for payment are: (i) the value amount should not be less than 10 euros and (ii) the user should always receive a purchase confirmation by phone message. These restrictions are grouped into a single contract for payment verification.

An EXCEPTIONAL BEHAVIOR happens when a contract is not respected. When this happens a new function is called or the process is stopped. For example, if the bank does not authorize the payment, the system offers alternative forms of payment such as PayPal.

Finally a POLICY groups similar contracts. For example, security contracts are grouped into a security policy and performance contracts are grouped into a performance policy.

2.4.2 NFR Classification

Table 4 shows the NFR classification we propose, organized as follows (by column): (i) the level of abstraction, (ii) the proper term for this kind of abstraction and (iii) possible values to be used. The rows represent the abstraction level for modeling the NFR. The highest level is the *Business level*, the intermediary is the *Service level*; and finally the *System level*.

At the business modeling level, non-functional requirements are classified into business restrictions. We have adopted, *business* and *value* constraints to address the most abstract levels of restrictions in business modeling.

- **Business Constraint** - Represents rules for system development in terms of resource

Modeling Level	Concept / Notation	NFR / NFA
Business Level	Constraint	Business Constraint, Value Constraint
Service Level	Contract	Integrity, Transaction, Accessibility, Cost, Time Constraint, Platform, Privacy, Authentication, Resource, Capacity, Confidentiability
System Level	Policy	Security, Performance, Interoperability, Scalability, Reliability, Usability, Availability

Table 4: Non-Functional Requirements Classification.

availability, interoperability, performance, dependencies, timescales. Thus, *Business Constraint* represents restrictions over business activities.

- **Value Constraint** - Defines valid subsets of values of a variable of a given type. In the context of services, value constraint are restrictions on the data on service calls (input and output values), expressing which range of values is allowed. This can be applied to authentication policies, access control, data integrity, cost of services, privacy, and other factors.

The following items describe the non-functional aspects of the service level. At the service level, the non-functional properties will be guaranteed upon implementation of the service requirements.

- **Conformity** - Represents the degree at which a Web service function fulfills an application requirement.
- **Time constraint** - This property can be associated to the execution time of a particular service. This property is related to services performance and availability.
- **Capacity** - Represents the degree at which a service can process a given volume of data or requests.
- **Cost** - Represents the cost to run a service. It can be in time, bandwidth and money.

- **Privacy** - The challenge in data privacy is to share data while protecting personally identifiable information. Privacy of Web services is the ability of the service must have to preserve the data and user information.
- **Authentication** - Represents the process of making sure that the person who is asking to use the web service is really the person that they claim to be.
- **Accessibility** - Represents the quality aspect of a web service related with the degree to which it is capable of serving a request.
- **Access control** - Refers to exerting control over who can interact with a resource, *e.g.* computer-based information system, a service or a database [5].
- **Confidentiality** - Confidentiality ensures that information is not accessed by unauthorized people. The private data is accessed only by authorized users.
- **Integrity** - Define the security attributes of the system, restricting access to features or data to certain users and protecting the privacy of data entered into the software [54]. It also represents the ability of the web service to maintain the correctness of the interaction which is ensured by the proper execution of a web service transaction.
- **Transaction** - A transaction is a unit of work performed to maintain a data integrity in a known and consistent state [5].
- **Resource** - A resource is any physical or virtual component of limited availability within a computer or information management system. Computer resources include means for receiving, processing, notifying, transmitting and storing data [5].
- **Platform** - Web services platform management is the management of the platform on which a web service is installed and provided. It is available as long as such a platform is with the standard management interface [117].

The non-functional aspects, expressed as *Contracts*, at the system level represent the behaviour of the workflow. Following, the properties we considered important for this level are defined.

- **Usability** - ISO defines usability as “*The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use*”. Ease-of-use requirements address the factors that constitute the capacity of the software to be understood, learned, and used by its intended users [54].

In the context of services, usability can mean how efficient and easy to access a service is and whether it satisfies the needs of those who are invoking this service.

- **Interoperability** - Is a property referring to the ability of diverse systems to work together. If two or more systems are capable of communicating and exchanging data, they are exhibiting syntactic interoperability [33]. According to ISO interoperability is defined as “*The capability to communicate, execute functions, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units*”. In the context of services, interoperability can mean that the services have capacity to communicate with each other.
- **Scalability** - Software that is scalable has the ability to handle a wide variety of system configuration sizes. The nonfunctional requirements should specify the ways in which the system may be expected to scale up [54].
- **Reliability** - The concept of reliability has different applications in different areas. Applied to web services, reliability is the ability of the service to execute correctly the methods it exports. It is also important that the data presented are consistent with the expected result. Reliability specifies the capability of the software to maintain its performance over time. Unreliable software fails frequently, and certain tasks are more sensitive to failure [54].
- **Availability** - A system’s availability, or uptime, is the amount of time that it is operational and available for use [54]. Is the proportion of time a service is in a functioning condition. This is often described as a mission capable rate. Availability of web services, is the probability of a service call to be successful, given a specific time.
- **Performance** - In the context of web services, performance is how fast the service can be executed in accordance with the needs of the user. The performance constraints specify the timing characteristics of the software [54].
- **Security** - Represents protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction [33].

The relationship between non-functional requirements and attributes help to identify groups of restrictions and contracts to develop specific policies. The level of service shows non-functional requirements, and system level presents a finer granularity, describing the non-functional attributes. Thus, a set of contracts that are related to the same attribute form specific

policies for the system. The relationship between the concepts in business, service and system levels are presented in figure 2.

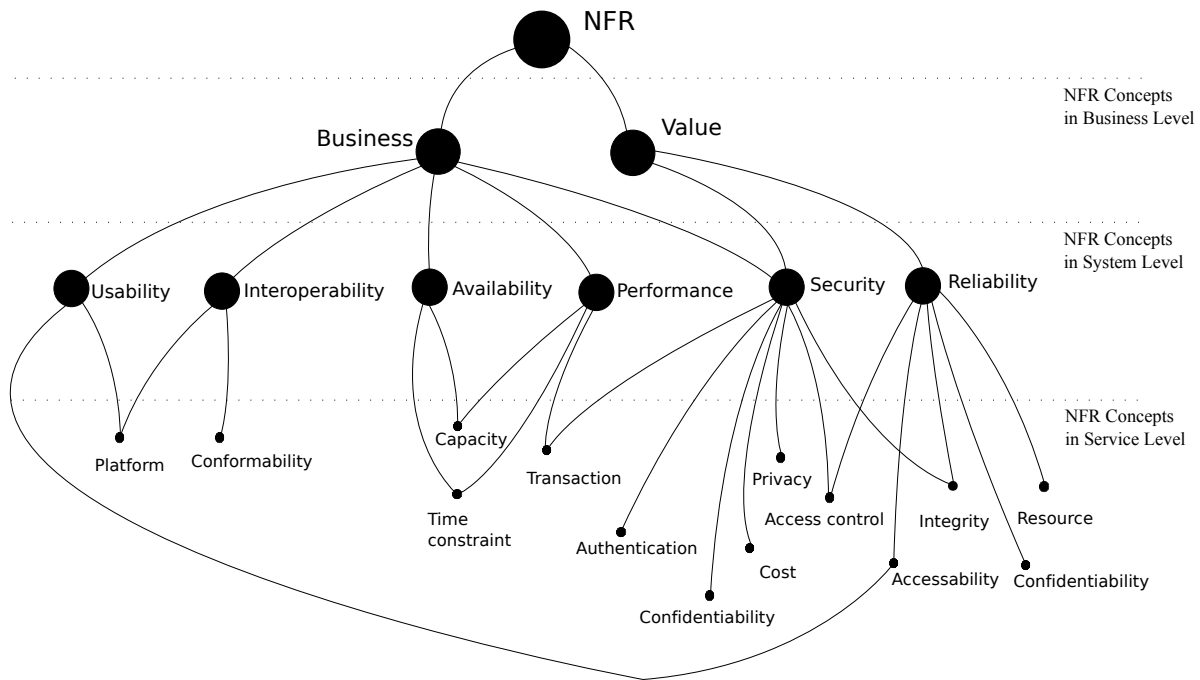


Figure 2: Relationship of the NFR Concepts.

In our work we adopt *security*, *performance*, *availability*, *interoperability*, *usability* and *reliability* as NFRs, and we consider as NFAs *access control*, *time constraint*, *privacy*, *accessibility*, *etc.* These terms and proposed values will be used in chapter 3. The methodology that will be presented in the next chapter uses this nomenclature for modeling the non-functional requirements in the development of service-oriented applications.

2.5 Conclusions

This chapter presented a review of related approaches for the service-oriented development of applications. We consider two important topics in our analysis: (i) non-functional requirements and (ii) methodologies for service-based development. The revised methods were grouped according to their characteristics, and analyzed their context of application.

We reviewed existing methodologies for service-based development. We analyze other general aspects of these methodologies as their development paradigm, use of the UML and formalism, and whether the method considered an MDA-based development or not. In this case we have studied the main techniques and methods proposed for modelling non-functional requirements, business process and web service composition.

To the best of our knowledge, there are no proposals that define a service-oriented approach for the whole development of systems, considering non-functional requirements. Although many efforts have been made to support the new technological proposals for the Web such as web services, in general, these methodologies focus their processes on traditional software engineering approaches, with emphasis on the functional aspects of the application.

The comparative study of such methodological approaches for service development is the basis for the design decisions of the method we propose in this work.

The analysis helped us identify a gap for improving the development of service-based applications. Our approach targets to deal with characteristics of the 3 areas⁷ in order to propose a methodology for developing service-based applications considering non-functional aspects.

Our work addresses the development of reliable service-based applications that uses services and non-functional aspects.

Similar to approaches such as WS-*, our work targets the specification and programming of service-oriented applications, including its non-functional aspects (i.e., atomicity, security, performance, persistence). Our work specifies *policies* for modeling the non-functional requirements of a service composition in an orthogonal way, *i.e.* separating the specification of non-functional requirements from the main functionalities of the application. This is interesting because once the policies for a given application have been defined they can be reused and/or specialized for another application with similar requirements.

The main goal of the methodology we propose in this thesis is to provide a way for supporting the development of service-oriented applications in the presence of non-functional properties. The use of the WS-* standards supposes that non-functional requirements are implemented according to the knowledge that a programmer has of a specific application requirements, without deriving them according to a methodology, thus leading to *ad-hoc* solutions that can be difficult to reuse. The goal is to provide a way for supporting the development of service-oriented applications in the presence of non-functional properties.

The methodology that will be presented provides a model-driven structure that encompasses concepts related with service-oriented development and non-functional properties (modelled as *Contracts* and *Policies*). The methodology provides a set of concepts, steps and transformations to develop service-oriented applications.

Given that our approach is MDA-based, it allows the specification of service compositions

⁷(i) Non-Functional Requirements; (ii) Methodology for Service-Based Development; and (iii) Model-Driven Engineering

as an abstract model, which may be realized as a concrete implementation (program).

In the following chapters we describe a service-based methodology that we propose and that focusses on non-functional aspects description and modeling. For this, we use model-based development (MDD) and the classification of non-functional requirements described in this chapter, as well as the meta-model with NFR notation.

3 *π SOD-M: A Methodology for Building Reliable Service Based Applications*

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”

C.A.R. Hoare

This chapter presents π SOD-M (*Policy-based Service Oriented Development Methodology*), a methodology for modeling and developing reliable web service based applications using a MDA approach. We propose guidelines for building service compositions that include non-functional requirements (NFRs). The methodology focuses on the development of service-based system, considering non-functional aspects of the software being developed. These aspects are implemented as policies for the application services. The design of this aspects integrates both functional and non-functional requirements.

This chapter is organized as follows. Section 3.1 introduces the concepts, general structure and motivations to define a specific methodology for policy-based web service development. We use a motivational example for illustrating the concepts and principle of π SOD-M. Sections 3.2, 3.3 and 3.4 describe the development models and transformations of our methodology. They also how to apply the methodology for building concrete applications. Section 3.5 concludes the chapter.

3.1 π SOD-M

π SOD-M is a MDA (Model Driven Architecture) based methodology. It provides an environment for building service compositions considering their non-functional requirements. π SOD-M extends the SOD-M [39] method by adding the concept of *Policy* [46, 47] for representing NFR associated to service-based applications. π SOD-M also proposes the generation of a set of models at different abstraction levels, as well as transformations between these models.

π SOD-M's models represent both the cross-cutting aspects of the application being modelled, as well as the constraints associated to services. The systems cross-cutting concerns affect functional concerns, such as availability; recovery; and persistence aspects. Constraints are restrictions that must be respected during the execution of the application, for example the fact that a service requires an authentication for executing system functions.

π SOD-M supports the construction of service-oriented applications that implement business processes. Therefore, it proposes a development process based on the definition of models (instances of the meta-modes) and transformations for semi-automatically generating different levels of abstraction that describe a service-oriented application from abstract platform independent views (CIM and PIM level) to platform dependent views and the PSM and implementation levels. We extended the Business and Service views of the SOD-M method [40]. The Business view defines the concepts for modeling a business process, while the Service view defines the concepts for designing services compositions. Our methodology introduces concepts (e.g. NF-requirement, constraint, assertion, contract, policy) in the Policy view for describing constraints associated to services and non-functional properties associated to service processes.

π SOD-M defines a service oriented approach providing a set of guidelines to build service based information systems (SIS) and proposes to use services as first-class objects for the whole system development process. π SOD-M extends the SOD-M models to include capabilities to model NFRs. The SOD-M models that are being extended are: *use case model*, *extended use case model*, *service process model* and *service composition model*. π SOD-M provides a conceptual structure to: (i) capture the system requirements and specification in high-level abstraction models (computation independent models, CIMs); (ii) obtain the PIMs from such models (specification documents). The platform independent models (PIMs) are designed to specify the system details; (iii) transform such models into platform specific models (PSMs)¹ that bundles the specification of the system with the details of the targeted platform; and (iv) serialize such model into the working-code that implements the system.

¹ π -PEWS language.

3.1.1 General Overview

Figure 3 presents the SOD-M models in the context of MDA architecture, and defines which models are extended by our approach. The π -*UseCase* model describes services requirements, constraints and quality requirements. The π -*ServiceProcess* model defines the concept of service contract for representing the service input and output data restrictions, as well as function restrictions. In this model we propose the definition of service contracts. The π -*ServiceProcess* model groups the constraints described in the π -*UseCase* model into contracts that are associated with services. The π -*ServiceComposition* model provides the concept of *Policy* that groups contracts with similar non-functional requirements. For example, security restrictions, such as contracts for authentication, privacy data access or transactions are grouped into a security policy.

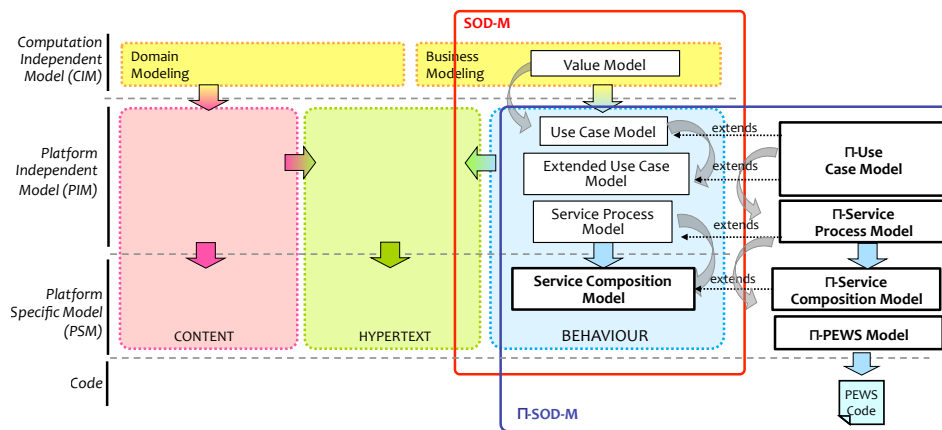


Figure 3: SOD-M and π SOD-M Extension Models.

In the π -*UseCase* model, use cases restrictions are designed as constraints. In the π -*ServiceProcess*, the constraints are grouped into service contracts, and use cases are refined to services or functions. Then, contracts are grouped into policies.

The π SOD-M methodology proposes the π -*PEWS* model as platform specific model. An instance of the π -*PEWS* model is generated from the π -*ServiceComposition* model, and it is a representation of a π -*PEWS* specification [13, 105]. From a π -*PEWS* model it is possible to generate a service composition code.

π SOD-M also defines model-to-model transformation rules (starting from the π -*UseCase* model) to π -*ServiceComposition* model; and uses model-to-text transformations to generate the corresponding implementation code in the π -*PEWS* language.

3.1.2 Development Process

π SOD-M uses the concept of MDA viewpoint, used as a technique of abstraction to focus on a particular aspect of the issue or proposed system. MDA defines, specifically, three viewpoints (figure 4):

- **Computation Independent Viewpoint:** This level focusses the environment of the system, as well as on its business and requirements specifications. At this moment of the development, the structure and system processing details are still unknown or undetermined. In π SOD-M, this level is represented as a list of business services from a requirements and business specification document.
- **Platform Independent Viewpoint:** This level focusses the system functionality, hiding the details of any particular platform. This specification defines those parts of the system that do not change from one platform to another. In π SOD-M, this level is modelled by the system use case, service process and service composition models.
- **Platform Specific Viewpoint:** This level focusses the functionality, without hiding the details of a particular platform, combining the platform independent view with the specific aspects of the platform to implement the system. In π SOD-M, the result of this level is the π PEWS specification [105] which represents as a platform specific model.

Computation Independent Models (CIM) aim to represent the business view, while Platform Independent Models (PIM) and Platform Specific Models (PSM) aim to represent the information system view and detail the information system to be implemented to fulfill the requirements of a business environment.

Figure 4 presents the π SOD-M development process, which defines a service oriented approach providing the guidelines for building service-based information systems (SIS), that was the result of the SOD-M extension described in figure 3.

Next section presents the concepts that lead to modeling applications in π SOD-M. These concepts form the basis of our methodology for modeling the reliable service-based applications.

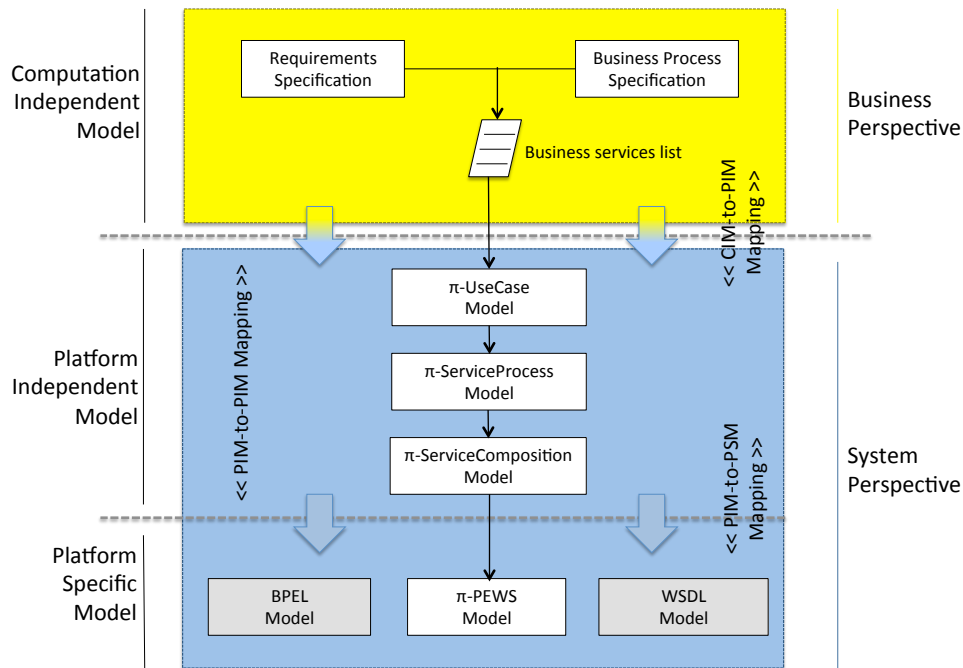


Figure 4: π SOD-M Development Process and Models.

3.1.3 Methodology Concepts

The concepts presented in figure 5 represent the main elements of the methodology used for system application modeling. The π SOD-M's meta-model² concepts are used in all methodology meta-models. They describe the key concepts that must be modelled in a service-oriented application. The π SOD-M meta-model consists of the set of concepts for modelling and development applications that use MDA. These concepts represent the reliable service-based system development, and are present in the modelling any application.

Notice that the three π SOD-M methodology views (*Business*, *System* and *Policy*) are different to those levels proposed by the traditional MDA literature: *CIM*, *PIM* and *PSM*. In the π SOD-M meta-model, at different MDA levels, there are elements from the π SOD-M concepts.

All the proposed concepts and models for business modeling (CIM level), and for information system and policy modeling at PIM and PSM levels comprise the backbone for the π SOD-M based applications design. These concepts are important to identify the system requirements and properties throughout the development. Thus, the purpose of this section is to present a description of each concept, so that when applied and used, it can be seen in their

²Meta-model is the construction of a collection of "concepts" (things, terms, etc.) within a certain domain. A model is an abstraction of phenomena in the real world; a meta-model is yet another abstraction, highlighting properties of the model itself. A model conforms to its meta-model in the way that a computer program conforms to the grammar of the programming language in which it is written.

cessing system.

- **Policy View:** focuses on non-functional requirements and business constraints of the information system. The concepts that correspond to the *Policy View* describe the NFRs and constraints related to the system functionalities. π SOD-M's main goal is to model non-functional requirements considering the models proposed by the SOD-M methodology.

The organization into different levels of abstraction of non-functional properties proposed by π -SOD-M relies on its concept of non-functional Requirement (NFR) defined as a group of semantically correlated non-functional attributes (NFA). For example, *security* is an NFR that comprises attributes such as *confidentiality* and *integrity*. NFR are used for modeling global constraints, NFAs model non-functional characteristics that can be precisely measured, such as *performance* [99]. This kind of characteristics are closer to implementation. Then, the concept non-functional property proposed by [99, 32, 33] is used for modeling different types of software constraints. Thus, it generalizes the concepts NFR and NFA of our methodology. In π SOD-M, NFR and NFA are modelled along the different modeling levels as CONSTRAINT TYPES that can be grouped and transformed into CONTRACT TYPES, that can be transformed into POLICY TYPES. A constraint type is used for representing a non-functional property.

The π SOD-M concepts³ are:

- **Business view:**
 - BUSINESS PROCESS - Represents the aggregation of logically related tasks that are carried out to achieve a given business result.
 - END CONSUMER* - Represents an entity that needs and consumes business services. End consumers are those who pay (either with money or using any other kind of value) to obtain a service that they use themselves.
 - BUSINESS SERVICE* - Represents a result of a business process (or part of it) providing value for an end consumer.
 - Business task* - Represents a business function performed by a business collaborator as a part of a business process.
 - BUSINESS COLLABORATOR* - Represents an entity that collaborates in the business processes of an organization, performing certain tasks needed to provide a business service.

³The concepts marked with * are also part of the **System view**

- **System view:**

- REQUIREMENT - Represents a super type for functional and non-functional requirements. Thus, the use cases can be related to both types of requirements.
- USE CASE - Represents a set of actions performed by the system to carry out part of a business service.
- COMPOSITE USE CASE - Represents a set of actions performed by the system to carry out part of a business service, which can be broken down into different use cases, which may in turn be basic or composite.
- SERVICE COMPOSITE PROCESS - Represents a set of logically related activities necessary for carrying out a business service.
- SERVICE ACTIVITY - Represents a behaviour (set of individual actions) forming part of the execution flow of a business service.
- ACTION - Represents a fundamental behaviour unit that is part of a service activity and describes some transformation or processing in the system being modelled.

- **Policy view:**

- NON-FUNCTIONAL (NF) ATTRIBUTE - An attribute that describes the quality or characteristics of a functional requirement. For example *confidentiality* and *privacy* is an example for a non-functional attribute for the functional requirement *user registration*.
- NON-FUNCTIONAL (NF) REQUIREMENT - A group of semantically correlated non-functional attributes (NFA). For example, *security* is an NF Requirement that comprises attributes such as *confidentiality* and *integrity*.
- CONSTRAINT - A constraint prevents the system from achieving more than its goal. With the definition of constraints, the system can be more robust, and unexpected problems can be solved before they happen. For example, in a banking system, the customer can only withdraw money if they have a positive balance in the account.
- CONSTRAINT TYPE - Represents the types of constraints, that may be on a system function or values. This entity can have their property setted as a business and data (value) constraint (expressed as an attribute).
- CONTRACT - Is the formalization of obligations (requires) and benefits (ensures) of a function, service activity or component. The following questions can be used to define contracts: *What does it expect? What does it guarantee?* Contracts are crucial

to software correctness and therefore they should be part of the design process. An interface is a kind of contract.

- ASSERTION - Represents a predicate or a state of the application before it runs (its preconditions), or the state when it is finished running (post-conditions);
- EXCEPTIONAL BEHAVIOUR - Are alternative execution paths if any condition or restriction is not respected. For example, if a user's password is not correct after three attempts, the user account is locked for security reasons.
- POLICY - A policy is a set of rules applied to a particular scope. This scope can be defined as an action, an activity, a function or a workflow. A policy is a composition of contracts applied to a non-functional application requirement. For example, a security policy of a system constraint may include authentication, access and data privacy properties.
- RULE - Represents information about an event, condition, and action where the event part represents the moment in which a constraint can be evaluated according to a condition represented by the condition part and the action to be executed for reinforcing it represented by the action part.
- VARIABLE - Represents a symbolic name given to some known information. A variable are related with a Policy. A Policy can have one or many variables. Each Variable has a name and a type.

The following next sections will describe the example scenario to better explain the π SOD-M meta-model and the methodology concepts.

3.1.4 Case Study

We use a case study in order to better describe specific details of the methodology proposed in this work.

Consider the following scenario: An organization wants to provide the service-based application "*To Publish Music*" that monitors the music a person is listening during some periods of time and sends the song title to this person's Twitter or Facebook accounts. More specifically, this social network user will have their status synchronized in both Twitter⁴ and Facebook⁵,

⁴Twitter is an online social networking service and micro-blogging service that enables its users to send and read text-based posts of up to 140 characters, known as "*tweets*".

⁵Facebook is a social networking service and website. Users may join common-interest user groups, organized by workplace, school or college, or other characteristics, and categorize their friends into lists

with the title of the music the user is listening in Spotify⁶. The user may also want to download music. For this, the user needs to process the purchase of music they want to download. Payments will be processed via PayPal or credit card. All services are available via Spotify and our application needs to interact with Spotify users, so they can listen music, publish on Facebook and Twitter, and purchase music online. The corresponding use case model is illustrated in in figure 6.

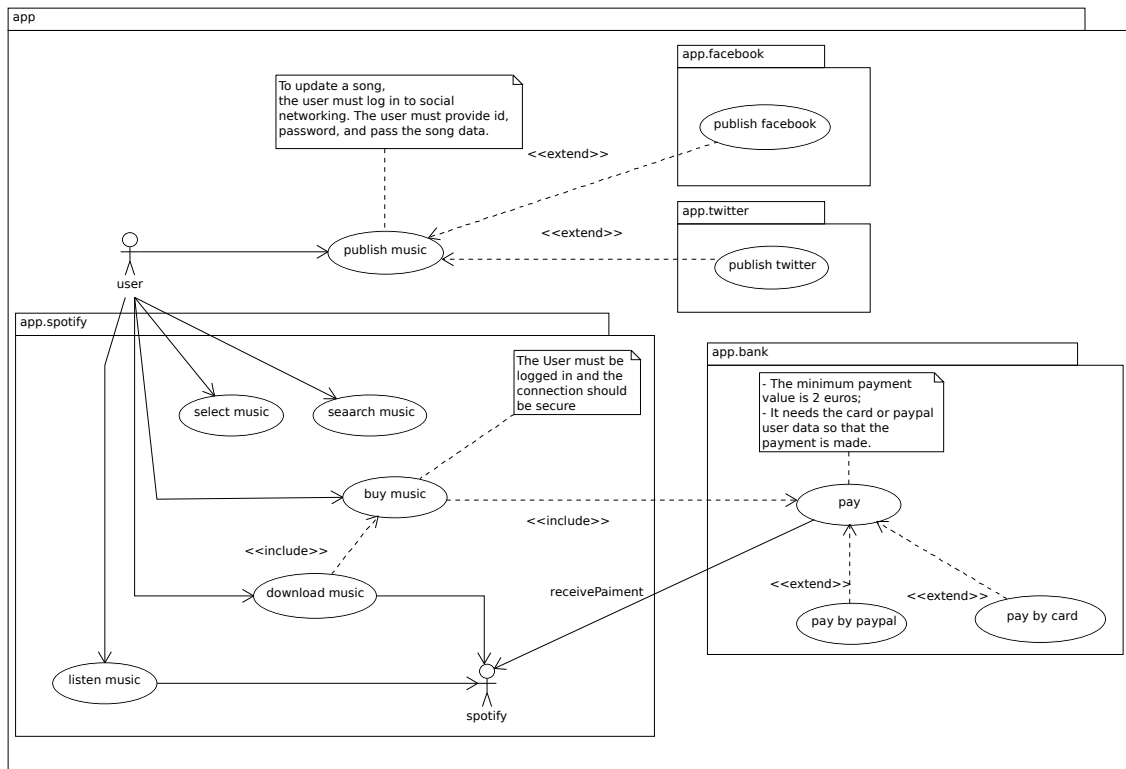


Figure 6: Use Case Model “To Publish Music” - Spotify Music Service.

Figure 7 shows the activity model for our scenario. It starts by contacting the music service Spotify for retrieving the user’s musical status. If the user wishes to download a particular song, they can download the song to purchase (activities *Buy Music* and *Download Music*). Finally the user can publish the music. The activity *publish music* can also be modelled as a parallel workflow for updating different social networks, as follows: Twitter and Facebook services can be contacted in parallel for updating the user’s status with the corresponding song title.

The execution flow is described as follows. Each action from the activity diagram represents one or more functions in the application. Each function or service has an interface, consisting of its input and output, and the specification of how to call this function/service. An activity must

⁶Spotify is a music streaming service offering digitally restricted streaming of selected music from a range of major and independent record labels. Spotify allows registered users to integrate their account with existing Facebook and Twitter accounts.

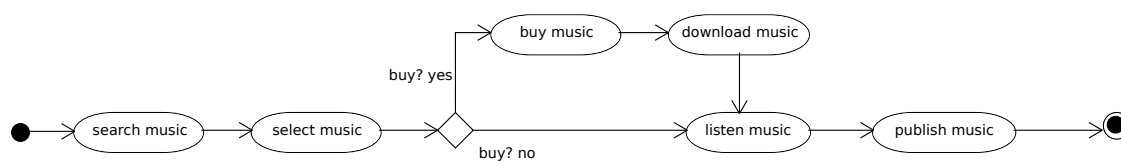


Figure 7: Service Process Model “To Publish Music”.

be executed according to the rules identified in the use case model. For example, the “*publish music*” action (figure 7), can be refined into two flows: (i) a flow to publish on Twitter and (ii) another to update the Facebook, since each service has different rules for update and login.

Given a set of services with their exported methods (known in advance or provided by a service provider), building service-based applications is a task that implies expressing an application logic as a service composition together with some restrictions.

Both models presented in figures 6 and 7 use the traditional UML to design requirements in use case diagram and workflow actions in the activity diagram. Figure 8 presents some of the main UML concepts that are used to model system diagrams. The structure of each π SOD-M model will be detailed in accordance with their respective meta-model and transformations necessary for achieving the aim of modeling reliable applications.

3.2 Platform Independent Models

The *Platform Independent Models (PIM)* are used for modeling the system functionality, abstracting technological details.

The models proposed at π SOD-M’s PIM level are (figure 4): π -*UseCase*, π -*ServiceProcess* and π -*ServiceComposition*. They provide concepts for expressing the design of: (i) system requirements and constraints; (ii) external services and their compositions; (iii) execution flow; (iv) input and output service restrictions; (v) service contracts and actions; and (vi) policies for the implementation of business services. We use these models to provide a representation of the system and to specify its restrictions.

3.2.1 π -UseCase Model

The π -*UseCase* (figure 13) is used to describe the functionalities (services and other functionalities) of the system. The services are represented in this model as use cases. We design this model by using the UML use cases, with a small extension for considering constraints.

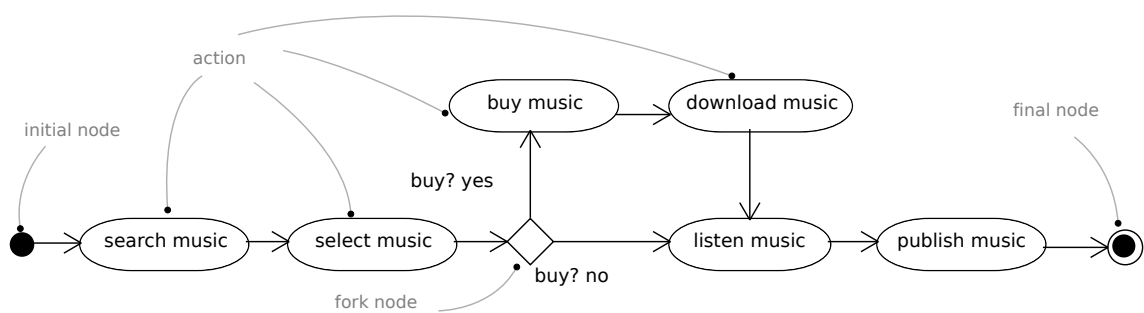
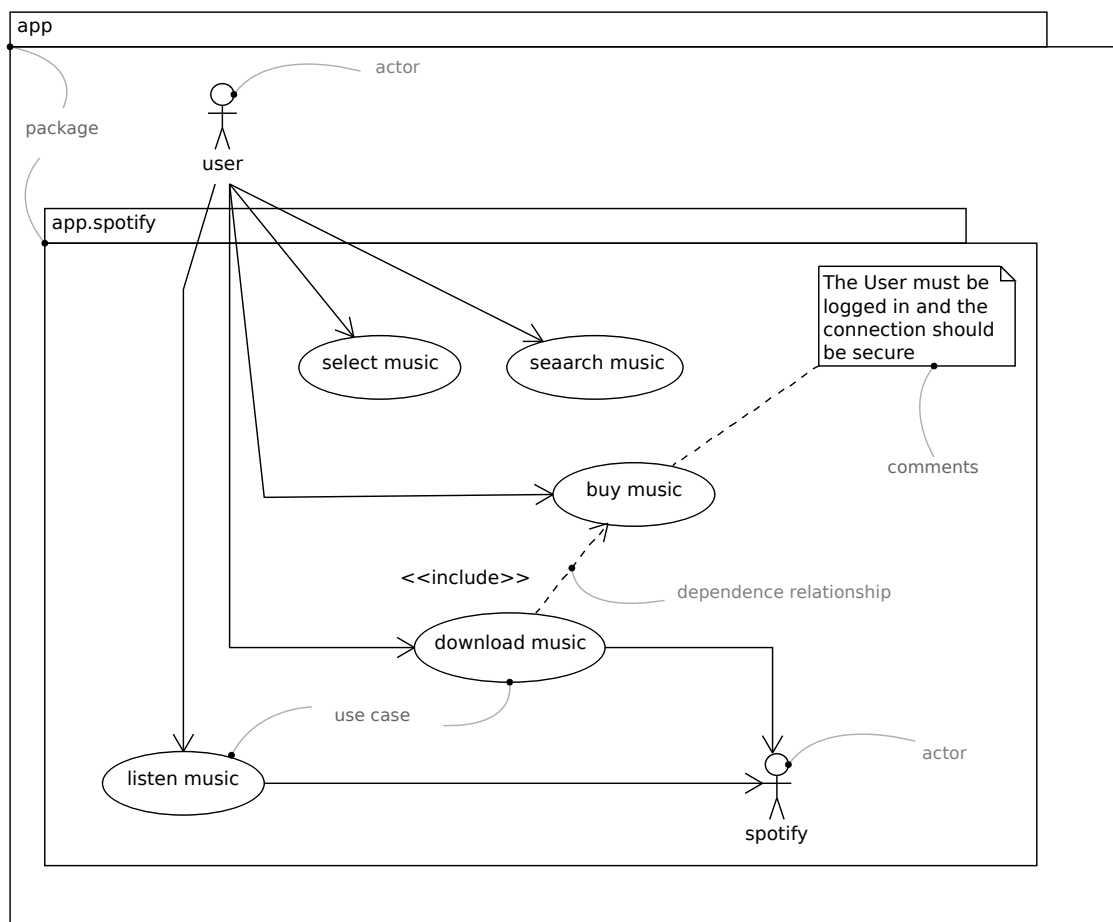


Figure 8: Traditional UML Concepts.

Application constraints are represented by stereotyped use cases and comments related to them. Each constraint describes what will be checked for a particular application function be executed. Figure 9 shows⁷ an example of how to describe constraints on use cases in π SOD-M. We use part of our case study to present them. Each use case may have multiple constraints. In this example the *publish music* use case has an authentication constraint (*user must authenticate*).

⁷The annotation in this figure are not part of the model. They are included for didactic explanation only. The same applies for the other figures in this section which present the same type of annotation.

For a song to be published on any social network it is necessary to provide data privacy. The constraint type concerns the authentication values and the constraint has an non-functional attributes. The use case is also related with requirement, a business service and a non-functional requirement properties.

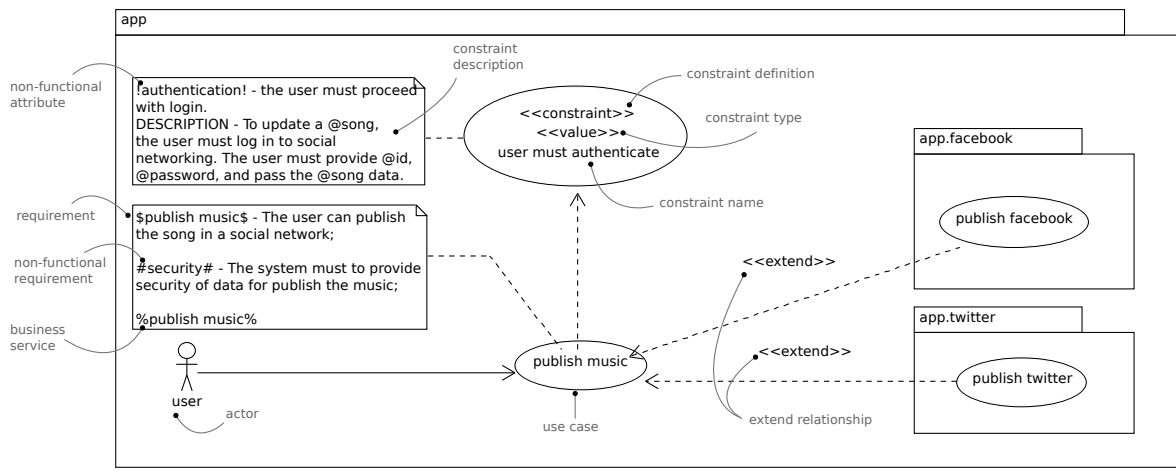


Figure 9: π -UseCase model Example.

3.2.1.1 π -UseCase Diagram, Terms and Concepts

The π -UseCase model describes the functional aspects of the application being developed and the restrictions that applies to each of them. These restrictions are modeled using the concepts: (i) constraint type and (ii) non-functional requirement with its specific attribute.

A π -UseCase model is a way to represent features of an application, services, constraints, and consumers of application functionality.

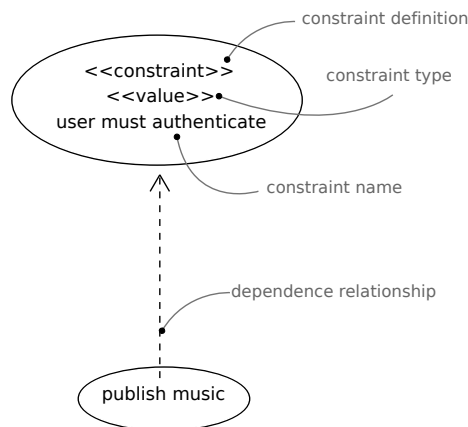


Figure 10: Constraint Representation.

All constraints are described by stereotyped use cases. Every relationship between an use

case and a constraint is done through a dependency relationship (from UML). All use cases that are related with constraint must be represented such as described in figure 10. A constraint is defined by a stereotype, containing its *type* and *name*. The CONSTRAINTS are represented by a stereotyped use case («constraint»).

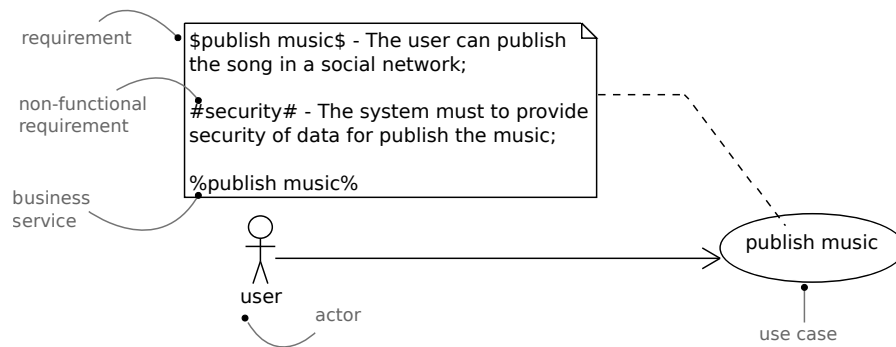


Figure 11: Non-functional Requirement Representation.

Each use case is related with one or more system requirements. Moreover, these use cases are related to non-functional requirements necessary for the proper execution of the application function. Similarly, the function described by an case of use can be part of a business service (in the case of service remote calls). These information to a specific use case may be described through comments as described in figure 11. All these information must be associated with the use case and its constraints.

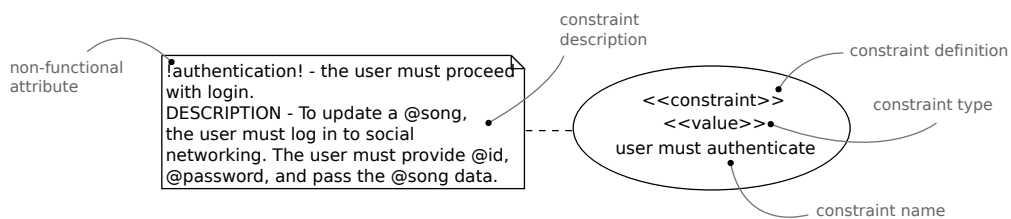


Figure 12: Non-functional Attribute Representation.

The constraints are always related to non-functional requirements. More specifically, each constraint describes a non-functional attribute, *e.g.* *authentication* (presented in figure 12). A constraint consists of the constraint description and the identification of variables be verified (by an @, for example @*song*), the identification of the non-functional attribute and its description.

These concepts that are described in the π SOD-M meta-model and used in π -UseCase represent the concepts we identified necessary for reliable service-based modeling.

3.2.1.2 Meta-model

The concepts modelled in the π -UseCase meta-model are: BUSINESS SERVICE, END CONSUMER, REQUIREMENT, USE CASE, COMPOSITE USE CASE, NON-FUNCTIONAL REQUIREMENT, NON-FUNCTIONAL ATTRIBUTE and CONSTRAINT. Figure 13 presents the π -UseCase meta-model and the relationship between each concept. We highlight the elements that represent the *policy view*.

In the π -UseCase an END CONSUMER is represented by an ACTOR (figure 13). An ACTOR is related to USE CASES (from the original UML definition), while a COMPOSITE USE CASE is a set of actions performed by the system which can be broken into different USE CASES.

BUSINESS SERVICE aggregates several USE CASES (figure 13), simply put, a service can be expressed by one or more use cases. Notice that not always an use case will be part of a business service. However, using our approach, we can identify in the early stages of modeling, which use cases are related to business services.

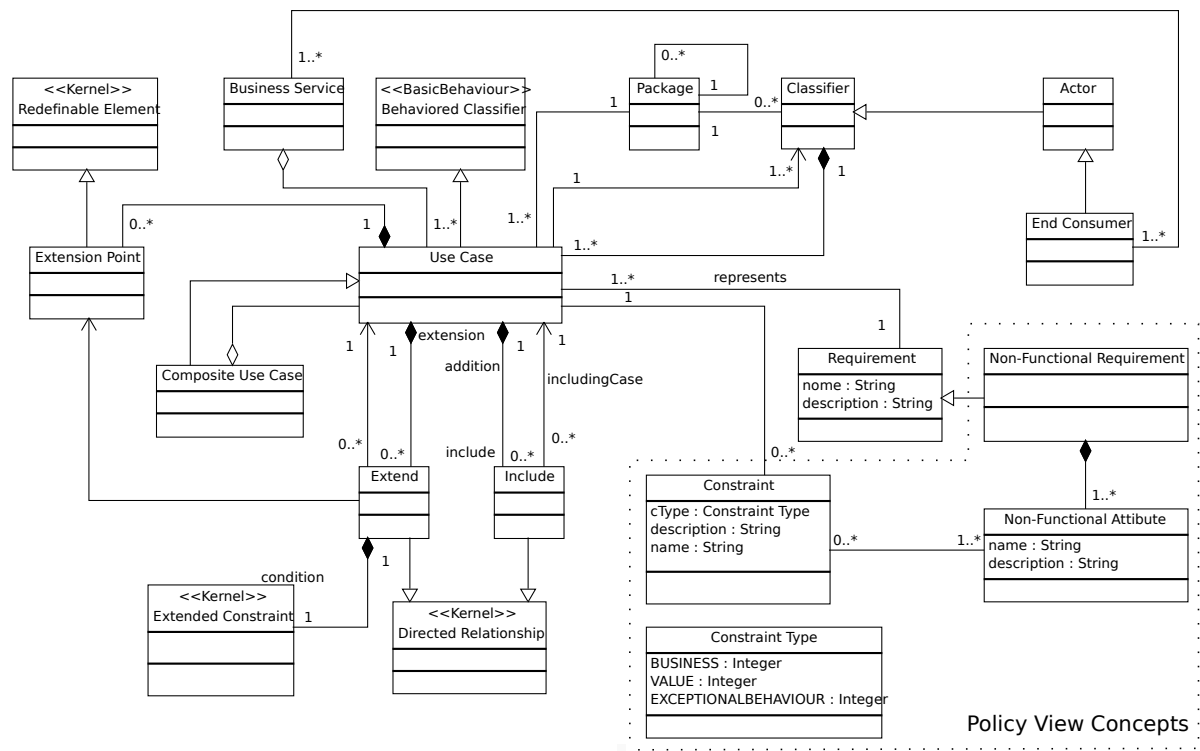


Figure 13: π -UseCase Concepts (Meta-model).

The BUSINESS COLLABORATORS concept are represented in the π -UseCase model through PACKAGES. A BUSINESS COLLABORATOR represents an external service or a system that interact with the application that are being modelled. Each BUSINESS COLLABORA-

TORS combines the features described in each PACKAGE. Thus, the services functions can be specified being grouped into packages (figure 14).

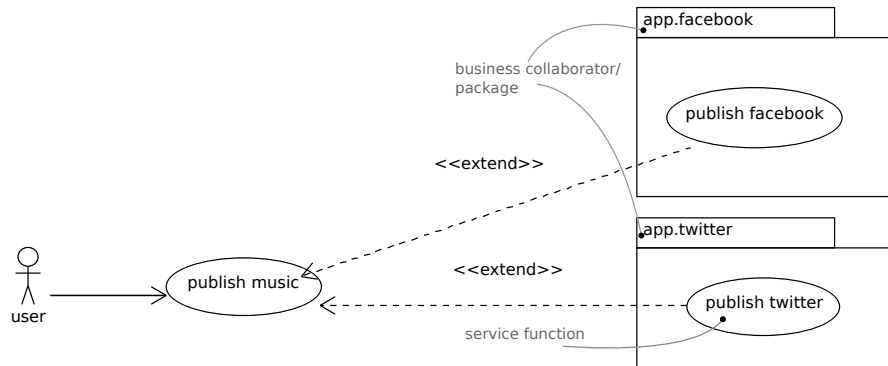


Figure 14: Business Collaborator / Package Representation.

The NON-FUNCTIONAL REQUIREMENT and NON-FUNCTIONAL ATTRIBUTE concepts are represented as part of description of USE CASES and CONSTRAINTS. For model this concepts the designer will use special tags to represent them (figures 11 and 12).

An USE CASE may have several CONSTRAINTS. Each CONSTRAINT has its name, description, and which ones should be checked during the execution of the application. Each CONSTRAINT is represented as a stereotyped (`<<constraint>>`) use case. A restriction may be a restriction on data (VALUE CONSTRAINT), represented as the stereotype `<<value>>` (figure 10); or on business rules (*Business Constraint*), represented as the stereotype `<<business>>`; and EXCEPTIONAL BEHAVIOUR constraints, which represents constraint violation for the use case execution, represented as the stereotype `<<exceptional_behaviour>>`.

The π -UseCase meta-model also represents the use of include and extend relations among the different use cases identified. The semantics are the same as in the traditional UML use case model. An include relation specifies the existence of a flow of events in which the base use case includes the behaviour of the other use case and an extend relation specifies that the behaviour of a base use case can be optionally extended by the behaviour of another use case.

3.2.1.3 UML Concepts Representation

Once described the meaning of each π -UseCase concept in the meta-model, it is important to know how specific models can be created from this general representation. Every π -UseCase model created to describe application features must follow the meta-model concepts. The π -UseCase meta-model describes what can be specified in each model.

The π -UseCase concepts are modelled as described in figure 15 and the relationship be-

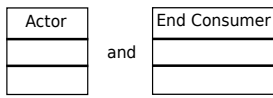


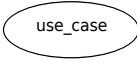

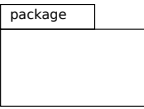

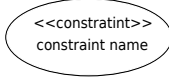
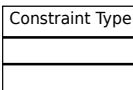
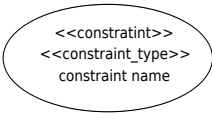
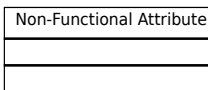

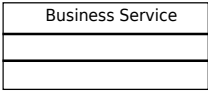
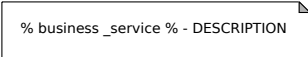
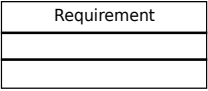
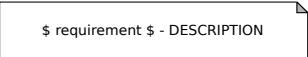
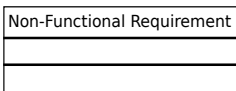
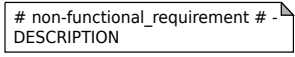
Meta-model Concept	UML Representation
 <p>Actor and End Consumer</p>	 <p>actor</p>
 <p>Use Case</p>	 <p>use_case</p>
 <p>Package</p>	 <p>package</p>
 <p>Constraint</p>	 <p><<constraint>> constraint name</p>
 <p>Constraint Type</p>	 <p><<constraint>> <<constraint_type>> constraint name</p>
 <p>Non-Functional Attribute</p>	 <p>! non-functional-attribute ! - DESCRIPTION</p>
 <p>Business Service</p>	 <p>% business_service % - DESCRIPTION</p>
 <p>Requirement</p>	 <p>\$ requirement \$ - DESCRIPTION</p>
 <p>Non-Functional Requirement</p>	 <p># non-functional_requirement # - DESCRIPTION</p>

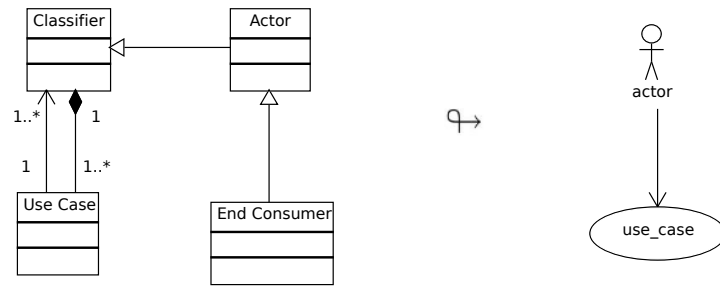
Figure 15: π -UseCase Concepts Represented as UML Elements.

tween these concepts is also applied to the UML elements. Figures 16 and 17 detail how to represent the relationship between the meta-model concepts in a π -UseCase model.

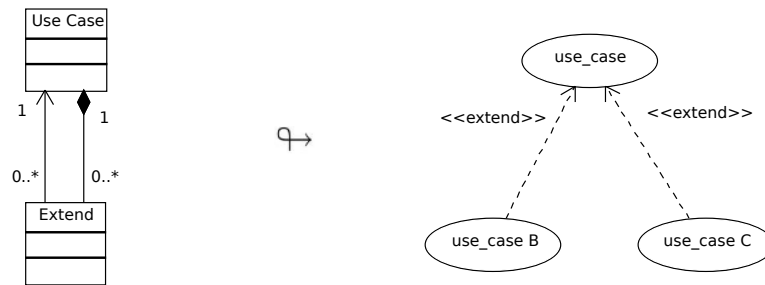
The *loop arrow* symbol (\looparrowright) is used in figures 16 and 17 to relate the concepts of the meta-model and the UML elements used in the representation of the concepts.

This means how the π -UseCase meta-model concepts (left side) can be modelled as an UML model (right side).

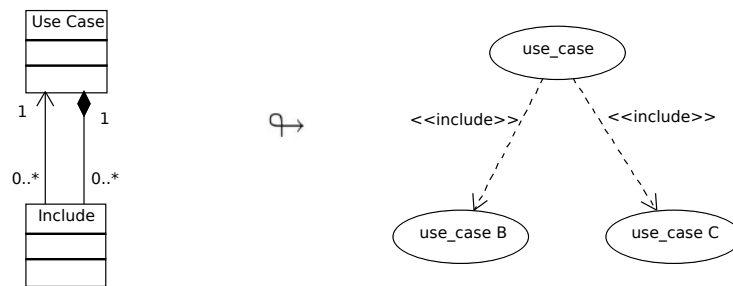
An ACTOR is an abstraction of the END COSTUMER concept, presented in π SOD-M meta-model. Thus, it is possible to associate several ACTORS with an USE CASE and many USE CASES with an ACTOR (END COSTUMER). The original use case meta-model defines that



(a) Use Case and Actor Model



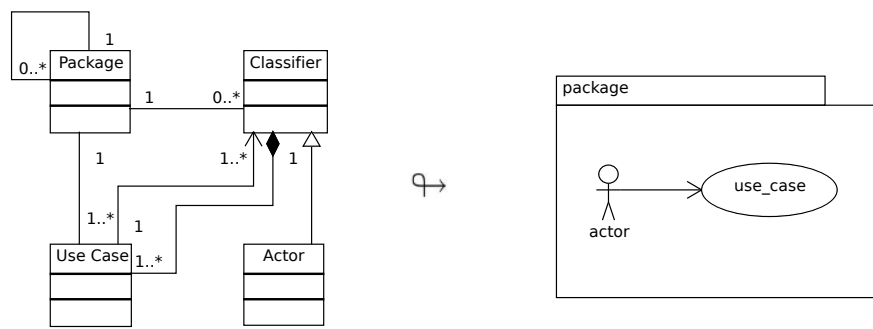
(b) Extended Use Case Model



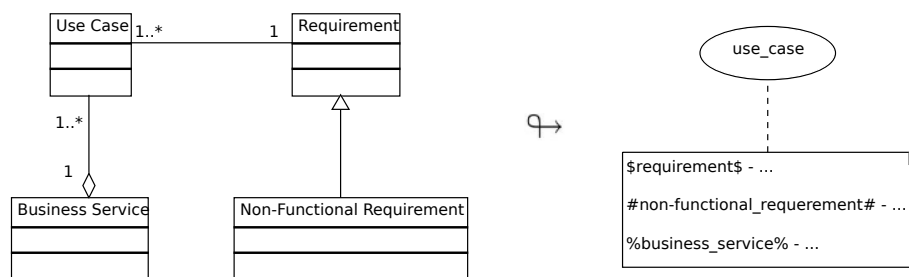
(c) Include Use Case Model

Figure 16: π -UseCase Model Representation.

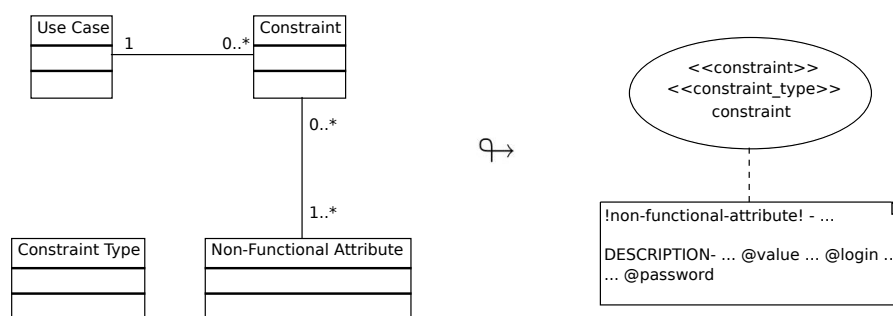
CLASSIFIER is a superclass of ACTOR. A CLASSIFIER is a category of UML elements that have some common features, such as *attributes* or *methods*. The relationship between these entities is many to many (figure 16a). An USE CASE can also be related with an EXTENDS or an INCLUDE entity in the π -UseCase model, and this model preserve the original UML standard semantic. EXTENDS and INCLUDE are types of stereotyped dependence relationships (<<extend>> and



(a) Package Model



(b) Use Case, Requirement and Non-Functional Requirement Model



(c) Constraint Model

Figure 17: π -UseCase Model Representation (2).

«include» in figures 16b and 16c, respectively). The EXTENDS relationship can modify the behaviour of the base use case. Suppose someone wants to close a bank account, however still have some money there. Before closing the account, there should be a withdrawal or transfer of money in this account, so that the balance is zero. This process is modelled with a EXTENDS dependence between the *close account* use case and the *withdrawal money* and *transfer money*

use cases. The INCLUDE relationship includes the steps from one use case into another.

A PACKAGE clause and its relationship with USE CASE and ACTOR can be modelled in π -UseCase model. The original semantics was preserved and it is possible have different levels of packages, due to PACKAGE entity auto reference, one to many. Each package can have many use cases and actors (17a).

The REQUIREMENTS, NON-FUNCTIONAL REQUIREMENTS and BUSINESS SERVICES must be associated with a specific USE CASE. At this level of modelling, these information are represented as comment tags, they are: \$... \$ for REQUIREMENTS, # ... # for NON-FUNCTIONAL REQUIREMENTS, and % ... % for BUSINESS SERVICE. A BUSINESS SERVICE is a aggregation of USE CASE, and each USE CASE must be related with a system requirement REQUIREMENTS. Figure 17b presents the relationship of this concepts, and how represent this information in UML.

All USE CASES may also be associated with a set of CONSTRAINTS. A CONSTRAINT is a stereotyped use case that describes the CONSTRAINT TYPE and its restriction. Additional information is given by comment tags, ! ... ! for NON-FUNCTIONAL ATTRIBUTE, and @ ... @ for candidate variables or values that must be checked during the system execution, as presented in figure 17c. The relationship between USE CASES and CONSTRAINT is made by a non-stereotyped dependence arrow.

3.2.1.4 To Publish Music Use Case

Considering the example scenario, the “To Publish Music” use case can update the Facebook or Twitter user status. Therefore, it is necessary to perform a social network authentication with the user’s data. Each social network uses different services and different forms of authentication. The *authentication* constraint is required to update a music status. The restriction is stereotyped as a «value» constraint, because the *user’s id* and *password* are verified. The NON-FUNCTIONAL REQUIREMENT that is being modelled for this use case is *Security*, because, considering table 4, “*authentication*” is a NON-FUNCTIONAL ATTRIBUTE of *Security*. These information will be refined and detailed in the following phases.

Figure 18 shows the model represented in figure 6 for the *buy music*, *download music*, *listen music* and *pay* use cases. The model was designed considering the meta-model representation detailed in figures 16 and 17, and also the π -UseCase meta-model. Notice that from π -UseCase concepts is possible to describe the actors, use cases, constraints, non-functional requirements and their attributes. For example, the *pay* use case (figure 18), there are extended use cases,

constraints, packages, non-functional requirements and non-functional attributes related to this function. The modeling was based on the described representation of the π -UseCase meta-model. The extended use case representation presented in figure 16b is used in the model to express the *pay by PayPal* and *pay by card* extended use cases. The include use case representation presented in figure 16c is used in the model to express the included relation of *buy music* and *pay* use cases. The package representation presented in figure 17a is expressed in the example by the *app.bank* package and the *pay* use case relation. The representation described in figures 17b and 17c is expressed by the *pay* use case properties' description. The combination of a π -UseCase meta-model's entities means that all entities presented in figures 9 and 18 can be modelled to express the quality requirements.

The use cases that have restrictions are *buy music* and *pay*. The process of buying a song requires the user's private data for a Spotify account, and also a secure connection, represented as «value» and «business» constraint, respectively. For payment, the user must provide the data of payment card or PayPal account login and password, represented as «value» stereotype. Other restriction is that the minimum payment value is 2 euros. The NON-FUNCTIONAL REQUIREMENT that are being modelled for this use cases are *Security* and *Reliability*. The specific NON-FUNCTIONAL ATTRIBUTES for this restriction is *Transaction* and *Confidentiality* (see figure 18).

After modeling the system services and features, and its restrictions, it is necessary to model the services process and its execution restrictions. For this, the methodology uses a service process diagram to describe the services execution and contracts. Next we present the π -ServiceProcess model. This model refines the application constraints through the workflow model (as presented in Figure 4).

3.2.2 π -ServiceProcess Model

A *Service Process* is a collection of related, structured activities that produce a specific service or product for a particular customer. It can be visualized with a flowchart as a sequence of activities with interleaving decision points. A set of linked activities that take an input and transform it to create an output. Ideally, the transformation that occurs in the process should add value to the input and create an output that is more useful and effective. To represent these properties our methodology proposes the π -ServiceProcess model.

The non-functional requirements are specified in this model using the concepts of contracts and assertions over functions. These contracts may represent the control of privacy information,

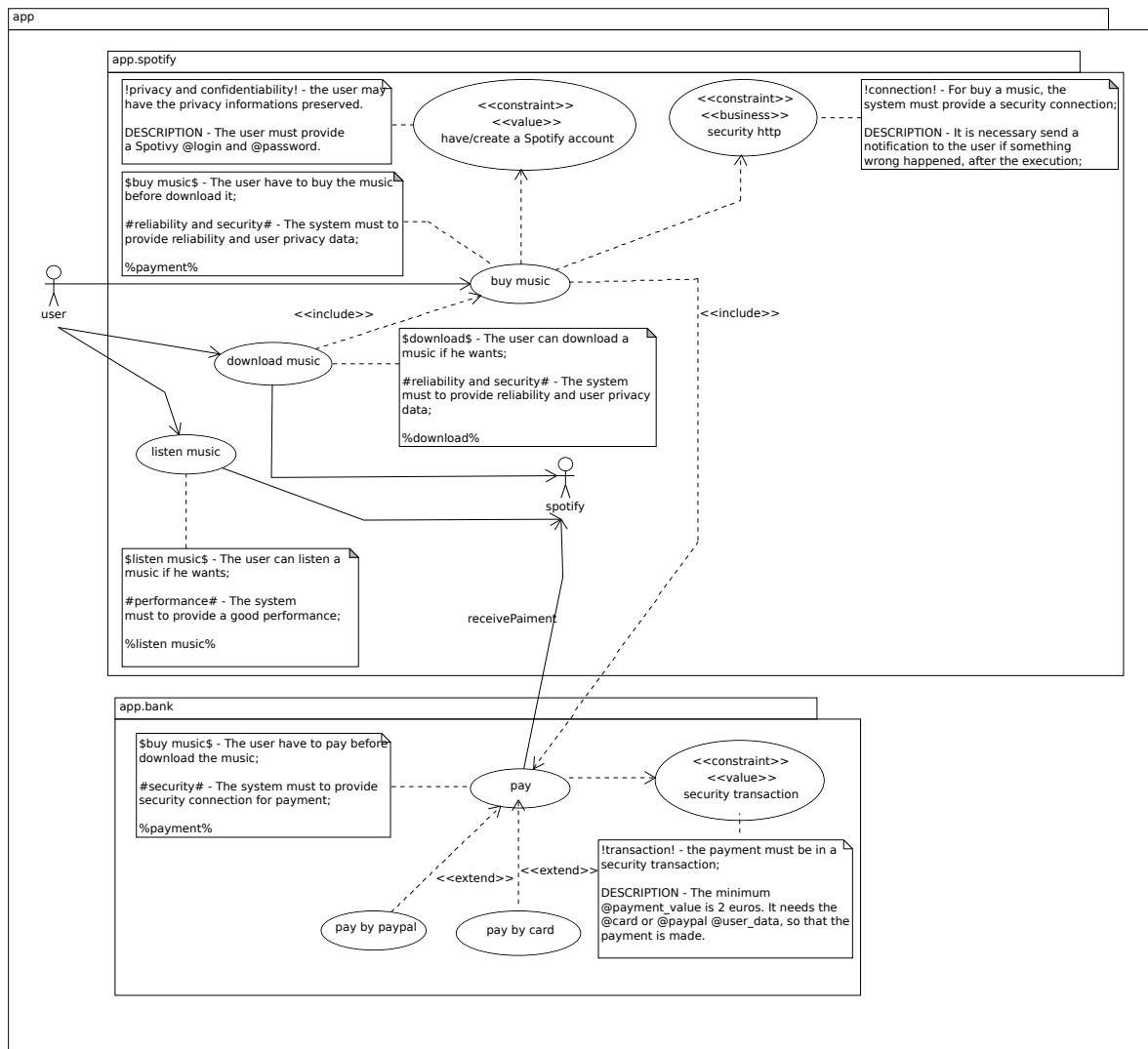


Figure 18: Use Case Model With Constraint Definition - Scenario Example Refined.

performance, reliability, and so on.

3.2.2.1 π -ServiceProcess Diagram, Terms and Concepts

Systems that use third party services add value to the application because the services are already available and validated. If the services do not offer any quality warranty, these quality requirements are the responsibility of the application designer. Thus, the contracts of services will be modeled as a wrapper which describes restrictions on input and output information on particular functions or a set of them.

π SOD-M proposes modeling *Service Process* diagram (π -ServiceProcess model), that is defined considering the functional and non-functional requirements designed in a π -UseCase model.

The π -ServiceProcess model represents the service processes and its contracts. It models the set of system activities and describe contracts for each activity or the process as a whole. Each service identified in the previous model (π -UseCase) is mapped to an action.

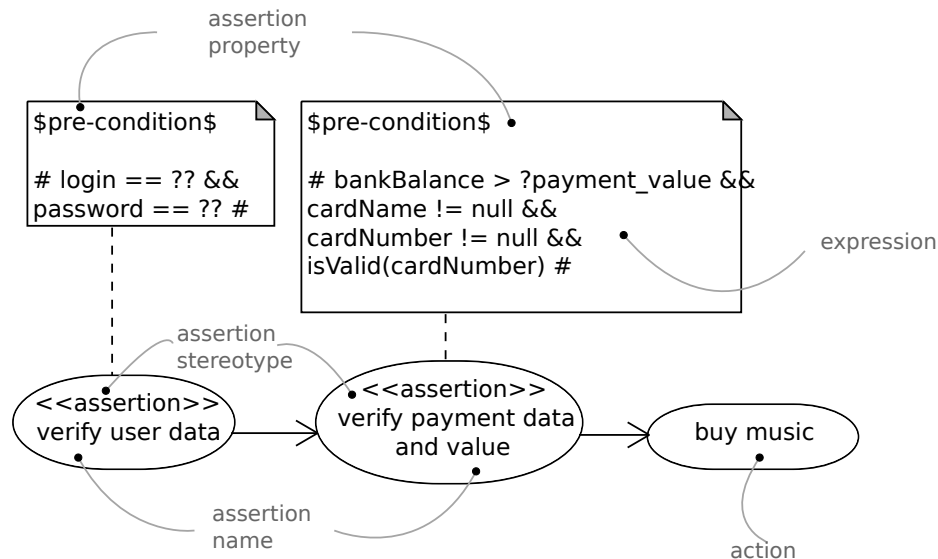


Figure 19: Assertion Representation.

A π -ServiceProcess model represents the system process, actions, assertions and contracts. This model describes the workflow for the modelled application behaviour. The behaviour is represented by each action and the restrictions by the assertion description. All assertions are described by stereotyped actions.

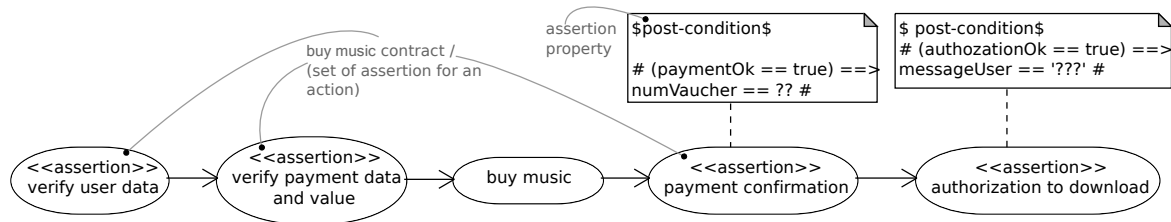


Figure 20: Contract Representation.

Figures 19 and 20 present the assertion and contract representation in this model. An Assertion is represented in UML, and a set of assertions forms a contract. Assertions describe restrictions on a single action, but actions may have different restrictions. Figure 19 shows how an assertion is modelled, with their property type, name and acceptable values (expressed through expressions), while figure 20 shows how this set of assertions form a single action contract.

Figure 21 shows the representation of the general concepts of this model, nodes, activity

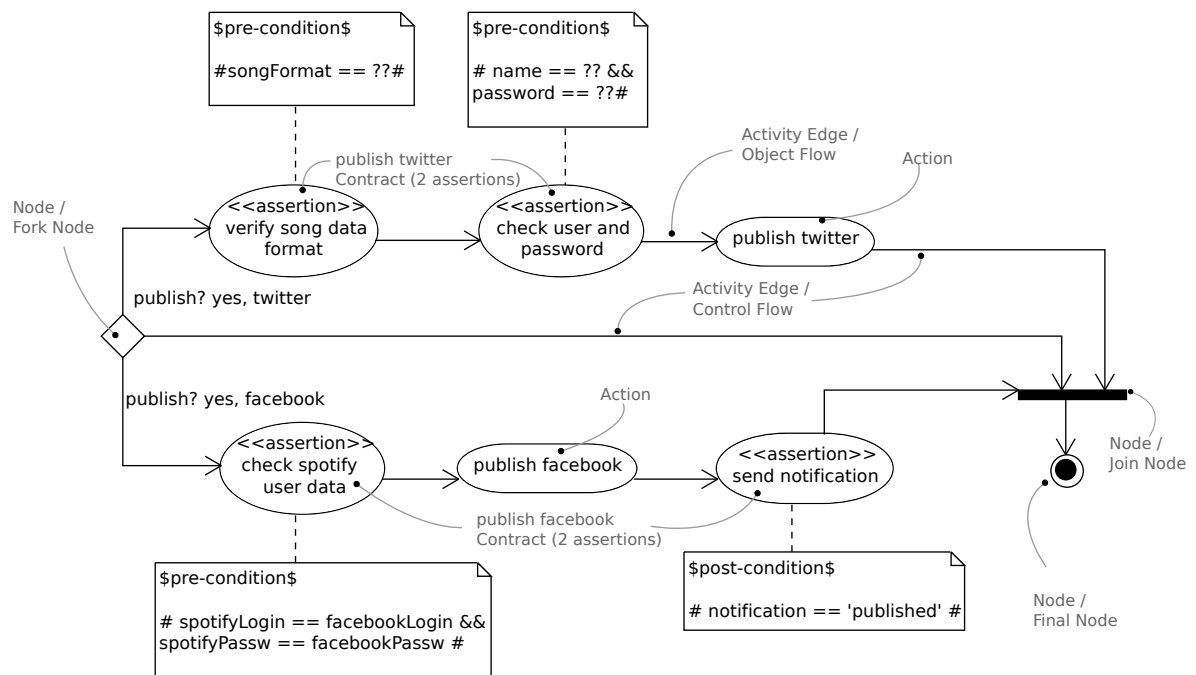


Figure 21: Node, Activity Edge and Contract Representation.

edges, actions and contracts. This figure also refines the information for publish a song in a social network.

3.2.2.2 Meta-model

Considering the π SOD-M concepts (figure 5), the π -ServiceProcess meta-model (figure 22) describes the system functionalities to be modelled in a contract based service process. The π SOD-M concepts modelled in the π -ServiceProcess meta-model are: CONTRACT, ASSERTION, EXCEPTIONAL BEHAVIOUR, ACTIVITY, SERVICE ACTIVITY, ACTION and CONSTRAINT.

Restrictions on activities and services are not described in the original SOD-M proposal. The concepts of CONTRACT, ASSERTION and EXCEPTIONAL BEHAVIOUR make it possible to model restrictions on services execution, ensuring greater reliability application. Entities highlighted in π -ServiceProcess meta-model (figure 22) are the difference from the original SOD-M service process model.

A specific π -ServiceProcess model shows the set of logically related activities that need to be performed in a service-based system. So, the activities of this model represent a behaviour that is part of the system workflow. π SOD-M proposes representing this model using the UML activity diagram, with an extension for CONTRACT design, which encapsulates sev-

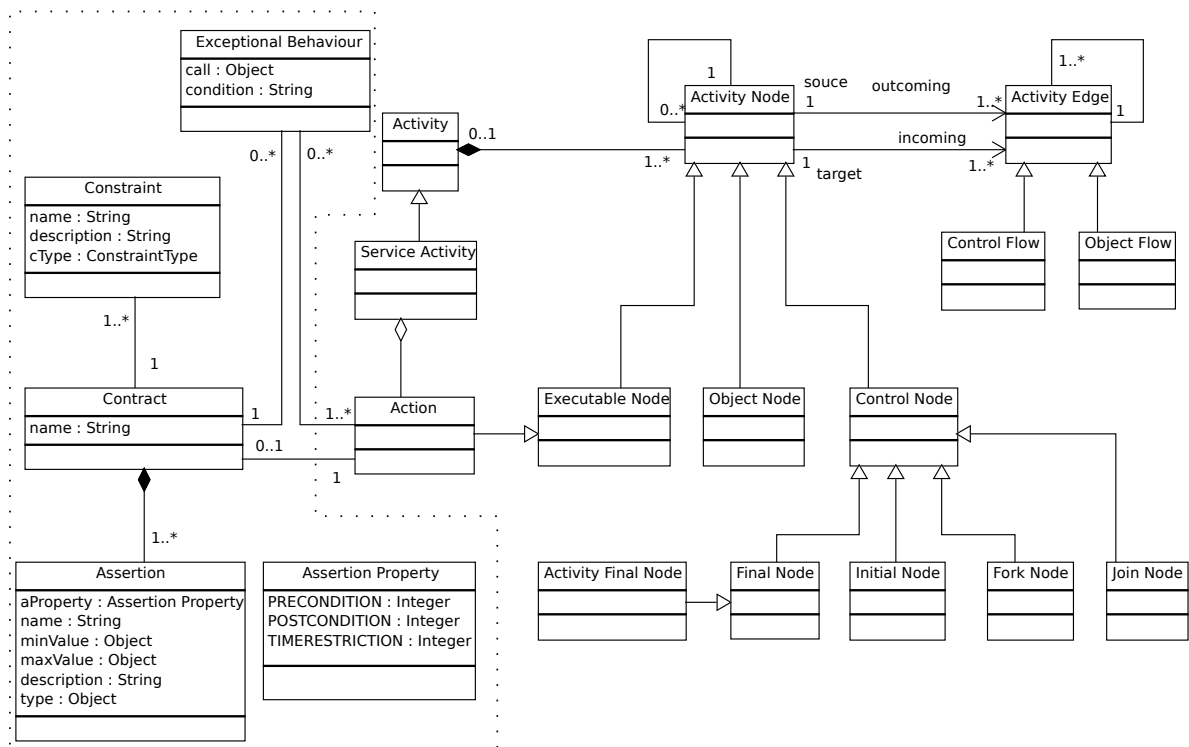
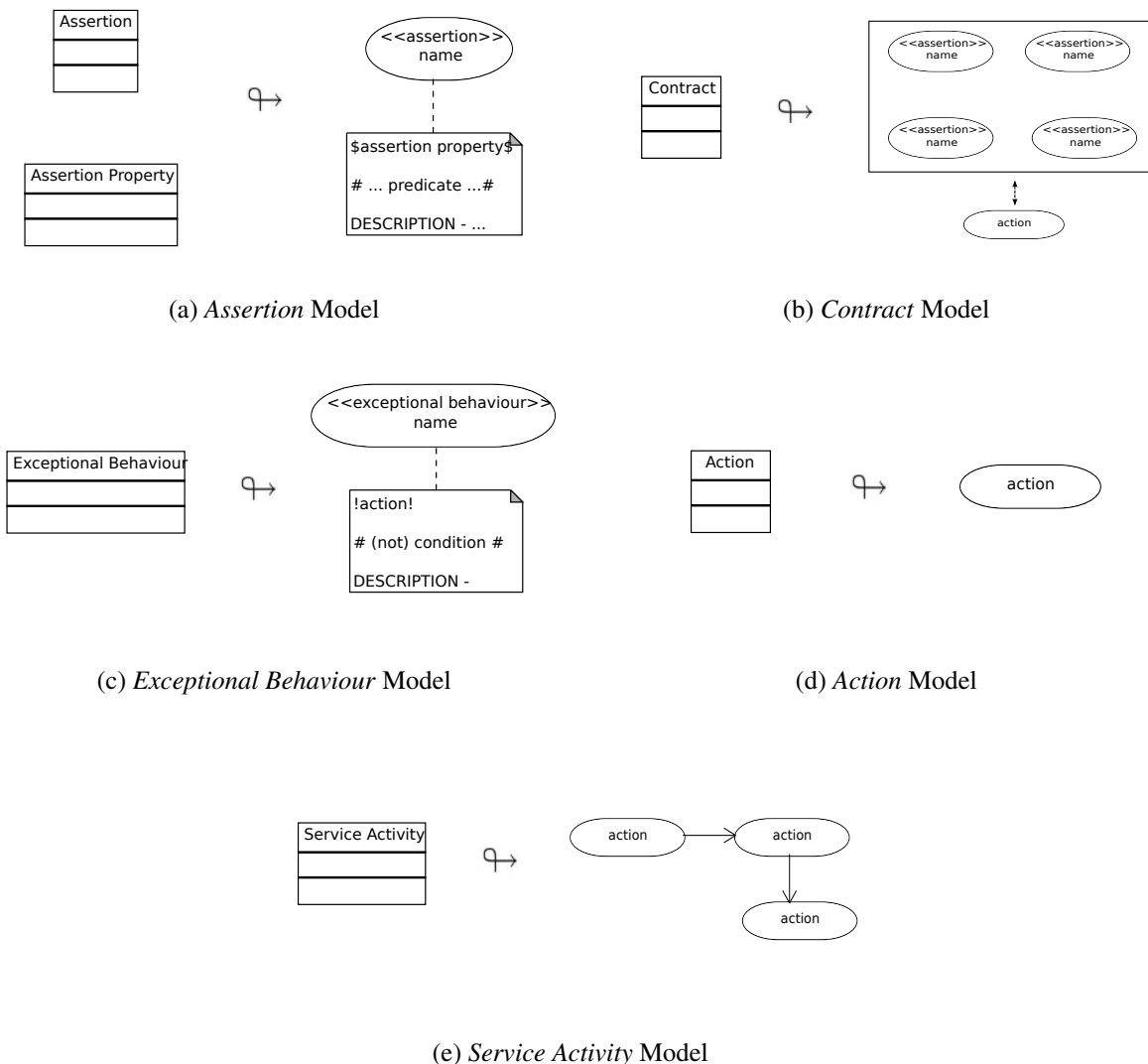


Figure 22: π -ServiceProcess Concepts (Meta-model).

eral CONSTRAINTS modelled in π -UseCase model. In π -ServiceProcess, three main features are represented: (i) *service process*, (ii) *service activity* and (iii) *activity contract*. The service process is represented by the model itself. A service activity represents a behaviour that is part of the execution flow, and is identified in this model as an ACTION. A activity contract represents the NON-FUNCTIONAL REQUIREMENT behaviour that is also part of the execution flow of a service, and is identified in this model as an stereotyped activity (`«assertion»`). All ASSERTION related with an ACTION compose a CONTRACT. From the CONTRACT and ASSERTION concepts is possible to define the interface specifications for each activity service, such as pre-conditions and post-conditions.

3.2.2.3 UML Concepts Representation

To formalize the concept of CONTRACT by rules, we use the definition of ASSERTION (figure 23a). An ASSERTION is represented by a stereotyped action (`«assertion»`), and ASSERTION PROPERTIES through comments. An ASSERTION may be a *pre-condition*, *post-condition* or a *time restriction* over the service execution (ASSERTION PROPERTY). These information are represented as the comment tags \$...\$. The assertion predicate is modelled as #...# tag. Figure 23a shows how these concepts are represented in UML.

Figure 23: π -ServiceProcess Model Representation.

A set of ASSERTION composes a service CONTRACT related with an ACTION (figure 23b). An Assertion describes a value (*value constraint* from π -UseCase model) restriction, setting maximum and minimum limits, or business restrictions, setting conditions for function calls (*business constraint* from π -UseCase model)). The business restrictions are defined over service functions. Every service contract may have an exception behaviour, if a constraint is not obeyed. For example, if the user card information are incorrect, the service is called again until the limit of 3 calls, otherwise, execution continues normally. In this case, the exception is the new call to the same service. For this, we also use the concept of EXCEPTIONAL BEHAVIOUR (figure 23c). A EXCEPTIONAL BEHAVIOUR is also a stereotyped action («exceptional behaviour») and its properties are modelled through comment tags. The condition to be verified is modelled as # ... # tag. Figure 23a shows how these concepts are represented in UML. If the condition is *false*, the predicate becomes true, because of the expression negation

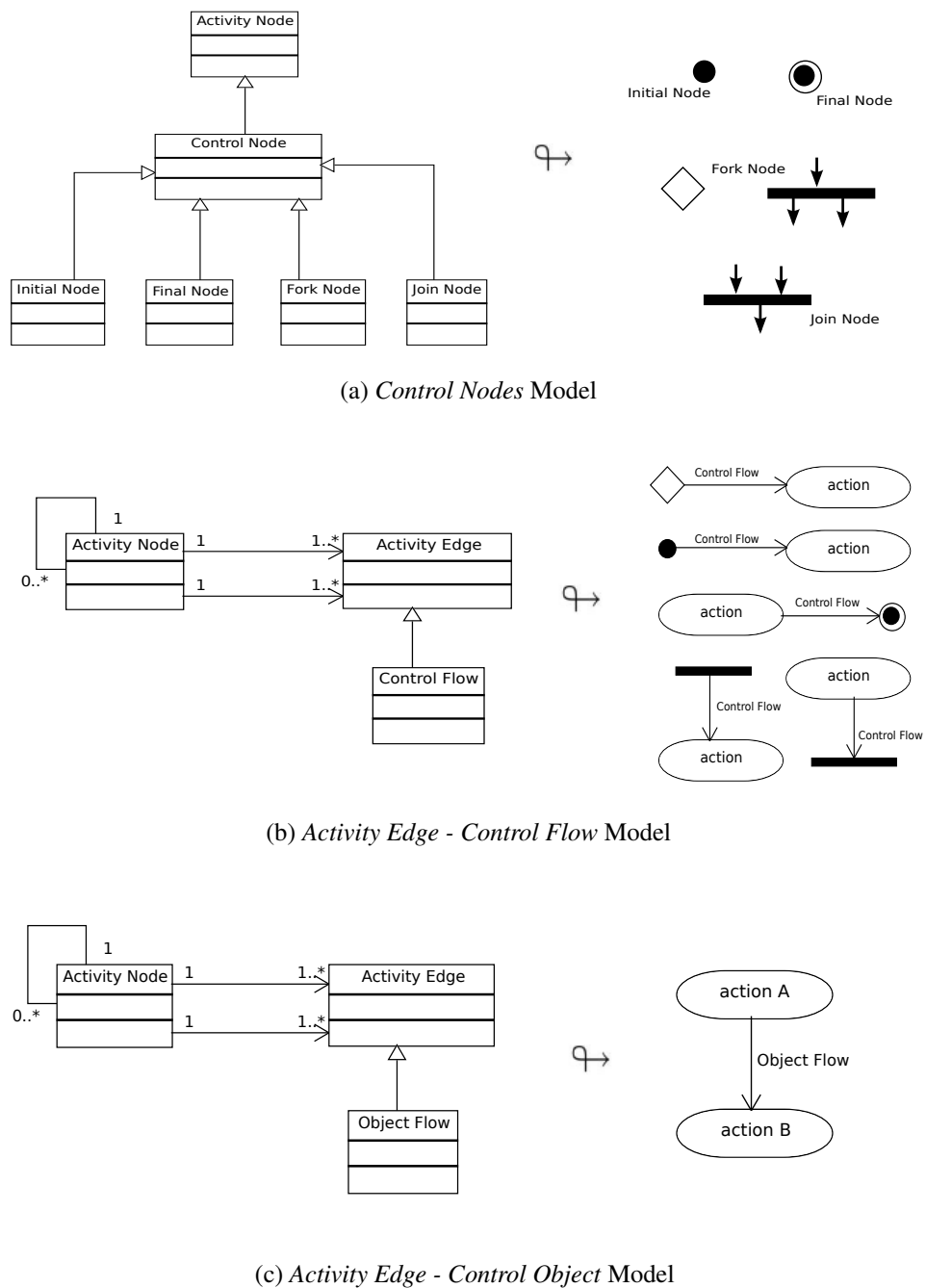


Figure 24: π -ServiceProcess Model Representation (2).

(not). Thus the action is performed. The action to be performed is modelled as ! ...! tag. The ACTION concept is modelled in UML as the action component (figure 23d), and a SERVICE ACTIVITY represents a set of ACTION (figure 23e). We highlight the models in figures 23a, 23b and 23c, because they present how an assertion action, a contract, and a exceptional behaviour can be designed.

Figure24 shows the UML representation of the CONTROL NODES and ACTIVITY EDGES concepts in π -ServiceProcess meta-model. This representation follows the original UML spec-

ification. Figure 24a presents the CONTROL NODES and how its can be applied in a real modelling, and finally figures 24b and 24c show the ACTIVITY EDGES and its application.

3.2.2.4 To Publish Music Process

Considering the example scenario, the contract based process of activities execution is shown in figure 25. The activities flow in figure 7 presents any restriction. Thus, it is necessary to represent each CONSTRAINT modelled in a more representative CONTRACT based service activity diagram.

Based on the concepts described and its representation in UML, we describe how to model a service process using the assertions as restrictions for each action. The difference of the model shown in figure 7 and the figure 25 is that restrictions are added on the actions in order to describe quality requirements.

Each USE CASE modelled in the use case diagram are refined into ACTIONS in the service process diagram. The set of CONSTRAINTS are transformed into ASSERTIONS and CONTRACTS. The *buy music* and *publish music* services (update Twitter and Facebook) has pre- and post-conditions assertions that are composed into a contract for each service. The *buy music* pre-conditions are: (i) verify if the User data are correct; (ii) if the User is already logged in Spotify; (iii) if bank account information are correct and; (iv) if there is enough money to make the payment. As post-condition, it ensures the complete transaction and verify if a notification were sent to the user and Spotify, about the payment authorization. There are four assertions for the *buy music* action, and each assertion has been detailed with the assertion property and predicate that must be verified. To *update* services, depending of each service, there may be different restrictions. As an example, a new verification of user data and message format is appropriate (maximum 140 characters), in case of Twitter. In the case of Facebook, it is required that the user is already logged on Spotify and these data are the same as Facebook. As post-condition, it ensures that the Facebook service send a notification of success. To update Twitter a pre-condition is required, while to update Facebook is necessary a pre-condition and a confirmation notice is modelled as post-condition. As a pre-condition for “*twitter update*” it is necessary that (i) the music format be correct and (ii) the twitter login and password be correct for the update.

After modeling of service process, its features, and the service contracts, it is necessary to model the service composition process and its policies. π SOD-M uses a service composition diagram to describe the third party services and each policy allied to each composition (as

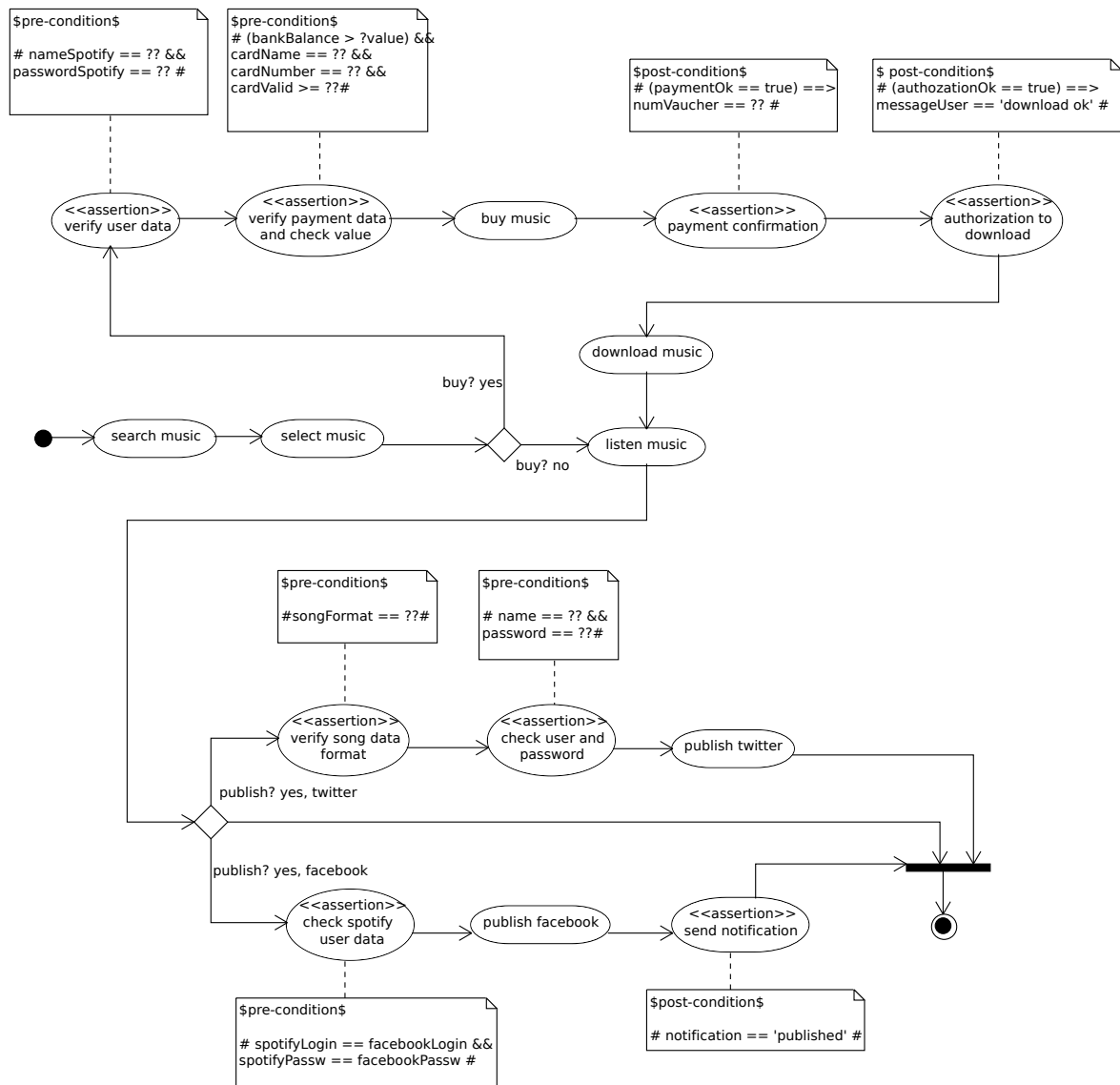


Figure 25: Service Process Model with Contract Definition - Complete Example.

presented in Figure 4) . In the next section we present this model and their respective policy concept .

3.2.3 π -ServiceComposition Model

The service composition model represents the workflow needed to execute a service or a set of them, but in a more detailed way, *i.e.*, identifying those entities that collaborate with the service process workflow, performing services that are necessary to execute the actions (in the business collaborators). Considering the workflow described in the π -ServiceProcess diagram, this model identifies the services and functions that correspond to each action in the business process.

Each action describes a fundamental behaviour unit that is part of a service activity and represents some behaviour in the system being modelled. The π -*ServiceComposition* model refines the concepts described in the π -*ServiceProcess* model, grouping contracts into policy (figure 22).

3.2.3.1 π -*ServiceComposition* Diagram, Terms and Concepts

A π -*ServiceComposition* model represents service compositions and their related business collaborators (external services), policies, rules associated with a policy and the whole application process and functionalities.

This model is essential for the organization and modeling of services and their location. We refine the representation of packages described in the π -*UseCase* model into business collaborator. The set of use cases described in a package (π -*UseCase* model) will be represented by an external service functions.

Figure 26 shows how to relate the actions described in the π -*ServiceProcess* model with services of a specific business collaborator. A service is represented by an stereotyped action («*service*»), and their connection is made by an edge (*Object Flow*). Thus each action described in the application workflow may be replaced by a service or a composition of them. If there is no relationship between the actions described in the application workflow (*External false* business collaborator) with external services, means that function is native to the application being developed.

The main properties described in the π -*ServiceComposition* model for the “to publish music” example are: (i) description of workflow application (*External false*), where the business collaborator *Application* points out that it is not an external workflow, (ii) the external services (*business collaborators*) and (iii) the application policies. Figures 26 and 27 presents how to model the π -*ServiceComposition* concepts (figure 28).

Figure 26 shows how each action is connected to an external service (through *Activity Edge - Object Flow*), and how to define the description of each business process, while Figure 27 shows each policy property and how it should be presented in the model.

Each business collaborator encompasses a set of services that can be performed by the application. These external services are used by the application must be described in this model, and not in the π -*ServiceProcess* model. The contracts described for each action are refined into policies, considering the non-functional requirement of each contract.

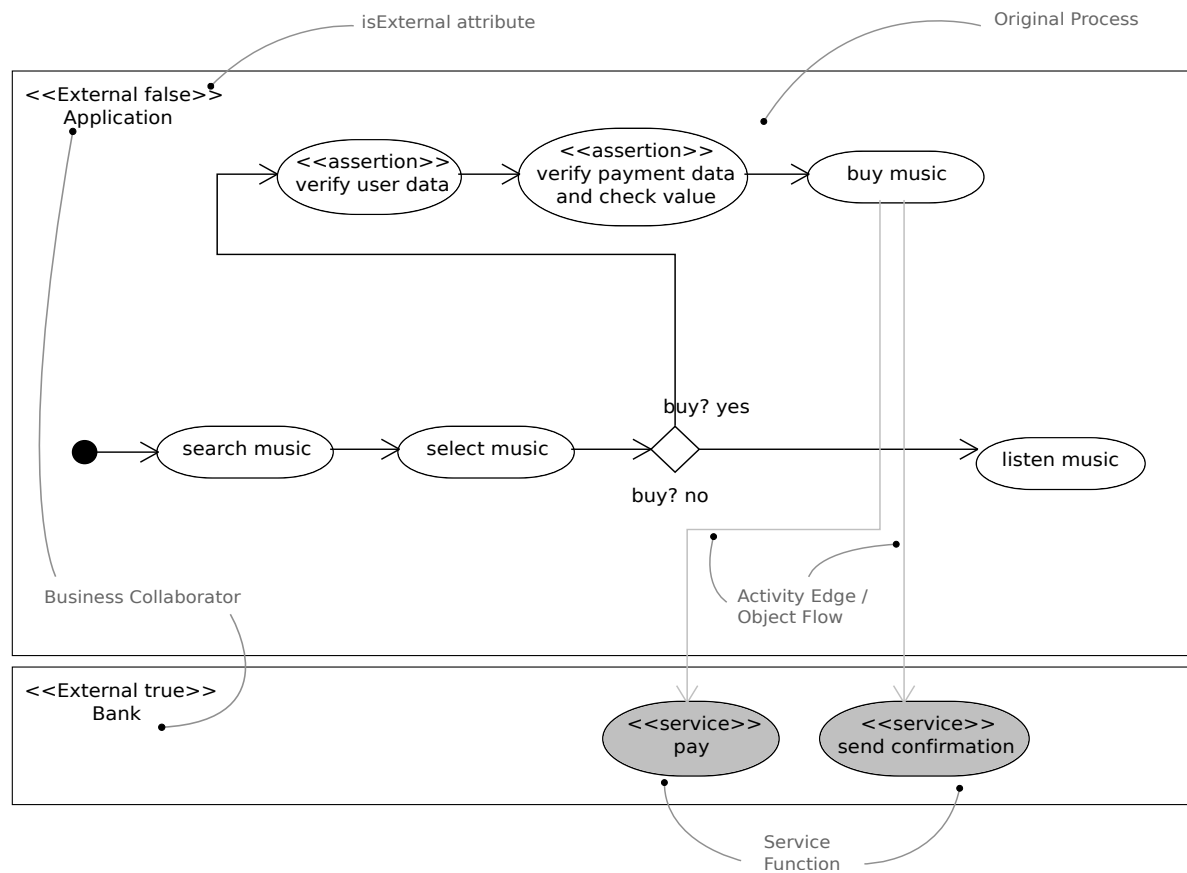


Figure 26: Business Collaborator Representation.

3.2.3.2 Meta-model

Considering the π SOD-M concepts (figure 5), the π -*ServiceComposition* meta-model (figure 28) describes the service compositions to be modelled and the policies related with each service composition. The π SOD-M concepts modelled in the π -*ServiceComposition* meta-model are: POLICY, RULE, VARIABLE, EXCEPTIONAL BEHAVIOUR, SERVICE ACTIVITY, ACTION, BUSINESS COLLABORATOR and NON-FUNCTIONAL REQUIREMENT.

π SOD-M proposes representing this model using the a extension of UML activity diagram technique. This extension describes the relationship of each ACTION for the respective SERVICE ACTIVITIES. Thus, as shown in figure 28, the meta-model includes typical modeling elements of the activity diagram such as ACTIVITY NODES, INITIAL NODES and FINAL NODES, DECISION NODES, etc., along with new elements defined by π SOD-M such as BUSINESS COLLABORATOR, SERVICE ACTIVITY and ACTION (see figures 28 and 30).

All π -*ServiceComposition* models describe the set of services and its relation with the service process and its activities that need to be performed. So, the external services are represented

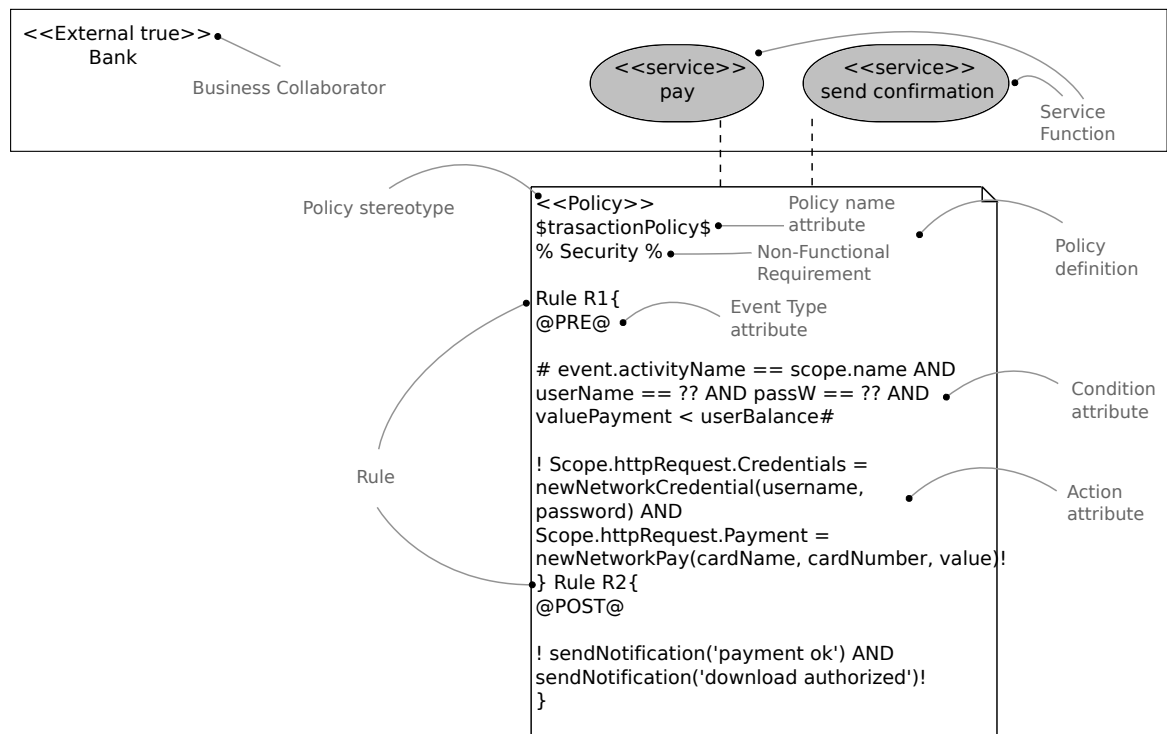


Figure 27: Policy Representation.

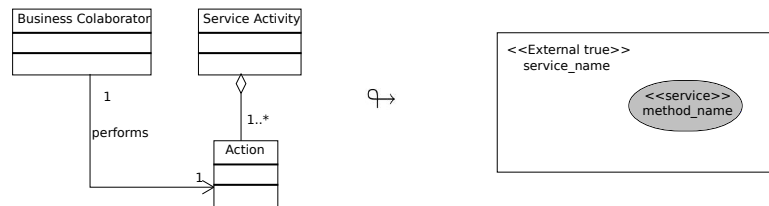
by the BUSINESS COLLABORATOR concept. All service composition may specify a POLICY as restriction. A POLICY is the set of CONTRACTS related with the service process expressed in the previous model (π -ServiceProcess).

Policy over service activities and business collaborators are not described in the original SOD-M method. The rule concept grouped into policy and exceptional behaviour make possible to model restrictions on service composition. Entities highlighted in the π -ServiceComposition (figure 28) meta-model are the difference from the original SOD-M service composition model.

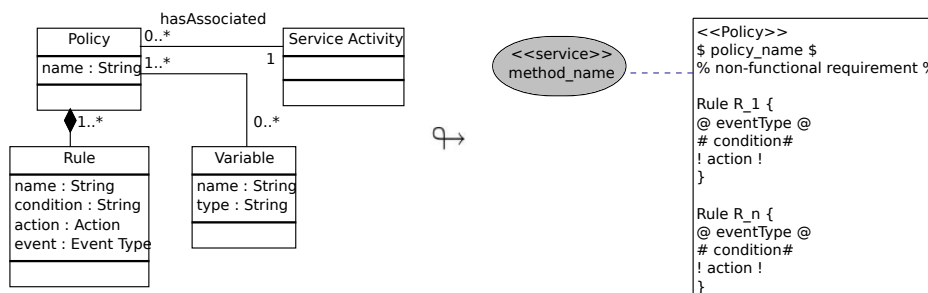
A BUSINESS COLLABORATOR element represents those entities that collaborate in the business processes by performing some of the required actions. They are graphically presented as a partition in the activity diagram. A collaborator can be either internal or external to the system being modelled. When the collaborator of the business is external to the system, the attribute *IsExternal* of the collaborator is set to *true*.

ACTION, a kind of EXECUTABLE NODE, are represented in the model as an activity. Each action identified in the model describes a fundamental behaviour unit which represents some type of transformation or processing in the system being modelled. There are two types of actions: i) a Web Service (attribute Type is WS); and ii) a simple operation that is not supported by a Web Service, called an ACTIVITY OPERATION (attribute Type is AOP).

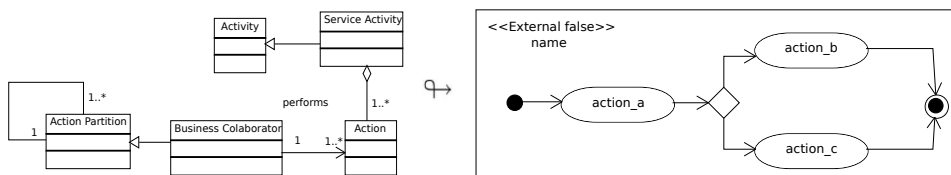
of π -ServiceComposition meta-model, those concepts will be used to describe how BUSINESS COLLABORATOR, ACTION, SERVICE ACTIVITY, POLICY AND RULES can be specified. Figure 29 also describes how the proposed concepts can be modelled in real system development cases. All π -ServiceComposition based models (presented in figure 29) must follow the meta-model description and its constraints.



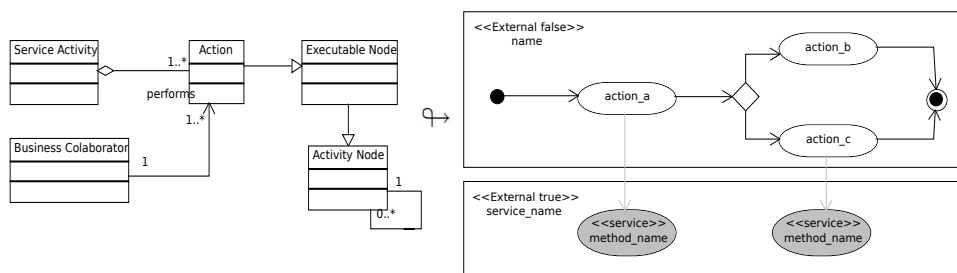
(a) Service Activity and Business Collaborator Model



(b) Policy and Rule Model



(c) Business Collaborator - External false Model



(d) Business Collaborator - External true Model

Figure 29: π -ServiceComposition Representation Models.

Figure 29a presents the representation for the relation between ACTION and BUSINESS

COLLABORATOR. As a SERVICE ACTIVITY are modelled as a set of ACTIONS (service functions), they are represented as «External true» BUSINESS COLLABORATOR. It means that an external service is invoked by the application's function. Each BUSINESS COLLABORATOR can represent an external service. Figure 29b presents the representation for the relation between ACTION, POLICY and RULE. Figure 29c presents the representation of the original *Service Process* associated to the «External false» BUSINESS COLLABORATOR and figure 29d presents how to associate the *Service Process*' ACTION with real services.

Notice that all π ServiceComposition meta-model concepts can be represented in the real application model, as we illustrate next using the scenario example.

3.2.3.4 Publish Music Service Composition

To illustrate the use of the π -ServiceComposition model we used it for defining the policy based composition model of the example scenario (see figure 30). There are four external BUSINESS COLLABORATORS, they are: *Bank*, *Spotify*, *Twitter* and *Facebook*. The model also shows the business process of the application that consists of six SERVICE ACTIVITIES (see figure 7): *search music*, *select music*, *buy music*, *download music*, *listen music* and *publish music*. Note that the action *publish music* of the application calls the actions of two service collaborators namely Facebook and Twitter, and the action *buy music* of the application calls two actions of the service collaborator Bank.

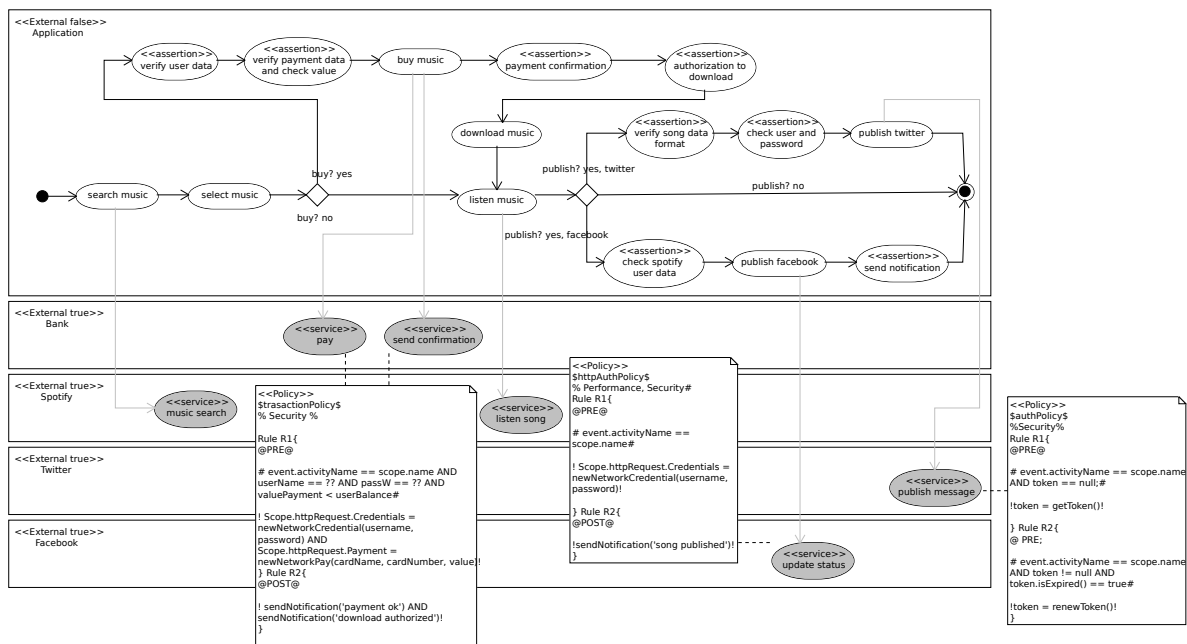


Figure 30: Service Composition Model with Policy Definition - Spotify Music Service.

The *Facebook* and *Twitter* services require authentication protocols in order to execute methods that will read and update the users' space. A call to such services must be part of the authentication protocol required by these services. In this example we associate two authentication policies, one for the open authentication protocol, represented by the class `Twitter AuthPolicy` that will be associated to the activity `UpdateTwitter` (see figure 30). In the same way, the class `Facebook HTTPAuthPolicy`, for the http authentication protocol will be associated to the activity `UpdateFacebook`. OAuth implements the open authentication protocol. As shown in figure 30, the *Policy* has a variable `Token` that will be used to store the authentication token provided by the service. This variable is imported through the library `Auth.Token`. The *Policy* defines two rules, both can be triggered by events of type `ActivityPrepared`: (i) if no token has been associated to the variable `token`, stated in by the condition of rule R_1 , then a token is obtained (action part of R_1); (ii) if the token has expired, stated in the condition of rule R_2 , then it is renewed (action part of R_2). Note that the code in the actions profits from the imported `Auth.Token` for transparently obtaining or renewing a token from a third party.

HTTP-Auth implements the HTTP-Auth protocol. As shown in figure 30, the *Policy* imports an http protocol library and it has two variables `username` and `password`. The event of type `ActivityPrepared` is the triggering event of the rule R_1 . On the notification of an event of that type, a credential is obtained using the `username` and `password` values. The object storing the credential is associated to the scope, i.e., the activity that will then use it for executing the method call.

Thanks to rules and policies it is possible to model and associate non-functional properties to services' compositions and then generate the code. For example, the atomic integration of information retrieved from different social network services, automatic generation of an integrated view of the operations executed in different social networks or for providing security in the communication channel when the payment service is called.

Back to the definition process of a SIS, once the *Policy* based services' composition model has been defined, then it can be transformed into a model (i.e., π -PEWS model, Section 3.3) that can support then executable code generation.

3.3 π -PEWS Platform Specific Models

Platform specific models (PSMs) are used to combine the specifications in the PIM with the details of the chosen implementation platform. PSM models are used to implement the

system (generating code automatically), and they must provide all information needed to build the system and its components. A PSM model can also run as a model to be used for further refinements by others PSM models.

π SOD-M platform model proposed at PSM level combines specific details of the service-based platforms. We use a π -PEWS language meta-model for representing the specification of a service composition on the language. A model can be then used to generate the corresponding code. π -PEWS [105] is an extension of the PEWS language that provides constructs for specifying policies for services through contracts clauses (see Appendix C).

3.3.1 π -PEWS Specification, Terms and Concepts

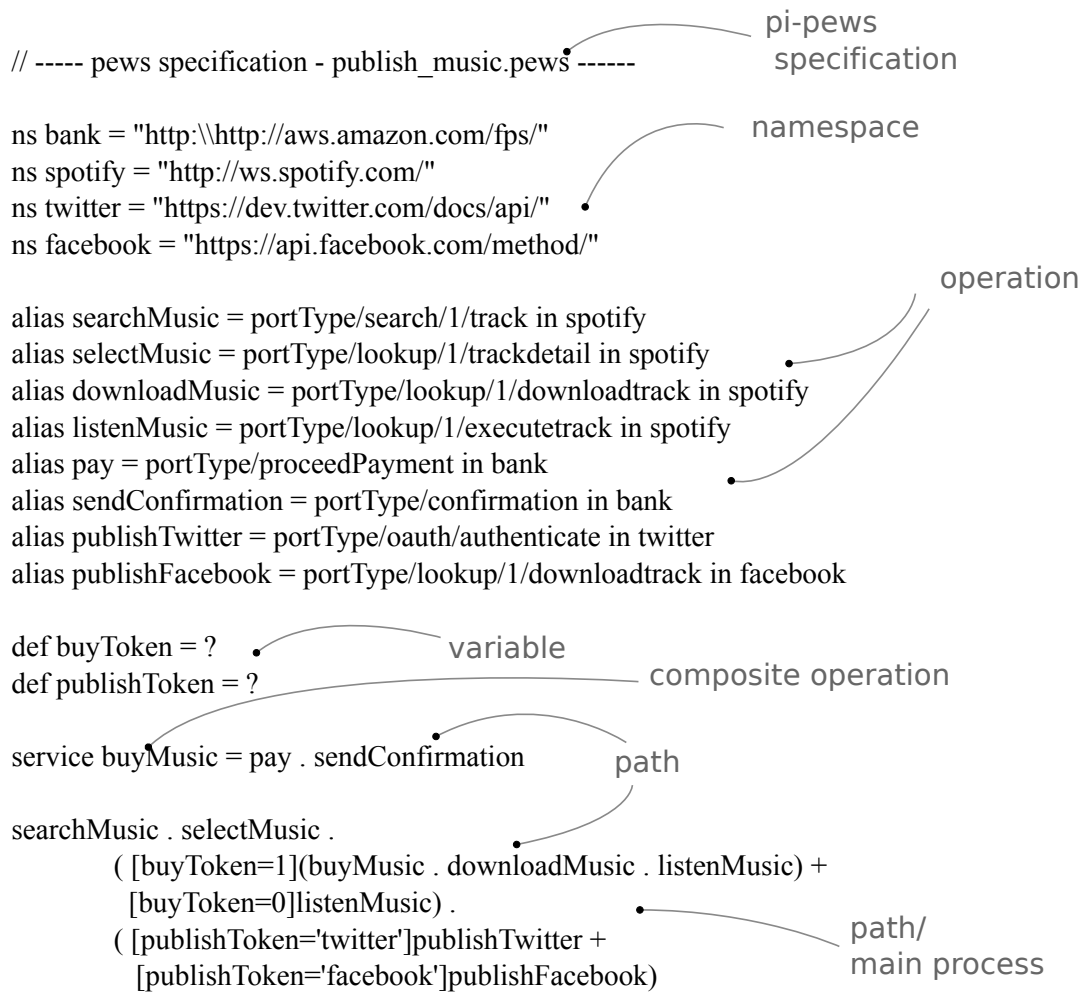
A π -PEWS model represents the system specification. This model is essential for the system specification, detailing the services, compositions and system's constraints. A π -PEWS specification describes the behavioural aspects of the system.

We extended the PEWS language [105] to support the notion *policy* through contracts definitions (see figure 32 and appendix C). The extension does not modify the syntax of existing PEWS programs, but complements it by specifying (non-functional) restrictions in a separate block. This feature is intended to enhance reusability and allows the programmer to adequate a program to specific requirements of the application being developed. This means that the program developer can reuse the control part of the program and add application-specific requirements as contract or time constraints.

Behavioural web service interface languages describe not only the input/output interface of a web service but also the expected behaviour of its components. This behaviour can be specified as a workflow, defining the *order* in which the components of a service will be executed, so that the workflow specifies the functional behaviour of the compound web service (figures 31 and 32).

Additionally, we can specify non-functional behaviour for a service; *i.e.*, to impose some additional restrictions which are separated from the application's workflow. This can be done by using the notion of *contract* to be added to each particular instantiation of the PEWS program. The result of this is to allow a given service (workflow) to have different restrictions in different contexts.

The policy model proposed here follows the main ideas presented in [46, 97], where contracts are added to a given composition, as a separate concern. The properties defined by a contract should be verified at runtime. Recovery actions, defined by the contract, are to be

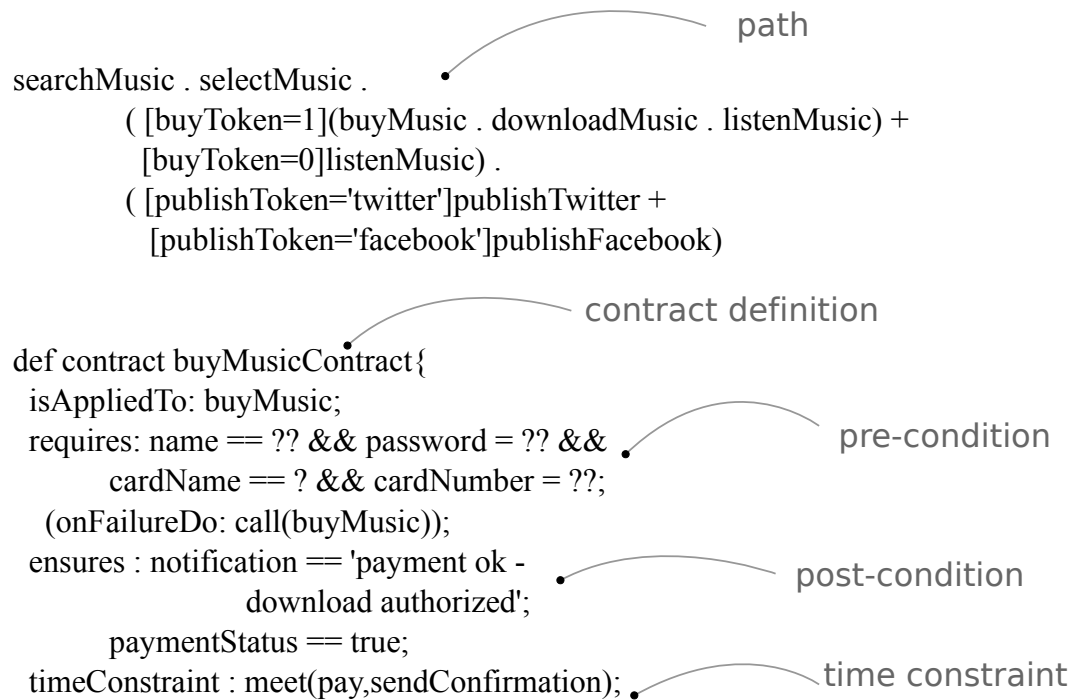
Figure 31: π -PEWS Specification Representation.

performed in case of failure of the contract's conditions. Recovery actions are defined by the contract itself.

3.3.2 Meta-model

The idea of the π -PEWS meta-model is based on the services' composition approach provided by the language PEWS [14, 105] (*Path Expressions for Web Services*). Figure 33 presents the π -PEWS meta-model consisting of classes representing:

- A services' composition: NAMESPACE representing the interface exported by a service, OPERATION that represents a call to a service method, COMPOSITE OPERATION, and OPERATOR for representing a services' composition and PATH representing a services' composition. A PATH can be an OPERATION or a COMPOSITE OPERATION denoted

Figure 32: π -PEWS Contract Representation.

by an identifier. A COMPOSITE OPERATION is defined using an OPERATOR that can be represent sequential (.) and parallel (||) composition of services, choice (+) among services, the sequential (*) and parallel ({ . . . }) repetition of an operation or the conditional execution of an operation ([C] S).

- *Policies* that can be associated to a services' composition: A-POLICY, RULE, EVENT, CONDITION, ACTION, STATE, and SCOPE.

As shown in the diagram an POLICY is applied to a SCOPE that can be either an OPERATION (e.g., an authentication protocol associated to a method exported by a service), an OPERATOR (e.g., a temporal constraint associated to a sequence of operators, the authorized delay between reading a song title in Spotify and updating the walls must be less then 30 seconds), and a PATH (e.g., executing the walls' update under a strict atomicity protocol – all or nothing). It groups a set of ECA rules, each rule having a classic semantics, i.e, *when an event of type E occurs if condition C is verified then execute the action A*. Thus, an *Policy* represents a set of reactions to be possibly executed if one or several triggering events of its rules are notified.

- The class SCOPE represents any element of a services' composition (i.e., operation, operator, path).

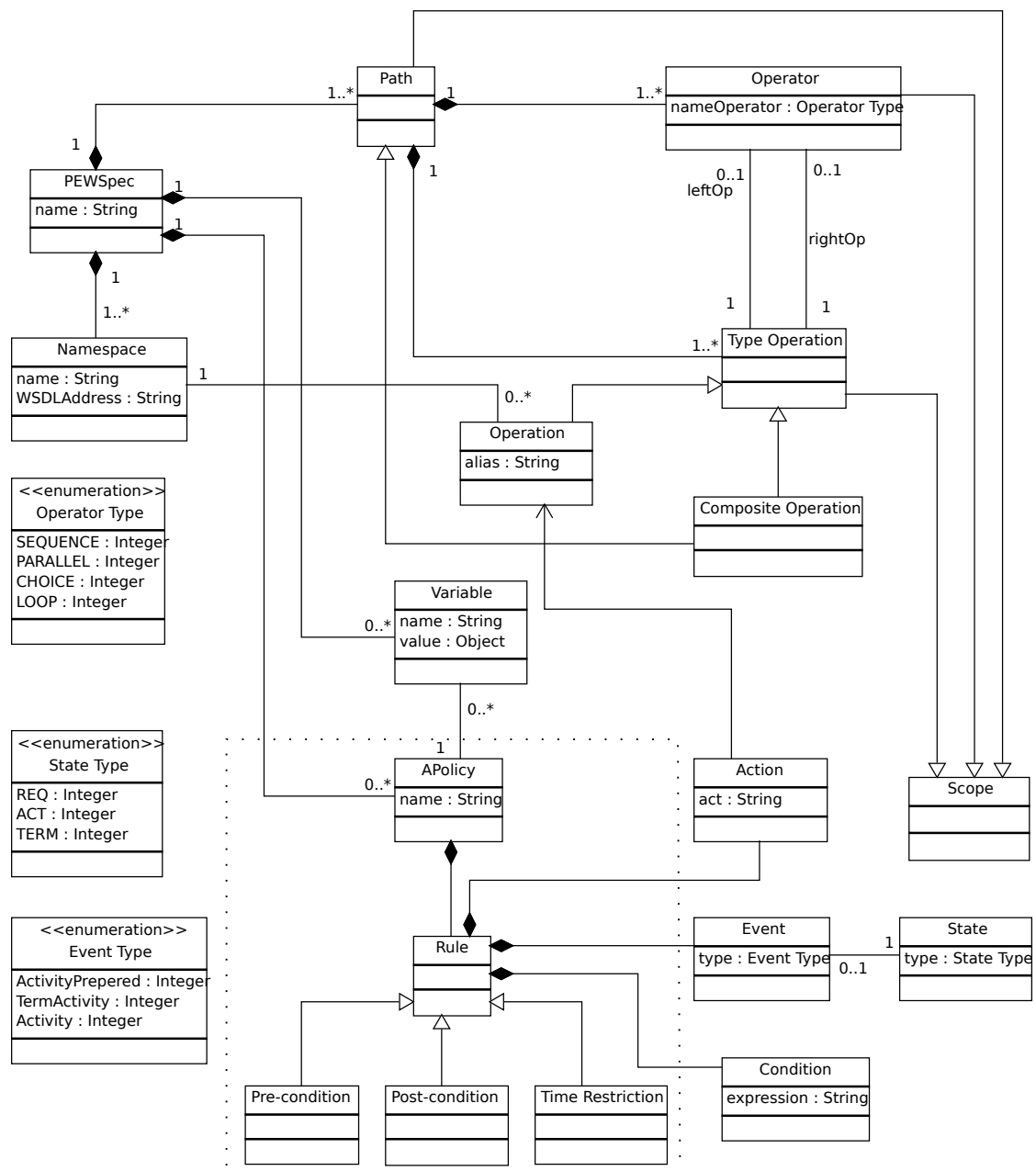


Figure 33: π -PEWS Meta-model.

- The class POLICY represents a recovery strategy implemented by ECA rules of the form EVENT - CONDITION - ACTION. A *Policy* has variables that represent the view of the execution state of its associated scope, that is required for executing the rules. The value of a variable is represented using the type VARIABLE. The class POLICY is specialized for defining specific constraints, for instance authentication policies.

Given a π -ServiceComposition model of a specific service-based application (expressed according to the π -ServiceComposition meta-model), it is possible to generate its corresponding π -PEWS model by using transformation rules. The following Section describes the transforma-

tion rules between the π -ServiceComposition and π -PEWS meta-models of our method.

The π -PEWS language (extension of PEWS) is described in the Appendix C. The generated code by π SOD-M, as end product, is based on the meta-model shown in figure 28 and the language syntax.

3.4 Model Transformations

A model transformation usually specifies which models are acceptable as input, and if appropriate what models it may produce as output, by specifying the meta-model to which model must conform. Model transformations can be seen as processes that take models as input and produces models as output. There is a wide variety of kinds of model transformation and uses of them, which differ in their inputs and outputs and also in the way they are expressed [80]. A model transformation is also a way of ensuring that a family of models is consistent, in a precise sense which the software engineer can define. The aim of using a model transformation is to save effort and reduce errors by automating the building and modification of models where possible [55].

The π SOD-M model transformations occur in the two levels, PIM and PSM, we defined: PIM-to-PIM and PIM-to-PSM transformations. Transformations in the same level are considered “horizontal transformations”. Transformations between different levels are called “vertical transformations”. There are 3 model transformations defined in π SOD-M: π -UseCase to π -ServiceProcess; π -ServiceProcess to π -ServiceComposition; and π -ServiceComposition to π -PEWS.

All π SOD-M transformation rules are described in natural language. These transformations ensure consistency between the concepts being refined and processed at different levels. Figure 34 presents the set of rules we defined to apply for each type of transformation. The entities of the left-hand side represent the source model, and those on the right-hand side represent the target model. Figure 34a shows the transformation rule in which a single source entity is transformed in a target model entity. Figure 34b describes a many to many entity transformation: a set of objects from the source model is transformed into a set of objects in the target model. Figure 34c presents a many to one transformation rule: many objects in the source model it will be transformed into a single object element in the target model. The next 3 rules are advanced rules, which transform single element in the source model into a set of different elements in the target model. In figure 34d, a source element is transformed in two different ones, *e.g.* from a “A” element, it generates a “B” and a “C” element in the target model. In figure 34e, a set

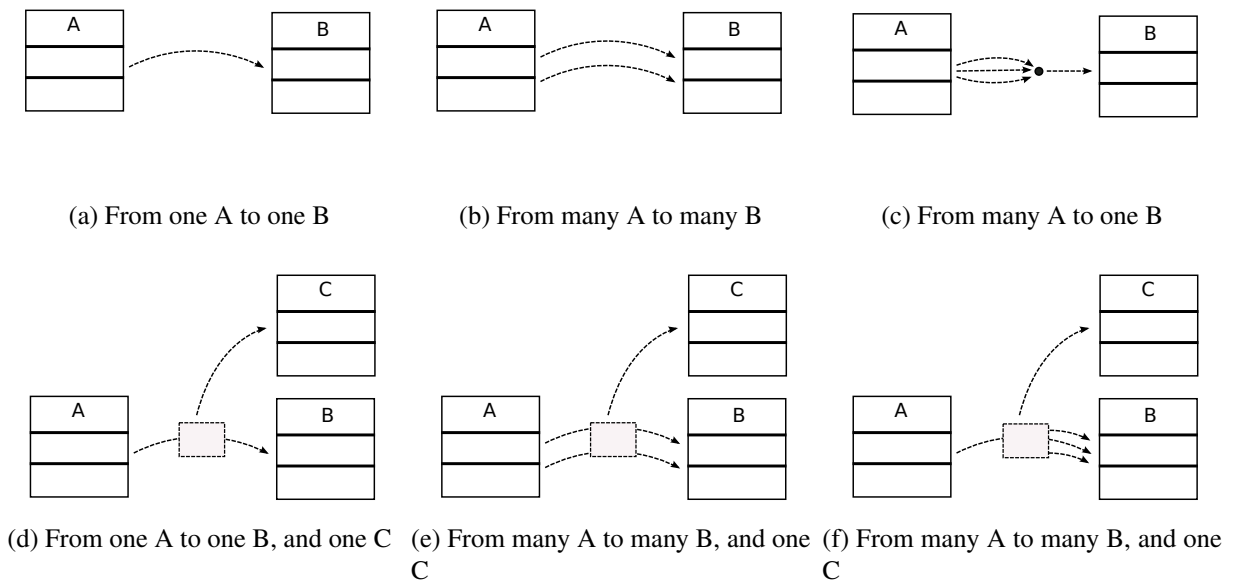


Figure 34: Entities' Model Transformation Rules.

of source elements is transformed into a set of elements “*B*” and one “*C*” element in the target model. Finally, figure 34f presents the rule which transforms source elements into a set of elements “*B*” and one element “*C*” in the target model.

3.4.1 From π -UseCase to π -ServiceProcess

The PIM to PIM transformation process between π -UseCase and π -ServiceProcess (Table 5) models defines how the application requirements are represented as a service process. At this level, services and general functions are represented as simple or composite use cases.

Every USE CASE is transformed into an ACTION in the π -ServiceProcess model, and every EXTEND association identified in the π -UseCase model is transformed into a FORK NODE, because several extends relations implies a conditional verification in the execution workflow . If the extend association has only one USE CASE, the fork will present the ACTION as an alternative flow, and later, both flows will meet. If the extend association has several source USE CASE, the fork will present different ACTIONS as mutual alternatives flows, and later, all these flows will meet. A ACTIVITY SERVICES (from π -ServiceProcess meta-model) consists of a composition of one or more ACTIONS. ACTION can be viewed as a method of the system application. Thus, services are represented as a set of functions in the π -ServiceProcess meta-model, because CONTRACTS modelled are associated to functions (ACTIONS).

All CONSTRAINTS in the source model (π -UseCase model) are transformed into a CON-

TRACT. A CONTRACT is represented as a set of CONSTRAINTS. Each constrain description is directly transformed into ASSERTION. A set of CONSTRAINT of a USE CASE or a COMPOSITE USE CASE is transformed into a set of ASSERTION used to define a CONTRACT. All USE CASES are associated to a NON-FUNCTIONAL REQUIREMENT and all CONSTRAINTS are associated with a NON-FUNCTIONAL ATTRIBUTE. Thus, CONSTRAINT representing the same USE CASE is transformed into a CONTRACT. A CONTRACT can have exceptional cases. The negation of the CONSTRAINT predicate are treated as EXCEPTIONAL BEHAVIOUR. In this particular case, a EXCEPTIONAL BEHAVIOUR will be generated in the target model.

The CONSTRAINT TYPE can have different transformations, depending of the following values:

- If the value of CONSTRAINT TYPE is *BUSINESS*, it will be transformed into none ASSERTION;
- If the value of CONSTRAINT TYPE is *VALUE* with the ASSERTION *isExceptionalBehaviour* attribute setted to *false*, it will be transformed into one ASSERTION;
- If the value of CONSTRAINT TYPE is *VALUE* with the ASSERTION *isExceptionalBehaviour* attribute setted to *true*, it will be transformed into a EXCEPTIONAL BEHAVIOUR;

Given the π -UseCase model, to all CONSTRAINT entity related with a USE CASE, there is a CONTRACT that compounds a set of ASSERTIONS entity (figure 35a) and the USE CASE is refined in a service ACTION.

Example: Considering the scenario example presented in Section 3.1.4, the transformation of the “*listen music*” USE CASE is transformed into a service ACTION. This action is a Spotify service function that can be invoked to play the music. For “*publish music*” USE CASE, all CONSTRAINS are transformed in a set of ASSERTIONS that are grouped in a CONTRACT (“*publishMusicContract*”). The “*publishMusicContract*” is related with the “*publishMusic*” ACTION. □

A CONSTRAIN transformation means that from CONSTRAINT and CONSTRAINT TYPE entities are generated detailed CONTRACT information, that are refined into EXCEPTIONAL BEHAVIOURS and ASSERTIONS entities (figure 35b). CONSTRAINT is related with a CONSTRAINT TYPE, and there are different cases for the transformation of this concept. A *Business* type are transformed into an ASSERTION that does not consider value attributes information like *maxValue* and *minValue* (see figure 13). These ASSERTION information attributes are considered if the CONSTRAINT TYPE is a *Value* type. This is semi-automatic, because there is not

Table 5: Transformation Rules: From π -UseCase to π -ServiceProcess.

Source π -UseCase meta-model	Mapping Rules	Target π -ServiceProcess meta-model
Constraint	<ul style="list-style-type: none"> - All Constraints in the source model will be transformed into a Assertion in the target model. All Use Cases are associated to a Non-Functional Requirement and all Constraints are associated with a Non-Functional Attribute. Thus, Constraint representing the same Use Case, is transformed in a Contract. - A Contract can also have a exceptional cases. The negation of the Constraint predicate are treated as the type <i>EXCEPTIONALBEHAVIOUR</i>. Thus, all the Assertions related to this Constraint, in this particular case, will be transformed in a Exceptional Behaviour. - The set of ASSERTIONS of a USE CASE are associated with a CONTRACT - The CONTRACT name is formed by "<useCaseName>" + "Contract" token. 	Assertion, Contract, Exceptional Behaviour
Constraint Type	<ul style="list-style-type: none"> - The Constraint Type in source model can have the following transformation: <ul style="list-style-type: none"> • BUSINESS type is transformed into none Assertion; • VALUE with the Assertion <i>isExceptionalBehaviour</i> attribute setted to false is transformed into a Assertion in the target model; • EXCEPTIONALBERAVIOR with the Assertion <i>isExceptionalBehaviour</i> attribute setted to true is transformed into a Exceptional Behaviour in the target model. - Both, the value and exceptional restriction are based on in accordance with the values of Assertions - Constraint Type defines the type of restriction on a Use Case. 	Assertion Exceptional Behaviour
Non-Functional Attribute	Every Non-Functional Attribute associated to an element (Constraint and Non-Functional Requirements) in the source model becomes a Non-Functional Attribute associated to the corresponding element (Contract) in the target model.	Non-Functional Attribute
Use Case	For every Use Case there will be a Action in the service delivery process model describing the Business Service	Action
Extend (Use Case)	<ul style="list-style-type: none"> - Every Extend association identified in the π-UseCase model will be represented in the target model by a Fork Node. - The Service Activity corresponding to the source Use Case of the extend association must be previous to the Service Activity corresponding to the target Use Case of the extend association. - If the extend association has only one source Use Case, the fork will present the Service Activity as an alternative to another flow with no Service Activity. Later, both flows will meet; - If the extend association has several source Use Case, the fork will present the different Service Activities as mutual alternatives to another flow with no Service Activity. Later, all these flows will meet; 	Control Node, Fork Node, Service Activity
Include (Use Case)	<ul style="list-style-type: none"> - An Include association is found in the π-UseCase model, the Service Activity corresponding to the source must be subsequent to the Service Activity corresponding to the target Use Case of the include association; - If the include association has several targets, the designer must decide the appropriate sequence for the different Service Activities corresponding to the target Use Case (which will obviously be previous to the Service Activity corresponding to the source Use Case). 	Service Activity
Requirement	<ul style="list-style-type: none"> - A Requirement in the source model is transformed into Service Activity of the target model. - The rules for a Requirement to be transformed into a service activity is analyzed after the transformations of included and extended use cases. - A Requirement associated with only one Use Case in the source model is transformed in a Service Activity and an Action, respectively. - All Use Cases must be associated with a Requirement. 	Service Activity

enough information in a π -UseCase model to run a complete automatic transformation. If the **CONSTRAINT TYPE** is a *Value* type, the designer must specify the variable information, and its boundary values after the transformation. For this, the designer must consider the **CONSTRAINT** and **USE CASE** descriptions. By default, *value constraints* are transformed into *pre-conditions* and *business constraints* are transformed into *post-conditions*. This rule may be adjusted by the designer at the time of transformation.

A **SERVICE ACTIVITY** and the **ACTIONS** are generated from two different elements in the source model (π -UseCase), from a **REQUIREMENT** or a **COMPOSITE USE CASE**. Figures 35c and 35d present the transformation schema for **REQUIREMENT** and **COMPOSITE USE CASE** concepts. Both elements are related with many **USE CASES** that are transformed in many **ACTIONS**. A **USE CASE** are related with, either, a **REQUIREMENT** and a **COMPOSITE USE**

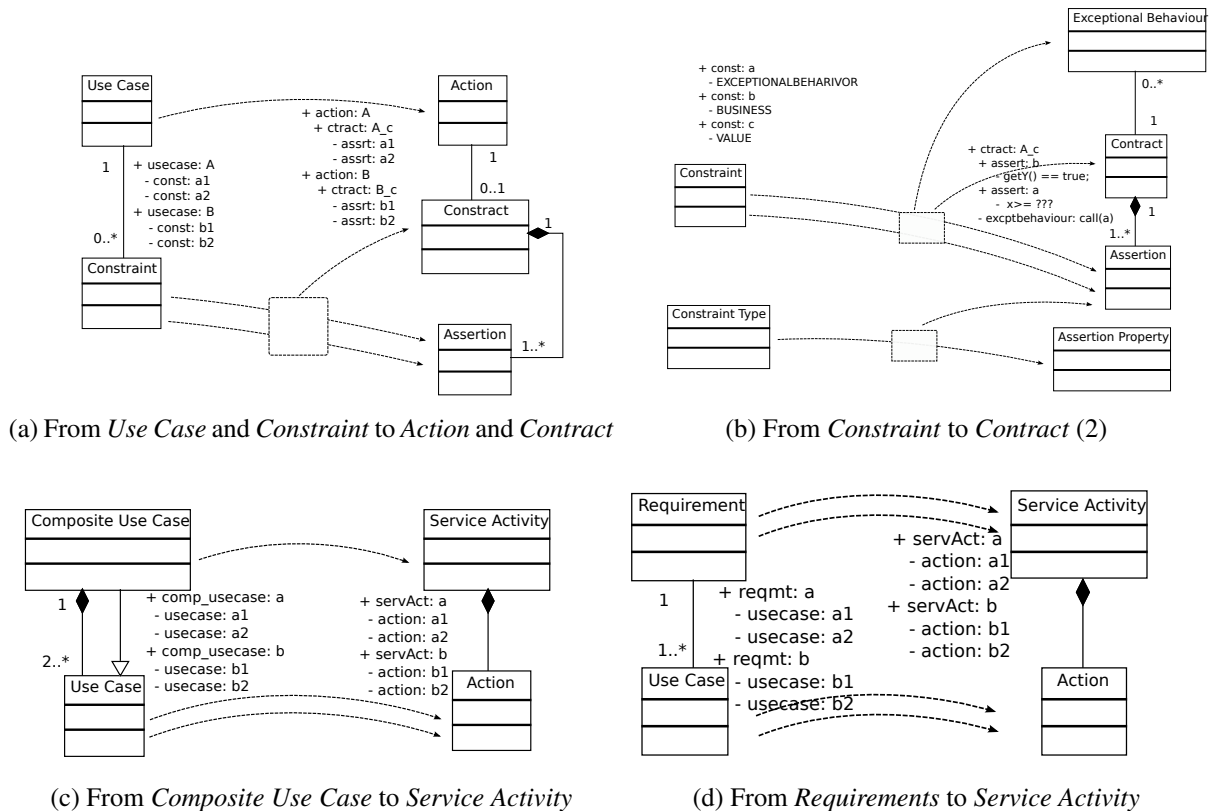


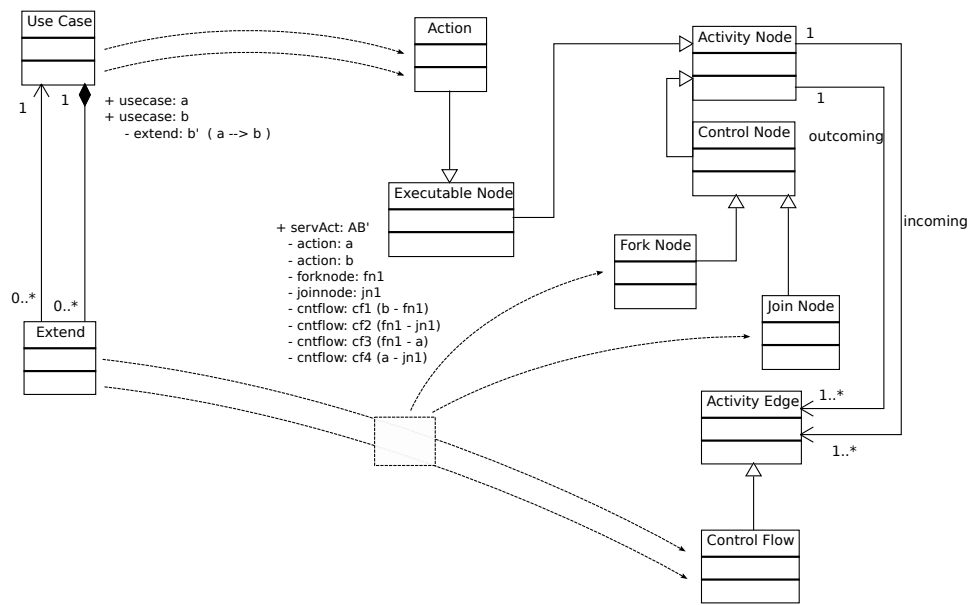
Figure 35: π -UseCase2 π -ServiceProcess Model Transformation Rules.

CASE. ACTIONS are related with a SERVICE ACTIVITY.

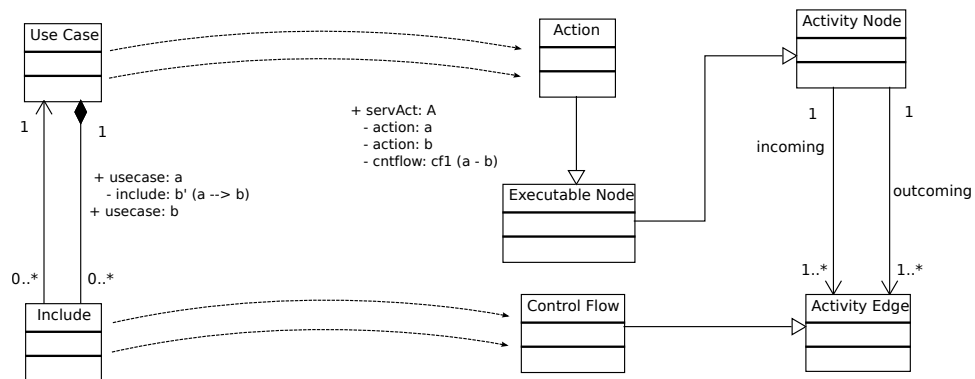
Example: For the execution of the “download music” use case is necessary process with the payment process. Thus, the set of USE CASES that include the “download music” process are transformed in ACTIONS, and a SERVICE ACTIVITY that aggregates all these ACTIONS is also generated. □

The transformations for EXTEND and INCLUDE dependence elements are not as simple as the previous transformations (figures 36a. The workflow generated for each «extends» relationship with just two use cases (figure 37a) is described by figure 37c. The generated workflow contains: one FORK NODE, one JOIN NODE, and four CONTROL FLOW elements, and also two ACTION elements, one for each USE CASE, if there are less than 2 extended USE CASE. When there is more than one «extends» relationship among different use cases (figures 37b and 37d), the transformation is proceed as: adding two more CONTROL FLOW for each new USE CASE, and one ACTION for each new extended USE CASE.

Example: Considering the scenario example, it is possible to apply the same rule for the “publish music” use case, which has two extended use cases, “public twitter” and “public Facebook”, applying the illustration expressed in figures 37b and 37d. □



(a) From *Extend* to Fork/Join Flow



(b) From *Include* to Activity Node Flow

Figure 36: π -UseCase2 π -ServiceProcess Model Transformation Rules (2).

For the *INCLUDE* use case elements, the transformation is represented as a *ACTION* sequence, as shown in figure 36b. For each *USE CASE* element an *ACTION* element is generated. For a set of n *USE CASES* we generate $n-1$ *OBJECT FLOW* elements. Each *CONTROL FLOW* links two *ACTIONS*.

Example: Given the “download music” use case from the scenario example, it includes payment process to buy the music. It is represented as a *include* in the π -UseCase model and is transformed in a sequence flow in the π -ServiceProcess model (as presented in figure 38). □

All this transformation rules from π -UseCase model to π -ServiceProcess model were validated with the scenario example. All transformations described in this section are completely automatic, however information about values and its boundaries are inserted by the designer after the transformation process.

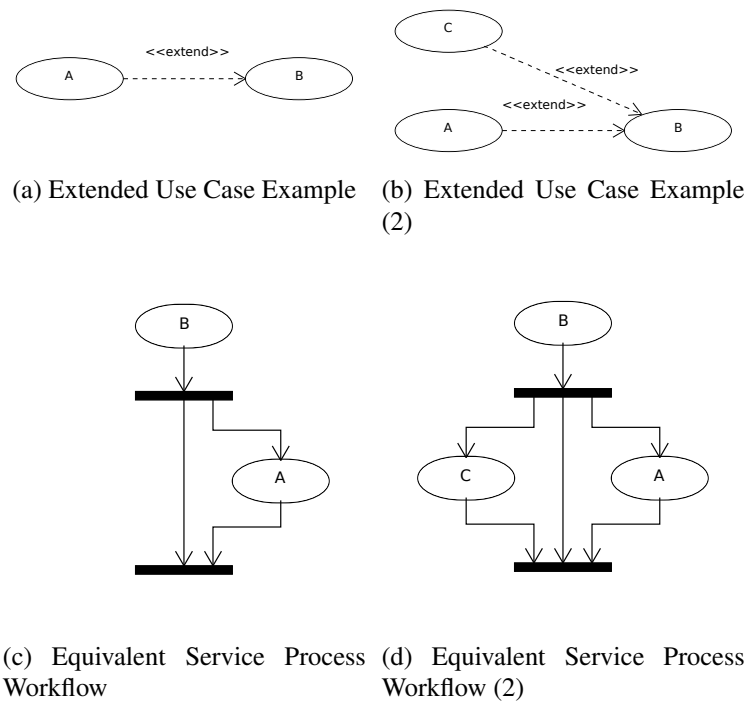


Figure 37: Extended Transformation Examples.

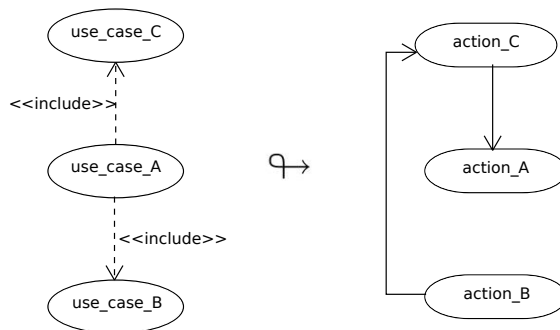


Figure 38: Include Transformation Example.

3.4.2 From π -ServiceProcess to π -ServiceComposition

The PIM to PIM transformation process from π -ServiceProcess to π -ServiceComposition model will refine further the application design. The main goal of this transformation is to group all CONTRACTS and ACTIONS in POLICIES and SERVICE ACTIVITIES, respectively.

All ACTIONS entities in the source model will be transformed into an ACTION in the target model, and every SERVICE ACTIVITY in the source model will be transformed into a SERVICE ACTIVITY in the target model. This happens because π -ServiceComposition is an extension of π -ServiceProcess.

For the non-functional specifications, the CONTRACT entity with the same NON-

FUNCTIONAL REQUIREMENT in the source model (π -ServiceProcess model) will be transformed into a POLICY in the target model (π -ServiceComposition model). Each ASSERTION will be transformed into a RULE:CONDITION attribute in the target model, but if the ASSERTION has a value type, the *name* and the attributes in the source model will be transformed into a VARIABLE in the target model. The other ASSERTION PROPERTY values remains unchanged.

The ASSERTION:APROPERTY attribute can have different transformations, depending of the following values:

- *POST-CONDITIONS* are transformed into *POST* in the π -ServiceComposition model;
- *PRECONDITIONS* are transformed into *PRE* in the π -ServiceComposition model;
- *TIMERESTRICTIONS* are transformed into *TIME* in the π -ServiceComposition model.

The EXCEPTIONAL BEHAVIOUR entities will be transformed into an ACTION in the π -ServiceComposition model, and every NON-FUNCTIONAL ATTRIBUTE associated to an element (CONTRACT and NON-FUNCTIONAL REQUIREMENT) in the π -ServiceProcess model becomes a NON-FUNCTIONAL ATTRIBUTE associated to the corresponding element (POLICY) in the π -ServiceComposition model.

Table 6 describes the transformations between π -ServiceProcess and π -ServiceComposition meta-models.

Table 6: Transformation Rules: From π -ServiceProcess to π -ServiceComposition.

Source π -ServiceProcess meta-model	Mapping Rules	Target π -ServiceComposition meta-model
Action	All Actions in the source model will be transformed in a Action in the target model.	Action
Service Activity	- All Service Activity in the source model will be transformed in a Service Activity in the target model.	Service Activity,
Contract, Non-Functional Attribute	All Contract with the same Non-Functional Requirement , in the source model, will be transformed in a Policy in the target model.	Policy
Assertion	Each Assertion will be transformed in a Rule:condition attribute in the target model.	Rule:condition
Assertion:name, Assertion:type	If the Assertion has the type equal to <i>VALUE</i> , the <i>name</i> and the <i>type</i> attribute in the source model will be transformed in a Variable in the target model.	Variable
Exceptional Behaviour	An Exceptional Behaviour and its attributes in the source model will be transformed in a Rule:action in the target model.	Rule:action
Assertion:aProperty	- Depending on the assertion Property type in the source model, it will be transformed in a Rule:event in the target model. The transformation rules are: <ul style="list-style-type: none"> • <i>PRECONDITION</i> type is transformed into <i>PRE</i> in the target model; • <i>POST-CONDITION</i> type is transformed into <i>POST</i> in the target model; • <i>TIMERESTRICTION</i> type is transformed into <i>TIME</i> in the target model. 	Rule:Event, Event Type
Non-Functional Attribute	Every Non-Functional Attribute associated to an element (Contract and Non-Functional Requirements) in the source model becomes a Non-Functional Attribute associated to the corresponding element (Policy) in the target model.	Non-Functional Attribute

Figure 39 presents the main transformation rules for this level. The main feature of this transformation is the generation of a POLICY from of a set of CONTRACTS. But, *what is the adopted criterion for this transformation?* The main criteria is that all CONTRACTS from the

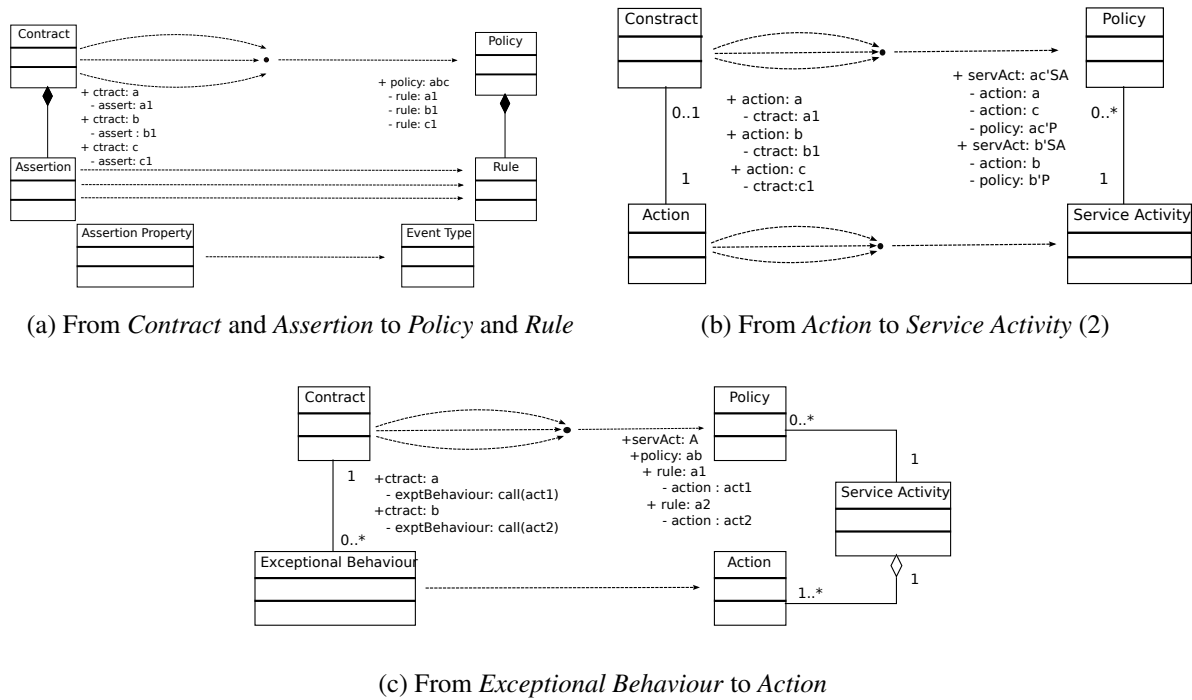


Figure 39: π -ServiceProcess2 π -ServiceComposition Model Transformation Rules.

same NON-FUNCTIONAL REQUIREMENT will be transformed in one POLICY. Thus, we can have POLICIES on: *Safety*, *Performance*, *Reliability*, and so on.

Each ASSERTION of a CONTRACT (π -UseCase model) is transformed into a RULE in the π -ServiceProcess model. The set of CONTRACTS generates a POLICY composed by those RULES. Thus, the set of RULES make up a single POLICY. It is important to highlight that only the CONTRACTS of the same NON-FUNCTIONAL REQUIREMENT is transformed into the same POLICY. Different NFRs' CONTRACTS, are transformed into different POLICIES. Figure 39a show how this transformation is made.

Similar to the transformation between CONTRACT and POLICY, a set of ACTIONS is transformed into a single SERVICE ACTIVITY entity. ACTIONS are related with BUSINESS COLLABORATOR, and all information related to a BUSINESS COLLABORATOR are transformed from PACKAGES information (π -UseCase model). All PACKAGES are transformed into BUSINESS COLLABORATOR. Figure 39b presents the transformation between ACTION and SERVICE ACTIVITY elements.

An exceptional behaviour is triggered by a failure in the verification of the condition of a rule. An EXCEPTIONAL BEHAVIOUR is performed to preserve the state of the application. Figure 39c presents how an EXCEPTIONAL BEHAVIOUR that can be transformed into an ACTION call.

As the π -ServiceComposition model refines the π -ServiceProcess concepts at PIM level, a service previously defined as actions (source model) is refined as composition of those actions (target model) that are necessary to represent a business service, identifying who are the partners involved in the realization (BUSINESS COLLABORATORS). In addition, π SOD-M defines a platform specific model based on web services composition. This model is explicitly indicates those actions which are (or will be, if not yet implemented) supported by web services.

Example: Considering the scenario example, the ACTION's *update music* CONTRACT is transformed is a POLICY with its RULES. All contract ASSERTIONS are transformed in RULE and its attributes, e.g. the login and password verification. The “*securityLoginPolicy*” is all set of RULES that were transformed from the ASSERTIONS in π -ServiceProcess model. The NON-FUNCTIONAL REQUIREMENT information will be used to the POLICY generation comes from the initial use case model. Also the BUSINESS COLLABORATOR *Facebook* and *Spotify* information came from PACKAGE π -UseCase entity element. All CONTRACTS of the same NON-FUNCTIONAL REQUIREMENT are composed in a POLICY. \square

3.4.3 From π -ServiceComposition to π -PEWS

Table 7 describes the PIM to PSM transformations between π -ServiceComposition and π -PEWS meta-models. We propose two groups of rules: those that transform services composition elements of the π -ServiceComposition into π -PEWS meta-model elements; and those that transform rules grouped by policies into A-policy types.

A named action of the π -ServiceComposition represented by *Action* and *Action:name* is transformed to a named class OPERATION with a corresponding attribute name OPERATION:NAME. A named service activity represented by the elements *Service Activity* and *Service Activity:name* of the π -ServiceComposition, are transformed into a named operation of the π -PEWS represented by the elements COMPOSITE OPERATION and COMPOSITE OPERATION:NAME. When more than one action is called, according to the following composition patterns expressed using the operators *merge*, *decision*, *fork* and *join* in the π -ServiceComposition the corresponding transformations, according to the PEWS operators presented above, are:

- $op_1.op_2$ if no *Control Node* is specified
- $(op_1 \parallel op_2).op_3$ if control nodes of type *fork*, *join* are combined
- $(op_1 + op_2).op_3$ if control nodes of type *decision*, *merge* are combined

The *A-policies* defined for the elements of the π -ServiceComposition are transformed into A-POLICY classes, named according to the names expressed in the source model. The transformation of the rules expressed in the π -ServiceComposition is guided by the event types associated to these rules. The variables associated to an *A-policy* expressed in the π -ServiceComposition as $\langle Variable:name, Variable:type \rangle$ are transformed into elements of type VARIABLE with attributes NAME and TYPE directly specified from the elements *Variable:name* and *Variable:type* of the π -ServiceComposition model.

Table 7: Transformation Rules: From π -ServiceComposition to π -PEWS.

Source π -ServiceComposition meta-model	Mapping Rules	Target π -PEWS meta-model
Action	- An <i>Action</i> in the source model corresponding to an external Business Collaborator is mapped to an <i>Operation</i> in target model. - The Action:name in the source model is transformed into Operation:name in the target model.	Operation:alias
Service Activity	- The Service Activity in the source model is mapped to a Composite Operation in target model when more than one Actions are called. - If Composite Operation is generated for a given <i>Service Activity</i> then the Service Activity:name in the source model is mapped to Composition Operation:name in the target model.	Type Operation, Composition Operation
Control Nodes	- The Control Node in the source model is mapped to a Operator in target model. According to the type of Control Node (merge, decision, join, fork) the expression of the Composite Operation is: • Sequence if no Control Node is specified; • Parallel - Sequence for a Control Nodes pattern fork - join; • Choice - Sequence for a Control Node pattern decision - merge	Operator
Business Collaborator	A Business Collaborator:isExternal in the source model generates a Namespaces in the target model	Namespace
Rule:event	The Rule's attribute event in the source model is transformed into an Event:type of the target model. In this case attribute is mapped to an entity with an attribute. The Event Type of a Rule in the target model is determined by the Rule type: • Event Type of a <i>Precondition Rule</i> is <i>ActivityPrepared</i> ; • Event Type of a <i>Postcondition Rule</i> is <i>TermActivity</i> ; • Event Type of a <i>TimeRestriction Rule</i> is <i>Activity</i> ;	Event Type, Event
Rule:condition	The Rule's attribute condition in the source model is transformed into a Condition:expression in the target model. In this case, an attribute is mapped into an entity with an attribute	Condition
Rule:action	The Rule:action in the source model is transformed in an Action:act in the target model. The attribute action is mapped to an entity with an attribute. In the target model an action is executed according to the rule condition value (true/false).	Action
Policy	- Every Policy associated to an element (Business Collaborator, Service, Activity, Action) in the source model becomes an APolicy associated to the corresponding element in the target model. - The name attribute of a Policy in the source model becomes an Apolicy:name of the target model.	APolicy
Variable	Every Variable , and its attributes, associated to a Policy in the source model becomes a Variable associated to an APolicy in the target model. The variables can be used in an APolicy's Condition of the target model.	Variable
Rule:event	For a Rule in the source model, depending on the Event Type , the corresponding transformation in the target model is: Precondition, Postcondition or TimeRestriction Rule	Precondition Postcondition TimeRestriction Rule

As shown in Table 7, for an event of type *Pre* the corresponding transformed rule is of type PRECONDITION; for an event of type *Post* the corresponding transformed rule is of type POSTCONDITION; finally, for an event of type *TimeRestriction* the corresponding transformed rule is of type TIME. The condition expression of a rule in the π -ServiceComposition (*Rule:condition*) is transformed into a class *Condition:expression* where the attributes of the expression are transformed into elements of type ATTRIBUTE.

Figures 31 and 32 present the π -PEWS specification resulting from the π -ServiceComposition model transformation of our scenario example.

3.5 Conclusions

This chapter presented π -SOD-M for specifying and designing reliable service based applications, using a MDA approach. As one of the main aims of MDA is to separate design from architecture and technologies, the π SOD-M's models describe the system behaviour and its restrictions, without considering definition of a standard architecture.

Non-functional constraints are related to business rules associated to the general semantics of the application. In the case of service-based applications, this type of applications also concern the use constraints imposed by the services. We worked on the definition of a method for explicitly expressing such properties in the early stages of the specification of services based applications. Having such business rules expressed and then translated and associated to the services' composition can help to ensure that the resulting application is compliant to the user requirements and also to the characteristics of the services it uses.

Our approach considers that it is possible to model both aspects, functional and non-functional, in an service-oriented application development throughout the whole development process.

A business process specification can be modelled with the concepts USE CASE and REQUIREMENT. In the same level, the NFRs can be modelled using the concepts of CONSTRAINT, CONSTRAINT TYPE and NF-REQUIREMENT. The concepts used to represent the functional aspects and the business process were modeled by SOD-M method.

In order to fill the gap between the abstract specification and the concrete implementation, the Business process activities that can be done by external entities can be modelled with the concepts ACTION, BUSINESS COLLABORATOR. In order to express NFP from NFRs we proposed the concepts of: CONTRACT, ASSERTION (PROPERTY) and EXCEPTIONAL BEHAVIOUR.

Since our objective is also to provide a platform dependent specification, We also propose a platform specific meta-model, named π -PEWS, which represents the π -PEWS specification language for define web services path expressions.

Designing and programming non-functional properties is not an easy task, so we are defining a set of predefined *Policy* types with the associated use rules for guiding the programmer when she associates them to a concrete application. *Policy* types that can also serve as patterns for programming or specializing the way non-functional constraints are programmed. We model and associate policies to service-based applications that represent both systems' cross-cutting

aspects and use constraints stemming from the services used for implementing them.

An advantage of our methodology is the use of high-level models, which by means of model transformations, helps software developers to make the most of the knowledge for specifying and developing business services.

4 π SOD-M Environment

*“The computer was born to solve
problems that did not exist before.”*

Bill Gates

This chapter describes the π SOD-M environment for developing reliable service based systems using the methodology we propose. The π SOD-M environment was developed to support the π SOD-M methodology. The π SOD-M based process consists in generating an application starting from a π -UseCase model and then transforming it into series of models at the PIM and PSM levels before finally generating the code that implements the application.

π SOD-M environment is built on the top of Eclipse framework (<http://www.eclipse.org>), which is a framework to build Integrated Development Environments (IDEs). We also use the Eclipse Modelling Framework (EMF), a meta-modelling framework that was devised to be extended and provides with the utilities needed to define, edit and handle (meta)-models. To automate the transformation models we use the ATL Language¹ [55], a model transformation language framed in Eclipse. Another language used for model transformation is Acceleo [84], a model to text engine. π SOD-M uses this environment² to generate service composition specification code in π -PEWS.

The remainder of the chapter is organized as follows. Section 4.1 describes the environment architecture, including the definition of meta-models; and, the implementation of the model transformation process and the code generation engine. In section 4.2 describes how to install and use the environment. Section 4.3 describes how to extend the environment, for adding new components for generation of system specification in different languages. Section 4.4 concludes the chapter.

¹ATL uses EMF to handle models (that is to serialize, to navigate and to modify them). Once the ATL transformation is executed, the ATL engine matches the source pattern and the source model and instantiates each pattern into the target model.

² The environmental requirements are the Eclipse Modelling (Galileo or higher) and all proposed plugins, π -UseCase, π -ServiceProcess, π -ServiceComposition and π -PEWS editors; and also the transformation plugins. Because the environment is proposed in the context of Eclipse, the environmental requirements are also the same needed to utilize the Eclipse tool.

4.1 General Architecture

Figure 40 presents the general architecture for the π SOD-M environment and details how components interact with each other. Each model has an editor, and transformation tools that support the application development.

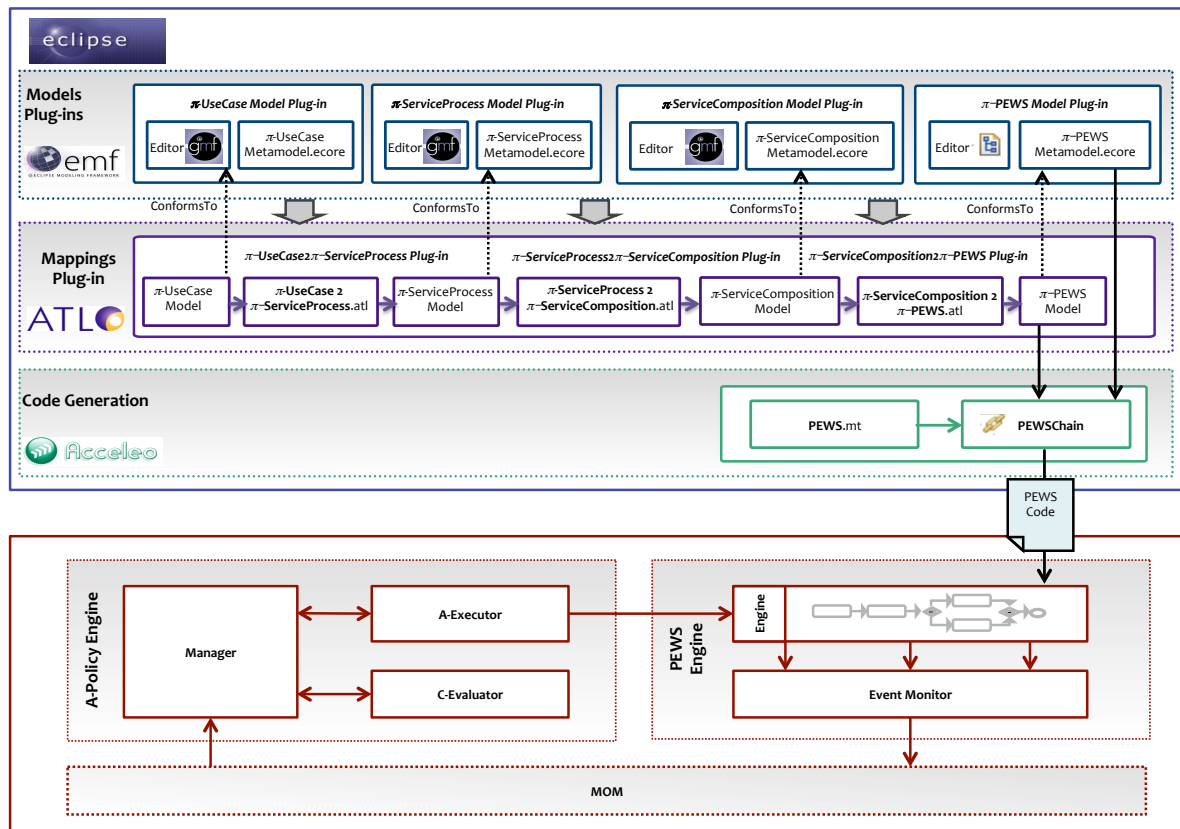


Figure 40: π SOD-M Development Environment.

The π SOD-M environment architecture is organized in three layers: (i) *Meta-model level*, (ii) *PIM-to-PIM Transformations level level*, (iii) *PIM-to-PSM Transformations level level* and (iv) *Code generation level*. Each layer comprises a set of components that together support the π SOD-M environment. This figure presents how those components are implemented by our tool: (i) the *Meta-model component* (figure 41) represents the *Model plugins module* of figure 40 (together with all methodology meta-models); (ii) the *PIM-to-PIM Transformations and PIM-to-PSM Transformations components* (figure 41) represent the *Mapping plugins module* of figure 40; (iii) the *Code Transformation component* of the figure 41 is implemented by the *Code generation module* of figure 40; and (iv) the *Execution engine* is represented by the lower tier of figure 40.

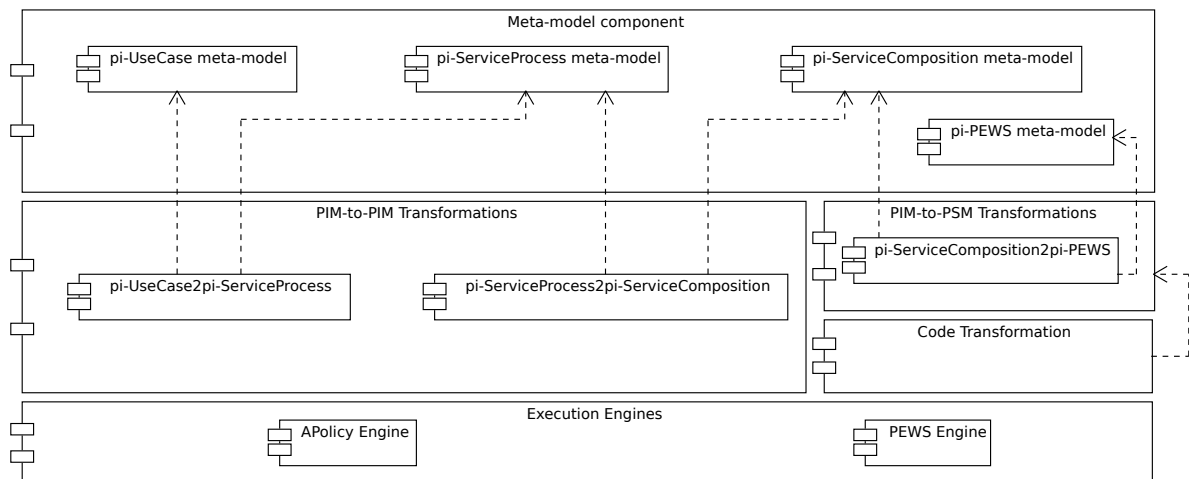


Figure 41: Environment Components.

The *Models plugin module* comprises the components that describe the π SOD-M meta-models, and how their models must be created. There are four meta-model components. All components of the *Mapping plugin module* depend of the definitions made in *Models plugin*.

When a model transformation is made, the models must comply with their respective meta-model. The process is executed as follows: every time a transformation is performed, a consistency check of both, source and target models is performed. After all transformation are made, the PSM model is translated into code of a particular platform, in the case of π SOD-M, π -PEWS is the chosen platform. The transformation of PSM model in code is the last stage of transformation. The component of the *Code generation module* depend of the PSM generated by the last model transformation. Finally, the *Execution Engine* component performs the execution of the service-based specification code.

4.1.1 Ecore Meta-models (*Models Plugin Module*)

The implementation of each meta-model is defined in *Ecore*³ files, they are: π -UseCase, π -ServiceProcess, π -ServiceComposition and π -PEWS. All meta-models have a related *genmodel*⁴ definition. The plugin editor can be created from the *genmodel* definition, so that models can be specified.

Using these set of tools (model editors) it is possible to create models at different π SOD-M

³The *Models Plugin Module* is a set of Ecore files that represents all π SOD-M meta-models. These meta-models are the sources for the development of each of the methodology's model. All models designed for an application must obey their meta-model specification. This module is composed by all the proposed methodology's models in EMF (*ecore extension*).

⁴A *genmodel* is a intermediary file format used to produce the syntax editor for each meta-model.

levels. There are editors for all π SOD-M models: π -*UseCase* editor, π -*ServiceProcess* editor, π -*ServiceComposition* editor and π -*PEWS* editor.

Although there are editors for each methodology model, it is still necessary components to perform transformations among them. Thus, the model specification process can be made in all methodology levels. However it can be made only at the highest level, *i.e.* π -*UseCase* model, and then execute automatic transformation to generate lowest level models.

4.1.2 Model Transformation (*Mapping Plugin Module*)

The *Model transformation level* has a set of components for processing and transforming models. The model transformation components are based on the source models for generating the equivalent target model. For example, from a π -*UseCase* model is generated the π -*ServiceProcess* model, from a π -*ServiceProcess* model is generated the π -*ServiceComposition* model, and finally, from a π -*ServiceComposition* model is generated a π -*PEWS* model. After the model generation the designer can perform refinements. Refinement can improve and adjust elements that require a more detailed description at this modeling level.

π SOD-M model transformation process is based on the rules described in chapter 3. The implementation process requires additional, more specific information to be taken into account. We had to consider the representation of our concepts in the Eclipse and ATL environments for MDA-based application development, *e.g.*, aspects of plugin generation; design model properties; compliance of the designed meta-model with the generated model; the specific model implementation; and etc.

We used the Eclipse Modeling Framework (EMF) for implementing the meta-models π -*UseCase*, π -*ServiceProcess*, π -*ServiceComposition* and π -*PEWS*. The ATL language was used for developing the mapping rules for transformation of models (π -*UseCase* 2π -*ServiceProcess*, π -*ServiceProcess* 2π -*ServiceComposition* and π -*ServiceComposition* 2π -*PEWS* plug-ins). This plugin takes as input a π -*PEWS* model implementing a specific service composition and it generates the code to be executed by the *Policy* based *Execution Engine* (figure 41).

Figures 42 and 43 present a general view of the π SOD-M models transformation, showing the set of plug-ins developed to implement it. Figure 42 presents the model-to-model transformations, while figure 43 shows the model-to-text transformation schema. The environment implements the abstract architecture shown in figure 40. This environment consists of plug-ins implementing the π -*UseCase*, π -*ServiceProcess*, π -*ServiceComposition* and π -*PEWS* meta-models used for defining models; and ATL rules for transforming PIM and PSM mod-

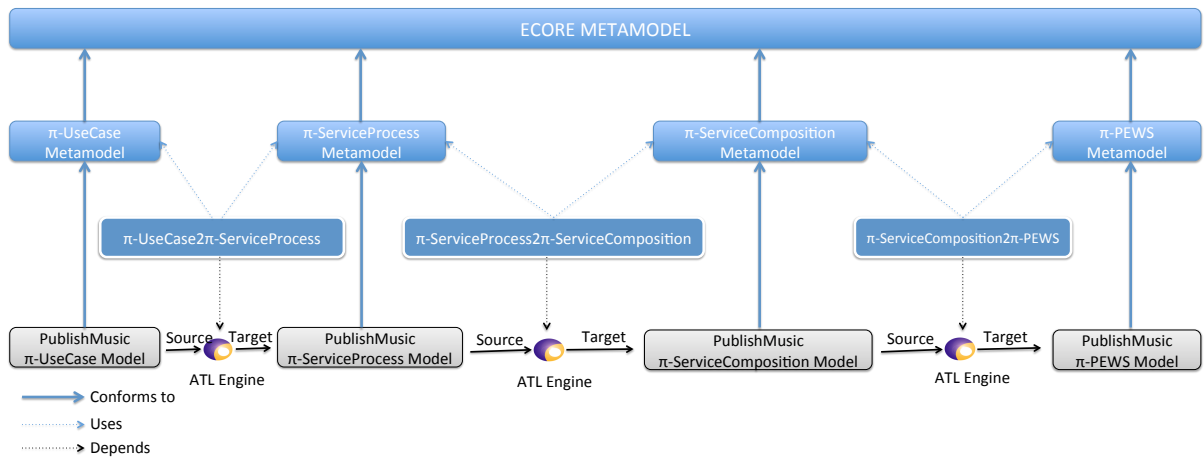


Figure 42: ATL Model to Model Transformation in π SOD-M.

els (model to model transformation) and finally generating code (model to text transformation) with Acceleo.

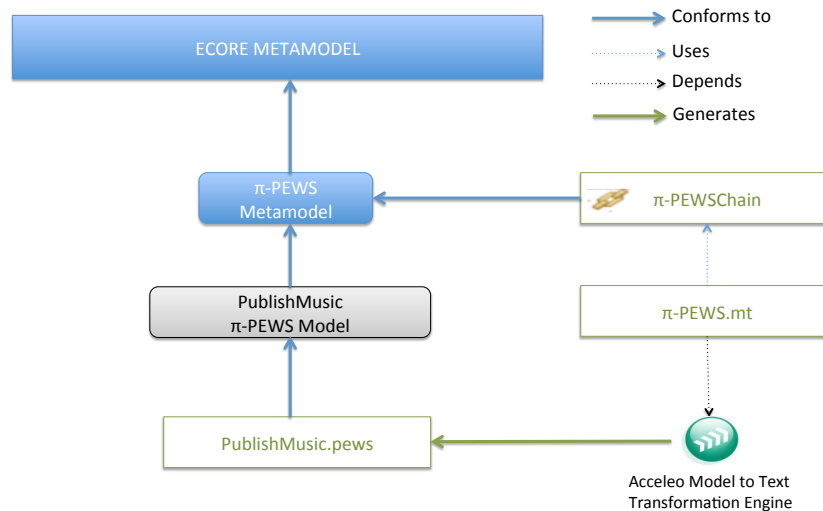


Figure 43: Acceleo Model to Text Transformation in π SOD-M.

In order to proceed with a model transformation it is necessary to configure the transformation environment. Figure 44 presents a standard screen for configuring each transformation (in this case for the “to publish music” scenario). From any source model, for example, π -UseCase, π -ServiceProcess or π -ServiceComposition, the system can perform the automatic transformation. Using the transformation tool of figure 44 requires from the user: (1) to indicate the ATL transformation rules file; (2) to choose the source meta-model reference (.ecore file); (3) to choose the target meta-model reference (.ecore file); (4) to choose the source model that will be transformed (e.g. music.piusecase file); (5) to choose the target model that will be generated (e.g. music.piserviceprocess file); and finally, (6) to run the tool. The same must be performed for all model-to-model transformations. For model-to-text transformation, the rule file to be

chosen should be the Acceleo file.

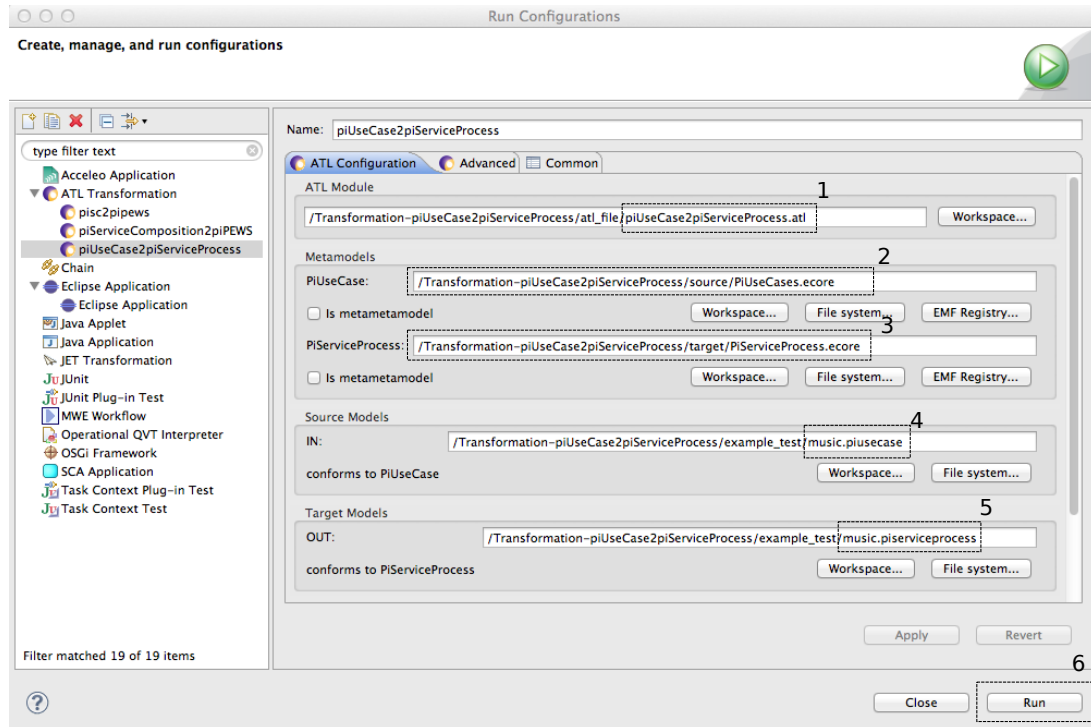


Figure 44: ATL Configuration for π SOD-M Transformation.

As an example, figure 44 shows the configuration for the transformation from π -UseCase (source) to π -ServiceProcess (target). Notice that, in this case, the reference meta-models must be the same. For further transformations, this process must follow the same sequence, changing only the models and reference meta-models.

4.1.2.1 π -UseCase2 π ServiceProcess Transformation Rules

The transformation rules describe how models are transformed. As a general rule, the automatic transformation of models favors a faster development of applications. In most cases, the environment or the designer should verify if the transformations are valid or not. In the π SOD-M environment, the consistency of the transformations must be performed by the designer. If a problem on the transformation is identified, the target model can be modified manually.

In ATL, there exist two different kinds of rules that correspond to the two different programming modes provided by ATL (e.g. declarative and imperative programming). The *matched*

Listing 4.1: ATL Example Rule.

```

1 rule ruleName{
2   from
3     var_sourceName: SourceModel!Entity
4   to
5     var_targetName: TargetModel!Entity (
6       attribute_1 <- var_sourceName.attribute_a ,
7       attribute_2 <- var_sourceName.attribute_b
8     )
9 }

```

Listing 4.2: ATL - piUseCase2piServiceProcess : useCase2action Rule.

```

1 rule useCase2action {
2   from usecase : PiUseCase!FunctionalBasicUseCase (
3     usecase.extend->size() >= 2
4   )
5   to sp_action : PiServiceProcess!Action (
6     name <- usecase.name
7   ), sp_serviceActivity : PiServiceProcess!ServiceActivity (
8     action <- sp_action ,
9     name <- usecase.name + 'SA' ,
10    serviceProcess <- thisModule.raiz_piUseCase
11  ), sp_contract: PiServiceProcess!Contract (
12    name <- usecase.name + 'Contract' ,
13    assertionInContract <- usecase.UCContract ,
14    action <- sp_action ,
15    serviceProcess <- thisModule.raiz_piUseCase
16  )
17 }

```

*rules*⁵ (declarative programming) and the *called rules*⁶ (imperative programming) [55]. For the π SOD-M environment, we use both types of rules. Listing 4.1 shows a general example of an ATL *matched rule*. According to this, we will present the main ATL rules used to implement the model transformation. As described in listing 4.1, a rule consists of a name (line 1), a source entity (*from* clause in lines 2-3), and one or more target entities (*to* clause in line 4-5). Each entity has a name, such as *var_sourceName* and *var_targetName*. The transformation is performed by each entity attribute and general rules established in each case.

⁵The matched rules constitute the core of an ATL declarative transformation since they make it possible to specify 1) for which kinds of source elements target elements must be generated, and 2) the way the generated target elements have to be initialized. A matched rule is identified by its name. It matches a given type of source model element, and generates one or more kinds of target model elements. The rule specifies the way generated target model elements must be initialized from each matched source model element[55].

⁶The called rules provide ATL developers with convenient imperative programming facilities. Called rules can be seen as a particular type of helpers: they have to be explicitly called to be executed and they can accept parameters. However, as opposed to helpers, called rules can generate target model elements as matched rules do. A called rule has to be called from an imperative code section, either from a match rule or another called rule[55].

Listing 4.3: ATL - piUseCase2piServiceProcess : constraint2contract Rule.

```

1 rule constraint2contract{
2   from
3     constraint: PiUseCase!UseCaseConstraint
4   to
5     assertion: PiServiceProcess!Assertion(
6       name <- constraint.name,
7       description <- constraint.description
8     )
9 }

```

Listing 4.4: ATL - piServiceProcess2piServiceComposition : root Rule.

```

1 rule root {
2   from
3     root: piServiceProcess!ServiceProcess
4   to
5     root_service: piServiceComposition!CompositionServiceModel(
6       activities <- root.activity ,
7       edges <- root.edge ,
8       compositionPolicies <- root.contract
9     )
10 }

```

Listing 4.2 shows the ATL transformation rule between a USE CASE and an ACTION, while listing 4.3 shows the transformation rule between CONSTRAINT and ASSERTION. There is a restriction in the rule presented in listing 4.2 (line 3), which defines that there must be more than two EXTEND relation between use cases. In this case, all use cases are transformed into actions, and are grouped into a SERVICE ACTIVITY (lines 7-10). And all the constraints are transformed in a set of assertions that are automatically associated with a CONTRACT (lines 11-15). This transformation (listing 4.2) describes that from a USE CASE (lines 2), ACTIONS, SERVICE ACTIVITY and CONTRACT can be generated (lines 5, 7 and 11). It depends on the existence of CONSTRAINT or dependence elements (INCLLUDE or EXTEND) related to this use case (line 3).

The listing 4.3 describes that all CONSTRAINT (*from* clause in lines 2-3) are directly transformed into an ASSERTION (*to* clause in lines 4-5) (there is no restriction in the this clause). The restrictions are treated in separate rules such as in listing 4.2.

4.1.2.2 π -ServiceProcess2 π -ServiceComposition Transformation Rules

As the π -ServiceComposition model is a refinement of some concepts defined in π -ServiceProcess, such as CONTRACT and ACTIVITY SERVICES, most of the transformation between these models are through direct transformation, without restrictions. We present the ATL transformation (listing 4.4) of the root element that comprises the main elements of both models.

Listing 4.4 shows the transformation rule for the main elements from π -ServiceProcess to π -ServiceComposition model. The ACTIVITIES and its EDGES, and CONTRACTS, which are transformed into POLICY (lines 6-8). Other rules that describe the transformation between ASSERTIONS into policy RULE are other type of transformation. However, other rules describe that all CONTRACTS must belong to a specific NON-FUNCTIONAL REQUIREMENT, for example, all contracts for the performance restrictions, will be grouped into a single performance POLICY. This listing is an ATL example for π -ServiceProcess model transformation into to

Listing 4.5: ATL - piServiceComposition2piPEWS : root Rule.

```

1 rule root{
2   from sCM : pisc!CompositionServiceModel
3   to
4     path: pipews!Path (),
5     pews : pipews!PEWSCTSspec (
6       name <- 'newModelPEWSpecName',
7       has <- path,
8       contains <- sCM.partition,
9       defines <- thisModule.policies
10    )
11 }
```

π -ServiceComposition model.

4.1.2.3 π -ServiceComposition2 π -PEWS Transformation Rules

The transformation π -ServiceComposition2 π -PEWS is unique among PIM and PSM levels, however there is no difference in the rules description in ATL, since all rules are defined in terms of meta-models rather than specific models. Thus, PIM-to-PSM transformation rules in ATL follow the same syntax and semantics of the PIM-to-PIM transformation rules.

The model to be generated in this transformation, namely π -PEWS model, will be used as input for the code generation component, so that the application code specification is generated. The listings 4.5 and 4.6 present two transformation rules in ATL. The first describes the transformations of the main elements for the description of a service composition, the main path, the name of the specification, the services and policies (lines 6 - 9), while listing 4.6 describes a policy RULE and its related concepts, such as ACTION, EVENT and CONDITION. The ATL rule has a constraint to be checked (line 4), what kind of RULE is being translated for the specific language, because depending on the type, the transformation will be change. The RULE element (line 15) consists of all the properties necessary to create a PRE-CONDITION, such as ACTION, EVENT and CONDITION (lines 16-18), and to which POLICY the PRE-CONDITION is related (line 19).

4.1.3 Code Generation (*Code Generation Module*)

The π SOD-M methodology and its environment use π -PEWS⁷ as intermediate language for supporting the generation of executable code. π -PEWS is a behavioral language for defining web services interfaces by specifying the input/output interface of a web service exported method but also the expected behaviour of its methods (e.g., dependencies, exchanged messages). The extension π -PEWS extension was inspired by languages like JCML [36], JML [65]

⁷Other meta-models and transformations can be defined for supporting code generation into WS-BPEL, XAML and other languages adapted for describing business/service process.

Listing 4.7: Acceleo - Namespace and Operation Code Specification.

```

1 <%script type="PiPEWSMetamodel.PEWSSpec" name="default"
2 file="<%name%>.pews"%>
3 #-----
4 #----- <%name%>.pews Service Specification -----
5 #-----

7 //Namespaces
8 <%for (contains){%>
9 namespace <%name%> = <%WDSLAdress%>
10 <%}%>

12 // Operations
13 <%for (has.eAllContents("Operation")){%>
14 alias <%alias%> = portType/<%name%> in <%isDefinedIn.name%>
15 <%}%>

```

Listing 4.8: Acceleo - Service Code Specification.

```

1 // Services
2 <%for (has.eAllContents("CompositeOperation")) {%>
3 service <%name%> = <%contains.left.filter("Operation").alias%>
4 <%if ((contains.nameOperator.toString().equalsIgnoreCase("sequence"))){%>
5
6 <%}else if((contains.nameOperator.toString().equalsIgnoreCase("parallel"))){%>
7 ||
8 <%}%>
9 <%contains.right.filter("Operation").alias%>
10 <%}%>

```

celeo specification. This figure shows the specification for the *Namespace*, *Operation*, *Service* and *Contract* code generation for π -PEWS. After the code transformation process, a *.pews* file is created. The listing 4.7, 4.8 and 4.9 present parts of Acceleo code for the π -PEWS code generation. The code are the same presented in figure 45, which presents the relation between the meta-model concepts and the Acceleo code. The code generation follow the language syntax described in appendix C.

In listing 4.7 (lines 1-2) references the meta-model, the root element (*PEWSSpec*), and the name of the generated file (*<%name%>.pews*). Lines 3-5 presents the name of the service specification. Lines 7-10 describes a *for* integration in the *contains* relationship between *PEWSSpec* and *Namespace*, and lines 12-15 all operations defined in each *Namespace* is generated. This operations came from the *isDefinedIn* relationship between *Namespace* and *Operation* entities.

Listing 4.8 presents the Acceleo specification for the π -PEWS model transformation into code. A service is an alias for one or more operations. Listing 4.9 specifies the contract generation, using the *defines* relationship between *APolicy* and *Rule*. Each contract have a set of rules for the specification.

The generated code can be executed in both, π -PEWS and A-Policy engines⁸ These 2 engines are not native components in the π SOD-M plugin. The environment supports the process design to generate code in any language. New language editor components, like BPEL or

⁸ Both engines are currently under development. The π -PEWS engine is being developed at UFRN and the A-Policy engine in being developed at Grenoble/France.

Listing 4.9: Acceleo - PEWS Contract Code Specification.

```

1 // Contract
2 <%for (defines) {%>
3   defContract <%name%>Contract{
4     isAppliedTo: <%isRelatedWith.filter("Operation").alias %>;
5     <%for (defines.filter("Precondition")) {%>
6       requires: <%defines.expression%>
7         (OnFailureDo: <%calls.act.toString()%>);
8     <%}%>
9     <%for (defines.filter("Postcondition")) {%>
10      ensures: <%defines.expression%>
11        (OnFailureDo: <%calls.act.toString()%>);
12    <%}%>
13    <%for (defines.filter("TimeRestriction")) {%>
14      timeConstraints: <%defines.expression%>
15    <%}%>
16 }

```

XAML, can be easily coupled to the environment. Therewith, it is necessary to add the language meta-model and make the transformation process. Thus, from a π -*ServiceComposition* model, different models and codes can be generated (not only π -PEWS). This requires only the definition of equivalent meta-models, and the corresponding code transformation rules.

The composition engine manages the life cycle of the composition. Once a composition instance is activated, the engine schedules the composition activities according to the composition control flow. Each activity is seen as a process where the service method call is executed. The execution of an activity has four states: prepared, started, terminated, and failure. The execution of the control flow (sequence, and/or split and join) can also be prepared, started, terminated and raise a failure.

At execution time, the evaluation of policies done by the POLICY manager must be synchronized with the execution of the services composition (i.e., the execution of an activity or a control flow). Policies associated to a scope are activated when the execution of its scope starts. A POLICY will have to be executed only if one or several of its rules is triggered. If several rules are triggered, the *Policy* manager first builds an execution plan that specifies the order in which such rules will be executed according to the strategies defined in the following section. If rules belonging to several policies are triggered then policies are also ordered according to an execution plan. The execution of policies is out of the scope of this thesis, the interested reader can refer to [47] for further details.

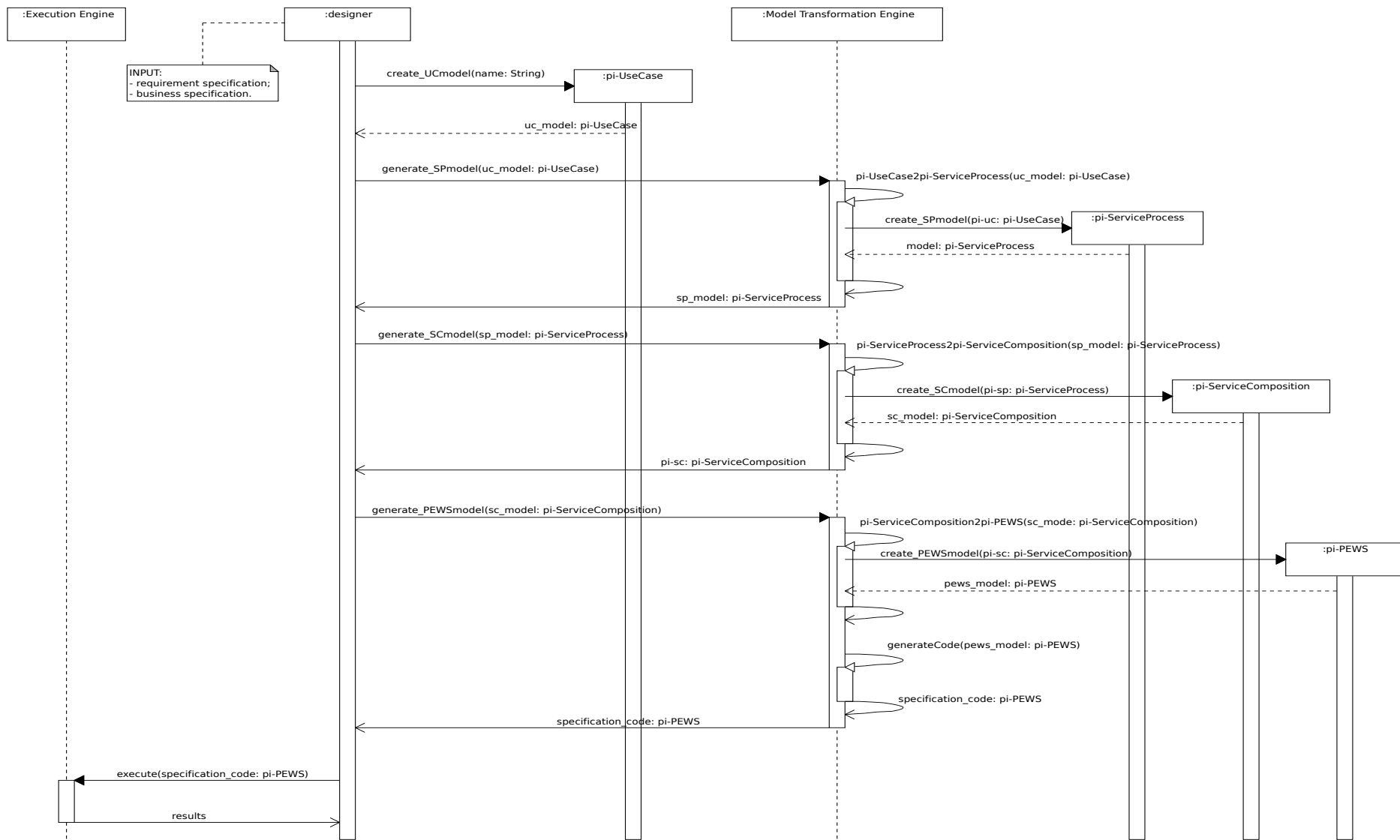


Figure 46: Model Transformation Process.

4.2 Defining Reliable Service Based Applications

The π SOD-M environment development starts with the creation of a project and then the definition of a π -UseCase model⁹, supposing that the business and requirement specification document have been previously completed. Figure 47 presents the views provided by the environment: *Project view*, *Menu view*, *Editor view* and *Properties view*.

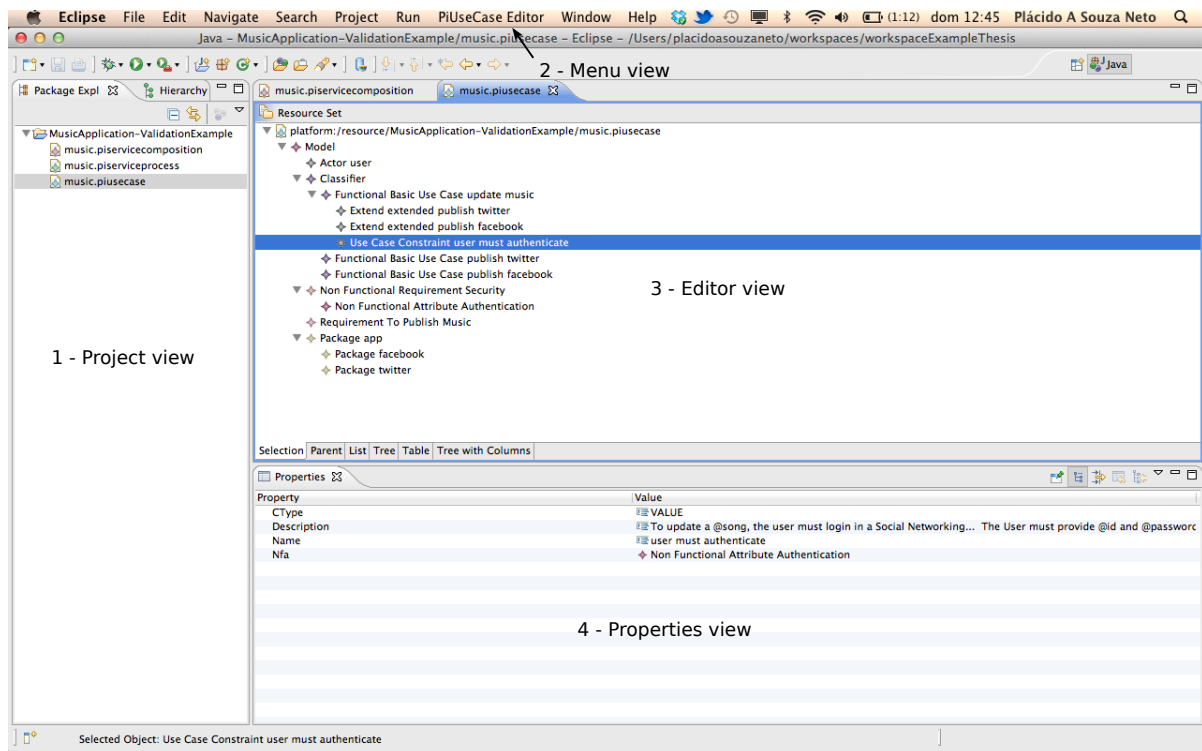


Figure 47: π SOD-M Eclipse Plugin Environment.

4.2.1 π -UseCase Model

The goal of creating a π -UseCase is to represent the functions and system services described in the requirements specification and business specification documents, which are the requirements input for this phase. In accordance to the process described in figure 46, the designer receives the documents as input and creates the π -UseCase model. With this model created, the transformation component generates the π -ServiceProcess model as output.

To create the π -UseCase model, it is necessary choose the root element (*Model*) as the starting point of modeling, during the process of create a “new .piusecase file” (using the sequence, *File > New > Other > EMF Model Wizard*). From this point the model is created as

⁹To create a model, the user must execute the sequence: *File > New > Other > EMF Model Wizard*, and choose one of the methodology’s model.

a tree of elements with its specific references, as shown in Figure 48, which shows the model elements and its equivalence in the graphical π -UseCase model, each element is built in an iterative way and must obey the hierarchy and its relationships.

Each model element has a number of childs or siblings to which it is related to, *e.g.*, an USE CASE relates CONSTRAINT, EXTEND and INCLUDE elements, as well as NFRs relates NFAs. Figure 48 shows how to create the model elements for the “to publish music” scenario.

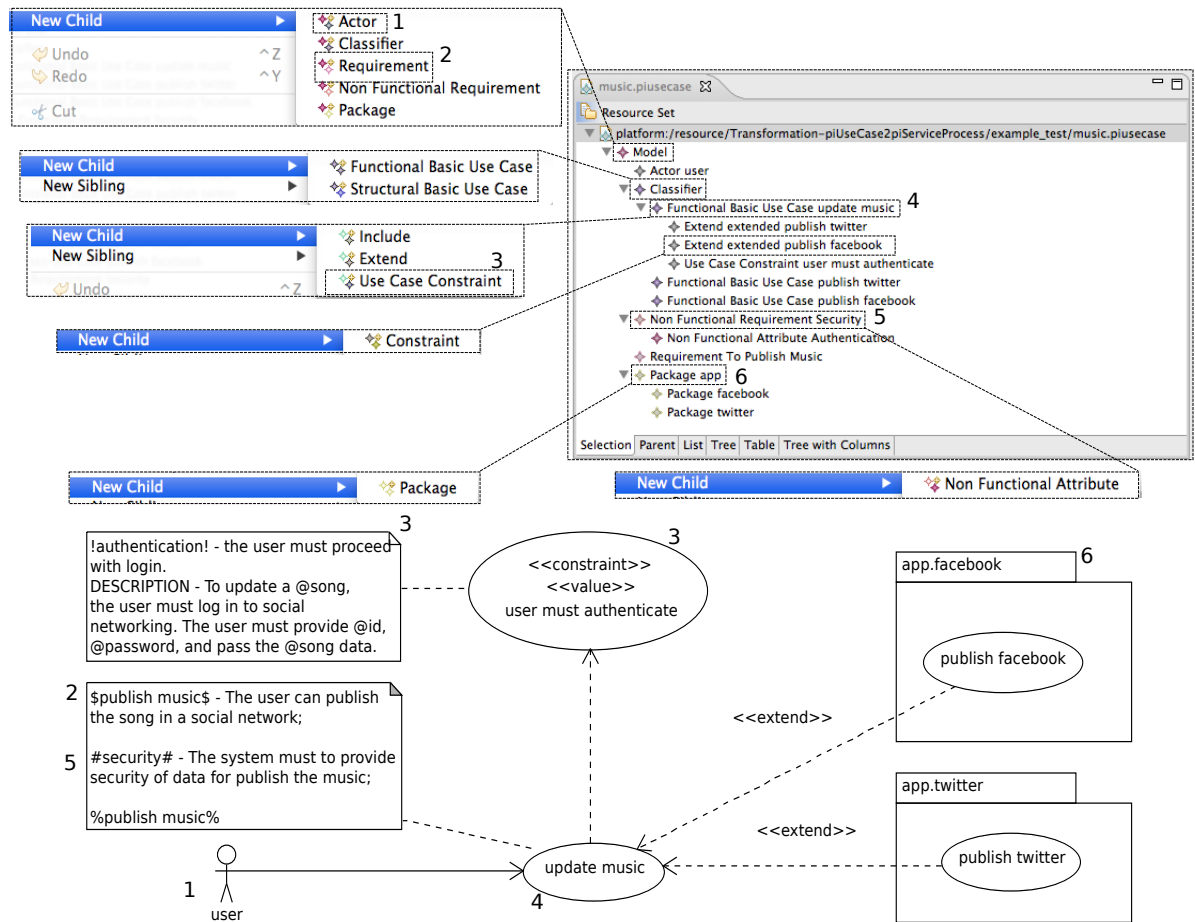


Figure 48: π -UseCase Model Definition in π SOD-M Eclipse Plugin.

Example: Items 1 and 2 in Figure 48 show how to create an ACTOR and a REQUIREMENT, item 3 presents the *update music* USE CASE and how to create a CONSTRAINT related to this element. Item 4 makes reference to a CONSTRAINT and finally, the items 5 and 6 are equivalent to a NFR and a PACKAGE elements, respectively. Each item in the model refers to a equivalent element in the graphical model. □

Figure 49 presents some configuration properties. After creating an element, it is necessary to set its properties. All elements have properties that describes their relationship and

specification values. These properties are used to give specification details, as well as future transformations.

Example: Item 1 in Figure 49 shows the USE CASE element properties for the “to publish music” application. After the creation of a use case it is necessary to create the actor, its *name*, which *requirement* the *use case* belongs, and the respective *package*. Item 2 shows the properties of a CONSTRAINT element. In a constraint, its type must be explicit (*VALUE*, *BUSINESS* or *EXCEPTIONALBEHAVIOUR*), its *description*, with all candidate *variables* described with an ‘@’, its *name* and which NON-FUNCTIONAL ATTRIBUTES element this constraint is related to, e.g., *Authentication*, is an attribute of the NON-FUNCTIONAL REQUIREMENTS, *Security*. Items 3 and 4 show the “To Publish Music” requirement properties and package “app”, respectively. The properties of each element obey the elements described in the π -UseCase meta-model.

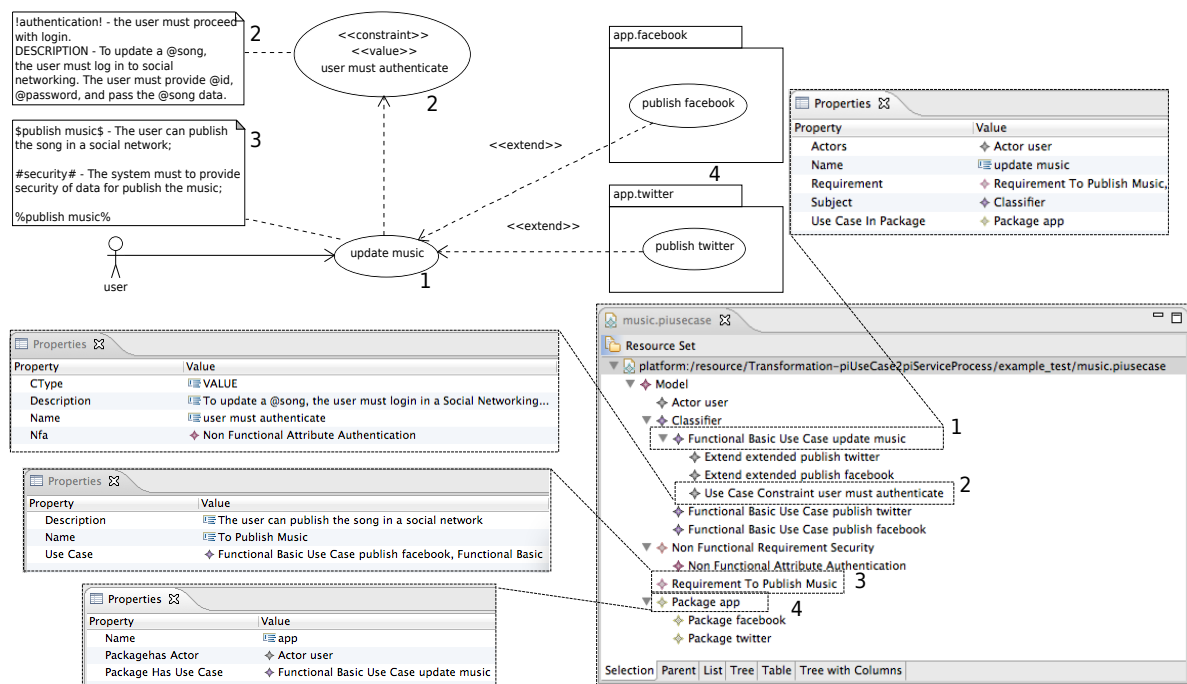


Figure 49: π -UseCase Properties in π SOD-M Eclipse Plugin.

4.2.2 π -ServiceProcess Models

The goal of creating a π -ServiceProcess model is to represent the system execution flow. The designer receives the π -UseCase model as input to generate the π -ServiceProcess model. After the π -ServiceProcess model be generated, the designer calls again the model transformation component to generate the π -ServiceComposition model as output (figure 46).

To create the π -*ServiceProcess* model, it is necessary choose the root object (*Service Process*) as the starting point of modeling, during the process of create a “*new .piserviceprocess file*” option. From this point on, the model is created as a tree with its specific references.

As this model is a refinement of the concepts described in the previous model, its information is part of the information and properties of the π -*UseCase* model, but they focus on the process workflow and application functions.

Example: Figure 50 shows the complete model of the example scenario and its main components. From the root element the user can create the following elements: SERVICE ACTIVITY, OBJECT FLOW, CONTROL FLOW, CONTRACT and NON-FUNCTIONAL ATTRIBUTE. In Figure 50, item marked with 1 highlights the creation of the CONTROL FLOW . Items 2 and 3 show the elements FORK NODE and JOIN NODE, respectively, essential for the execution flow description. Items 4 show the ACTION element, which describes the *download music*, *listen music* and *publish twitter* actions and finally, items 5 highlight the ASSERTION element, which is used to describe the restrictions over each ACTION. □

The designer must also configure the properties of each π -*ServiceProcess* model element to complete it by specifying all constraints that compose the contracts and the application process. Figure 51 presents the properties of the main model elements for our example scenario. These properties complement the modeling of the execution flow.

Example: Items 1 in Figure 51 describe the properties of a FORK NODE, which is associated with EDGE ACTIVITIES (CONTROL FLOW and OBJECT FLOW) that connect the ACTION elements. For example, the FORK NODE *fn_toPublishMusic* has an input edge (*cf1*) and three output streams edges (*cf2*, *cf3* and *cf5*). All these elements are CONTROL FLOW entities (items 3). Items 2 show the ACTION properties, and items 4 and 5 describe a CONTRACT and its ASSERTIONS properties, respectively. A CONTRACT has a *name* and the information of an ACTION. ASSERTIONS are CONTRACT’s child element. Each CONTRACT can have many ASSERTIONS. Each ASSERTION has six properties, they are: (i) *AProperty*, which describes the runtime verification execution time, (ii) *description*, (iii) *variable name*, (iv and v) a maximum and minimum allowed to be checked (*MaxValue* and *MinValue*) and the variable (vi) *type*. In cases of non numeric types, *e.g. Short Int, Flot and Double*, only *MinValue* property value is considered property and *MaxValue* is described as *null*. □

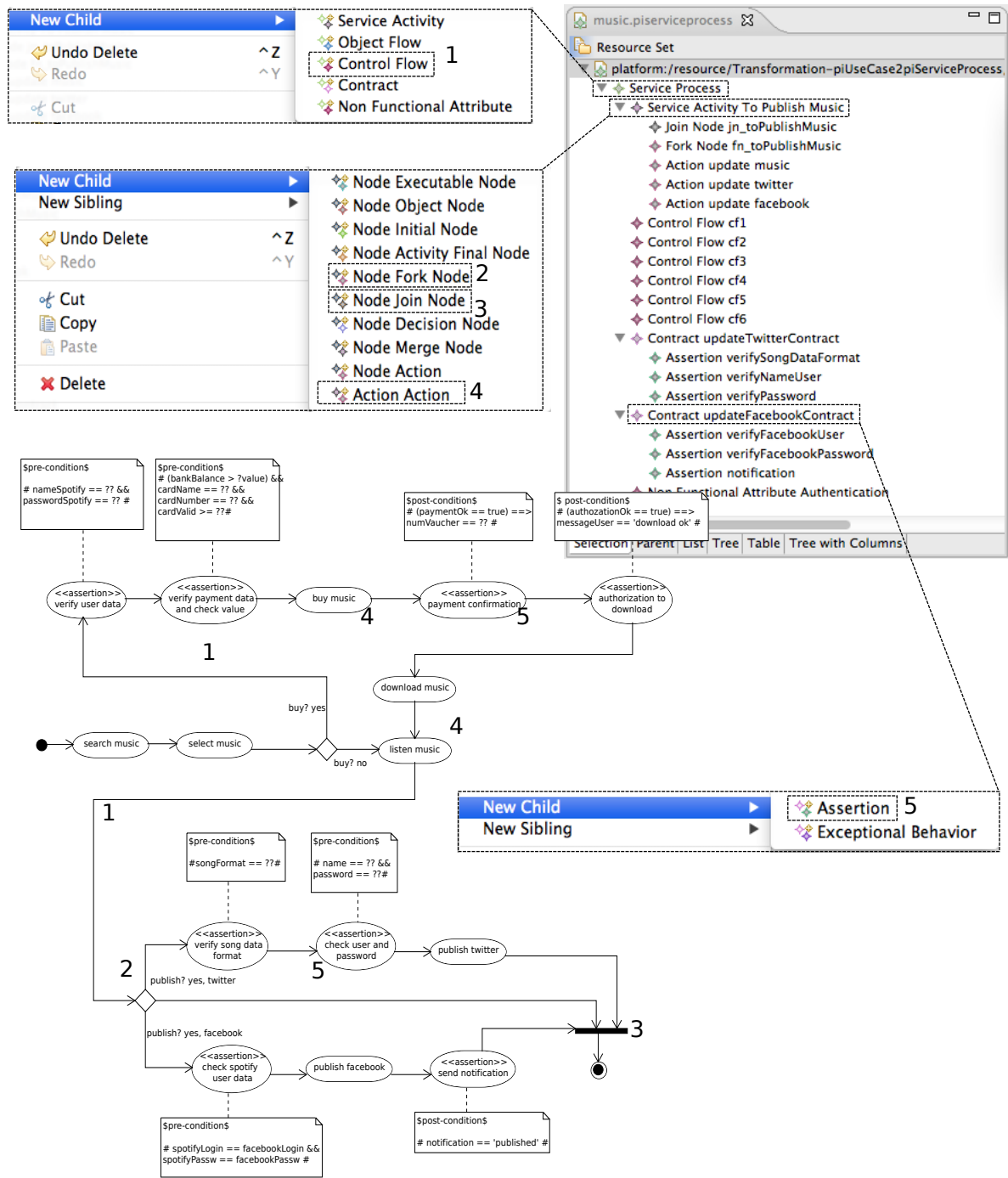


Figure 50: π -ServiceProcess Model Definition in π SOD-M Eclipse Plugin.

4.2.3 π -ServiceComposition Models

The goal of creating a π -ServiceComposition model is to represent the system service compositions and to reference the external services that are called by the application. The designer receives the π -ServiceProcess model as input to generate the π -ServiceComposition model. After the π -ServiceComposition model is generated, the designer calls again the model transformation component to generate the π -PEWS model as output (figure 46). The output model

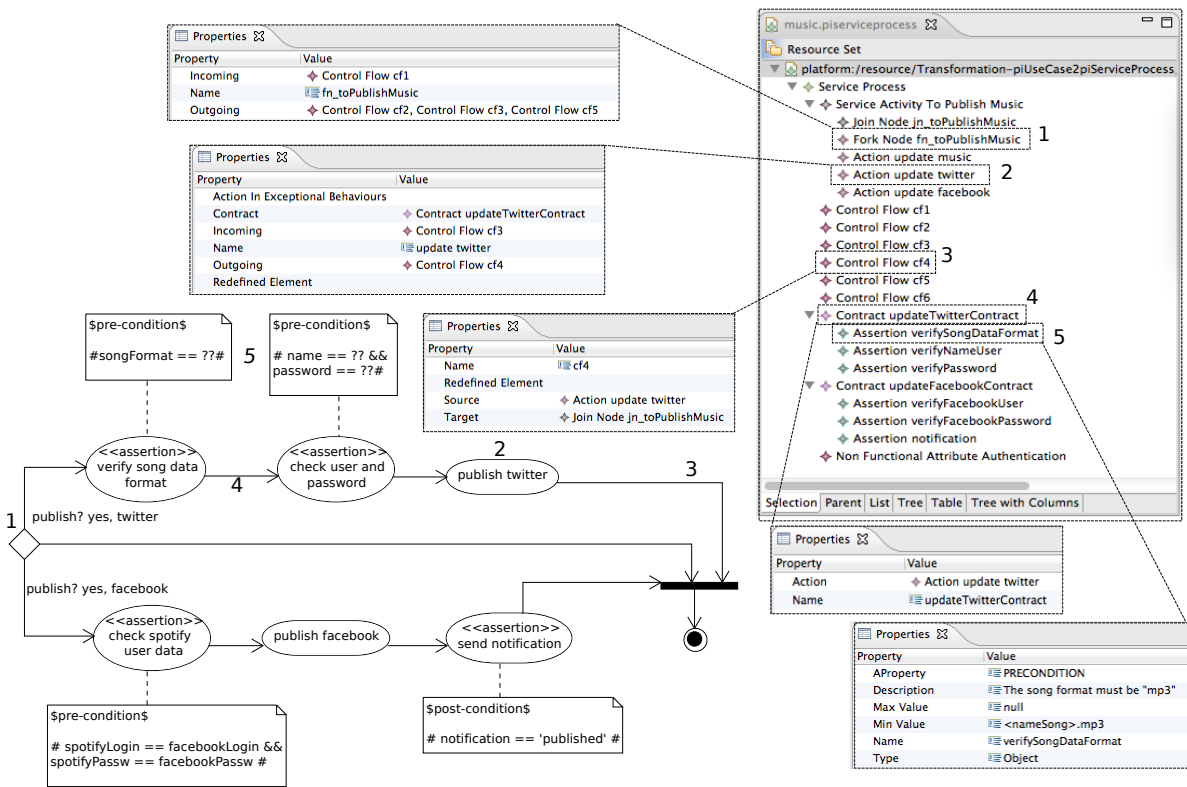


Figure 51: π -ServiceProcess Properties in π SOD-M Eclipse Plugin.

describes the system specification in a specific platform.

To create the π -ServiceComposition model, it is necessary to choose the root object (*Composition Service Model*) as the starting point of modeling, during the process of creating a “new .piservicecomposition file” option. From this point on, the model is created as a tree with its specific references.

Recall that the π -ServiceComposition model is a refinement of π -ServiceProcess model, most elements are the same, except for BUSINESS COLLABORATOR, POLICY, RULES and VARIABLE. Thus, we will detail these elements in the model editor description.

Example: Figure 52 shows the relationship of these elements for our example scenario. Each ACTION has an specific BUSINESS COLLABORATOR (item 3), this elements express the external service provider, such as Facebook, Twitter or Bank. Another example of a BUSINESS COLLABORATOR definition is a WSDL specification. The WSDL namespaces represents a kind of BUSINESS COLLABORATOR. Besides BUSINESS COLLABORATOR element, the main children root element (*Composition Service Model*) are: POLICY and SERVICE ACTIVITY (items 1), and each POLICY (items 2) is directly related with SERVICE ACTIVITIES. From a POLICY it is possible to create a set of RULES (item 4). Figure 52 also presents an equivalent graphic model of the application. In this figure, the *Application’s* BUSINESS COLLABORATOR presents

only the main ACTIONS (*buy music, listen music, publish Facebook* and *publish twitter*) that are related with a SERVICE ACTIVITIES that have one POLICY. The *Application's* BUSINESS COLLABORATOR model is the same presented in the previous section (π -*ServiceProcess* model).

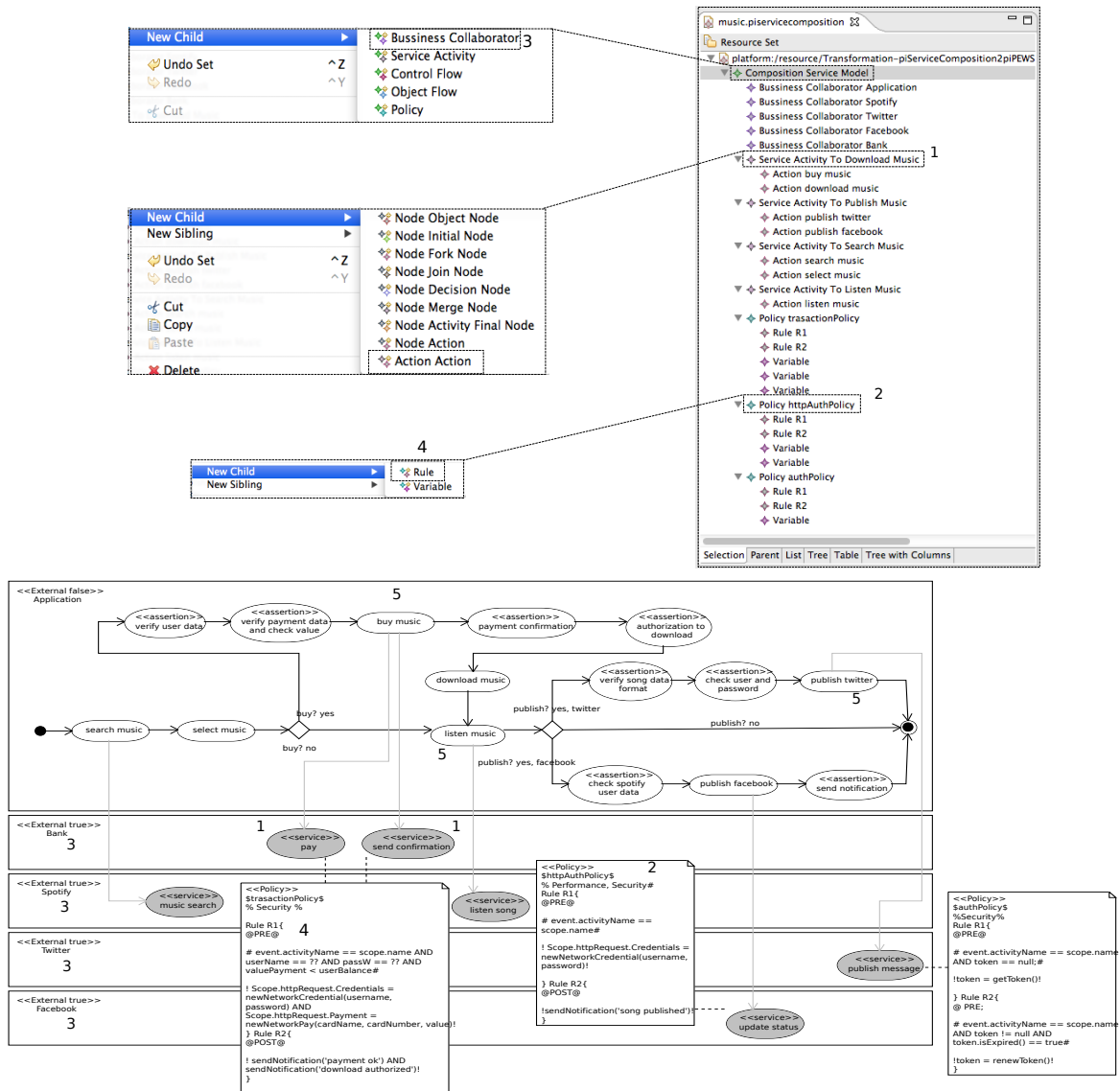


Figure 52: π -*ServiceComposition* Model Definition in π SOD-M Eclipse Plugin.

The properties of π -*ServiceComposition* elements are configured as the same way the other two previous editors, and the properties described in the π -*ServiceComposition* meta-model (Figure 28). Thus, to configure the properties of this model, it is necessary simply choose the desired element and to modify its values.

4.2.4 π -PEWS Models

The goal of creating a π -PEWS model is to represent the application in a specific platform. The designer receives the π -ServiceComposition model as input to generate the π -PEWS model. After the π -PEWS model be generated, the designer calls again the model transformation component to generate the π -PEWS specification code as output (figure 46). This code will be executed.

To create the π -PEWS model, it is necessary to choose the root object (*PEWS Spec*) as the starting point of modeling. From this point on, the model is created as a tree with its specific references.

The π -PEWS meta-model is a representation of the π -PEWS language. Each π -PEWS model is a program/specification written in that language. The elements described in this model represent parts of language's grammar constructs. Each entity in the model represents a piece of π -PEWS code.

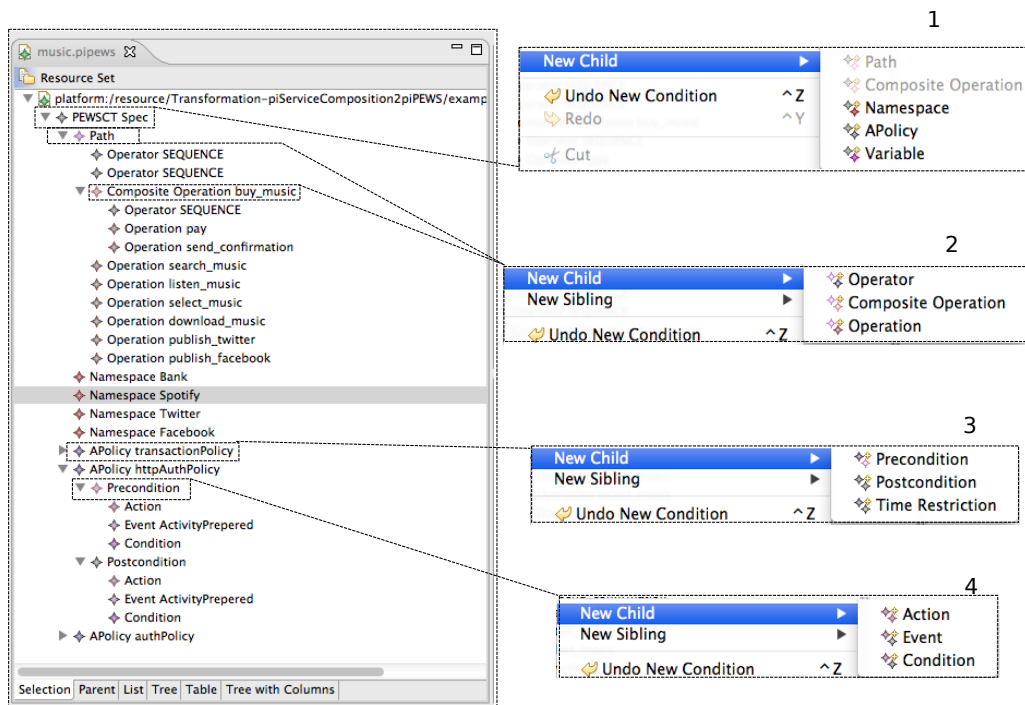


Figure 53: π -PEWS Model Definition in π SOD-M Eclipse Plugin.

Figures 53 and 54 show how the elements that compose a π -PEWS model can be specified in our tool. This model is generated automatically from the π -ServiceComposition model.

π -PEWS is the last model generated before the code generation. At this stage, it is expected that the designer proceeds with a general (manual) check of consistency of the model. It is

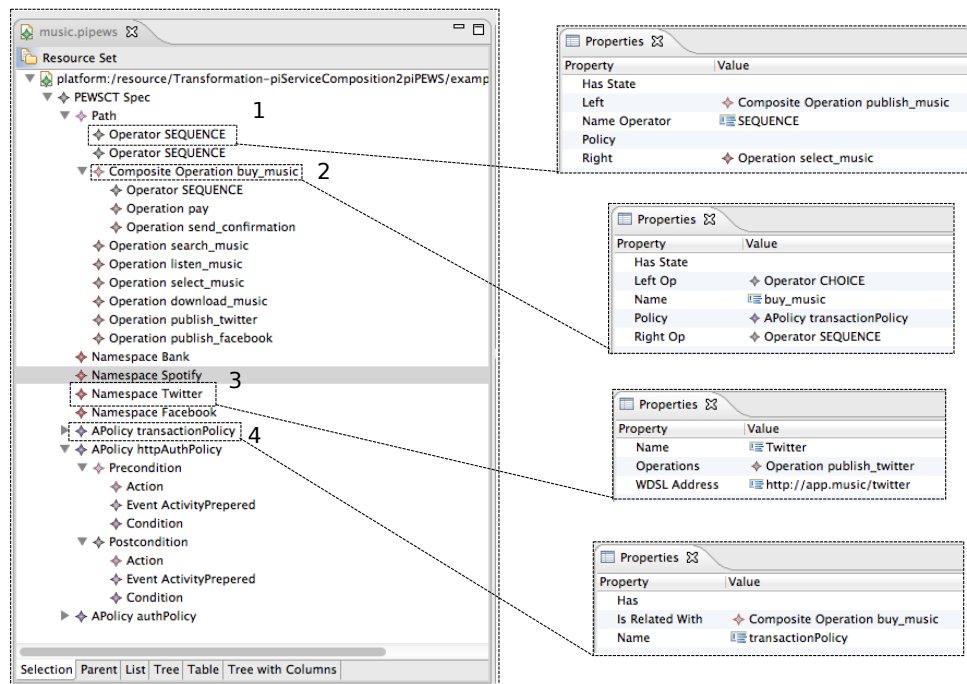


Figure 54: π -PEWS Model Properties in π SOD-M Eclipse Plugin.

important to remark that the environment alone does not replace the modeling work required to design and develop services based applications.

4.3 Extending the Environment

Both, the π SOD-M environment and methodology can be extended, in order to improve the components that describe and implement the methodology. Extensions can be done in two different levels: (i) adding new models to the existing infra-structure, and (ii) considering more abstract levels.

The extension may be in terms of language: new meta-models for other languages can be described and coupled to the environment. The extension process should take place as follows: new languages meta-models may be designed and coupled in the environment architecture (such as BPEL, XAML, WSDL and XLANG). After creating the desired meta-model, a mapping must be done. The mapping must respect the π -ServiceComposition meta-model. It is also necessary to describe the rules for code generation, using a code generator engine such Acceleo.

Considering the more abstract level of the methodology, new meta-models can be described. For instance, the computing independent level (CIM level) of SOD-M may be added to our methodology to describe the requirements and business restrictions in terms of models.

4.4 Conclusion

This chapter introduced the implementation of the π SOD-M methodology environment. We also presented a representation for our model description. The implementation includes all (i) meta-models of π SOD-M, (ii) editors for each model for the applications being developed, and (iii) the plugins for the transformations defined by the methodology. An example was also presented to describe the environment features.

5 *Evaluation*

“Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”

Donald E. Knuth

This chapter describes the experimentation we conducted for evaluating the π SOD-M methodology. The chapter presents three examples. The first two examples concern simplified service-based applications that focus on the application’s logic and explicit business rules. They show how π SOD-M can be used for developing applications and exemplify the consideration of non-functional aspects along the whole development process. The last example (*GesIMED* [39] application) aims to perform an evaluation of our approach, analyzing the main differences and particularities between SOD-M and π SOD-M.

The objective of the experimentation was to evaluate the features and concepts proposed by π SOD-M, considering the implementation of some examples, and perform a comparison with SOD-M. This comparison focuses on verifying which of those proposed aspects favored the development of service-oriented applications in the presence of non-functional requirements. In this direction, our intention is: (i) analyze if it is possible to model non-functional requirements from initial levels of abstraction in the service-oriented development, (ii) and verify what are the advantages of using the proposed approach, compared with the original proposal of SOD-M.

This chapter is organized as follows: section 5.1 shows the *To Listen Music* example, section 5.2 introduces the example *Crime Map* and section 5.3 introduces the *GesIMED* example that we used for validating the methodology. Section 5.4 presents the lessons learned with the validation process. Finally, Section 5.6 concludes the chapter and discusses on validation results.

5.1 Example 1: To Publish Music

An organization wants to provide the service-based application that monitors the music a person is listening during some periods of time and sends the song title to this person's Twitter or Facebook accounts. Thus, this social network user will have her status synchronized in Twitter or Facebook. The scenario begins by get through the music service Spotify for fetching the user's musical status. The social network services are then contacted to modify the user's status with the music information. The user can also download a specific music, he can proceed with the download process, which includes the payment process.

The user can download a music, after having proceed with the payment via PayPal or credit card. All services are available via Spotify and this application needs to interact with Spotify users, so they can listen music, publish them on Facebook and Twitter, or buy music online. The scenario to monitor and publish songs has the User and Spotify actors.

The functionality of the application are: *Search music*; *Choose music*; *Download music*; *Listen Music*; *Buy music* and *Publish music*.

5.1.1 π -UseCase Model

Figure 55 shows the main part of the use case model for the *To Publish Music* application¹. The π -UseCase model describes the system functions and constraints. Figures 56 and 57 detail the model and configuration properties for the described example (actual view taken from the π SOD-M environment). Three restrictions are specified on the use cases described by the model, two on the *buy music* use case and a restriction on the *pay* use case. These restrictions are value and business constraints. *Buy music* restrictions are related to secure connection and to ensure that the user needs to have an account in the music network. They are marked using #...#, figures 55, 56 and 57. Notice that those NFRs and constraints will be the base to define assertions and contracts by the transformations of models.

Figures 56 and 57 present the update music model and show how the properties are defined in the π SOD-M environment. This example describes the properties over the *update music* (item 4, figure 56) use case and the restriction over the authentication process (item 3, figure 56). To update a music in a social network, the user must be authenticated.

The modeling of the application properties using the environment π SOD-M are the same presented in the π -UseCase model, as described in Figure 57. For example, the authentication

¹The complete π -UseCase model can be found in appendix D.

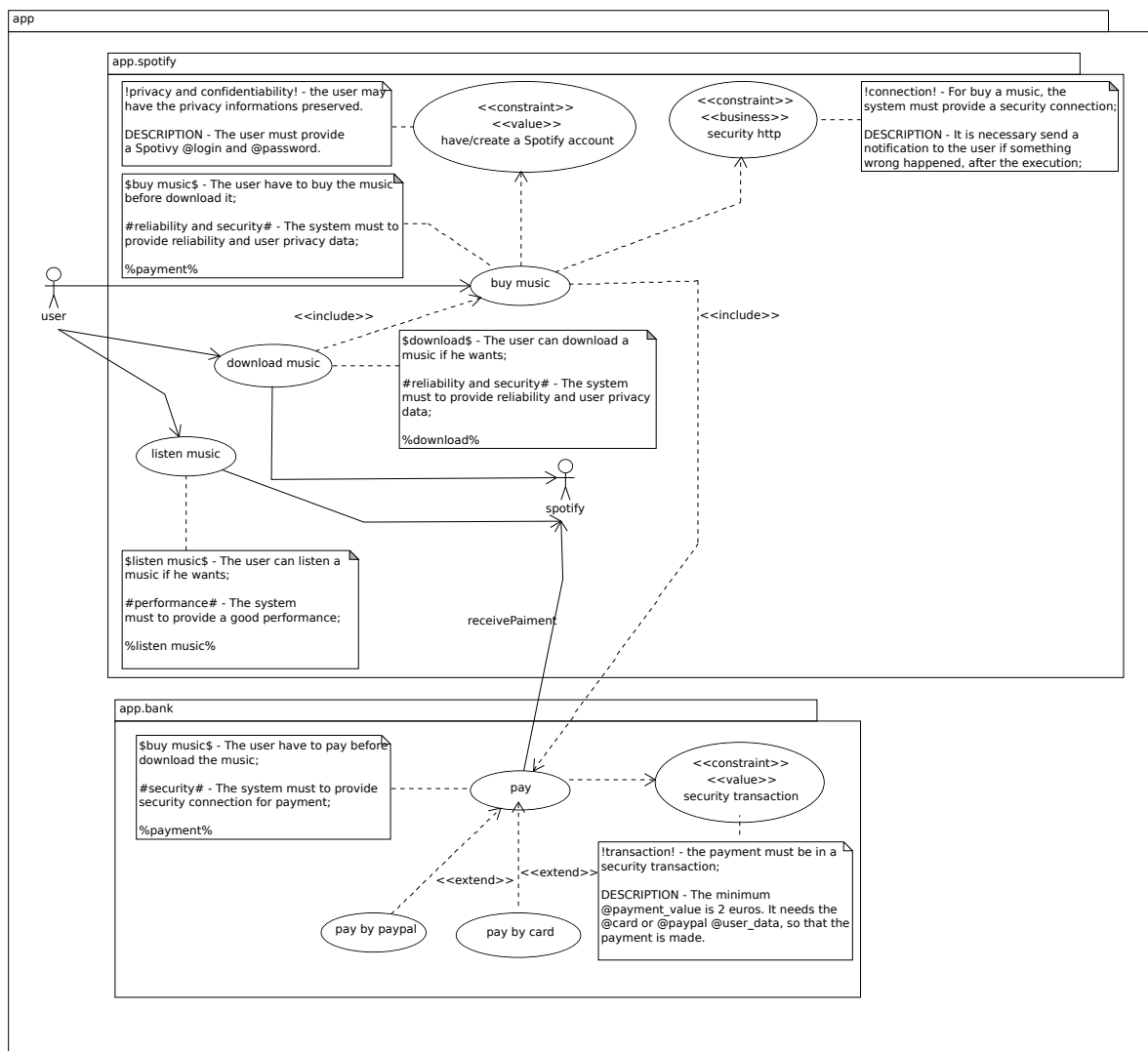


Figure 55: To Publish Music π-UseCase.

restriction details the type of constraint (*VALUE*), a *description*, *name* and a *non-functional attribute* (*authentication*). These properties are described in all constraints (item 2, figure 57).

Figure 58 shows the service components that must interact with the *application* and the interaction among them. There are four services that are invoked to produce results for the *Publish Music* application: *Spotify*, *Facebook*, *Twitter* and *Bank*.

5.1.2 π-ServiceProcess Model

Figure 59 shows the π-ServiceProcess diagram for the execution of activities based on contracts. We present here the part of the diagram produced from the concepts of the figure 57. The complete π-ServiceProcess diagram of the application is shown in figure 25. Notice that it is necessary to represent each CONSTRAINT modelled in a CONTRACT based service activity

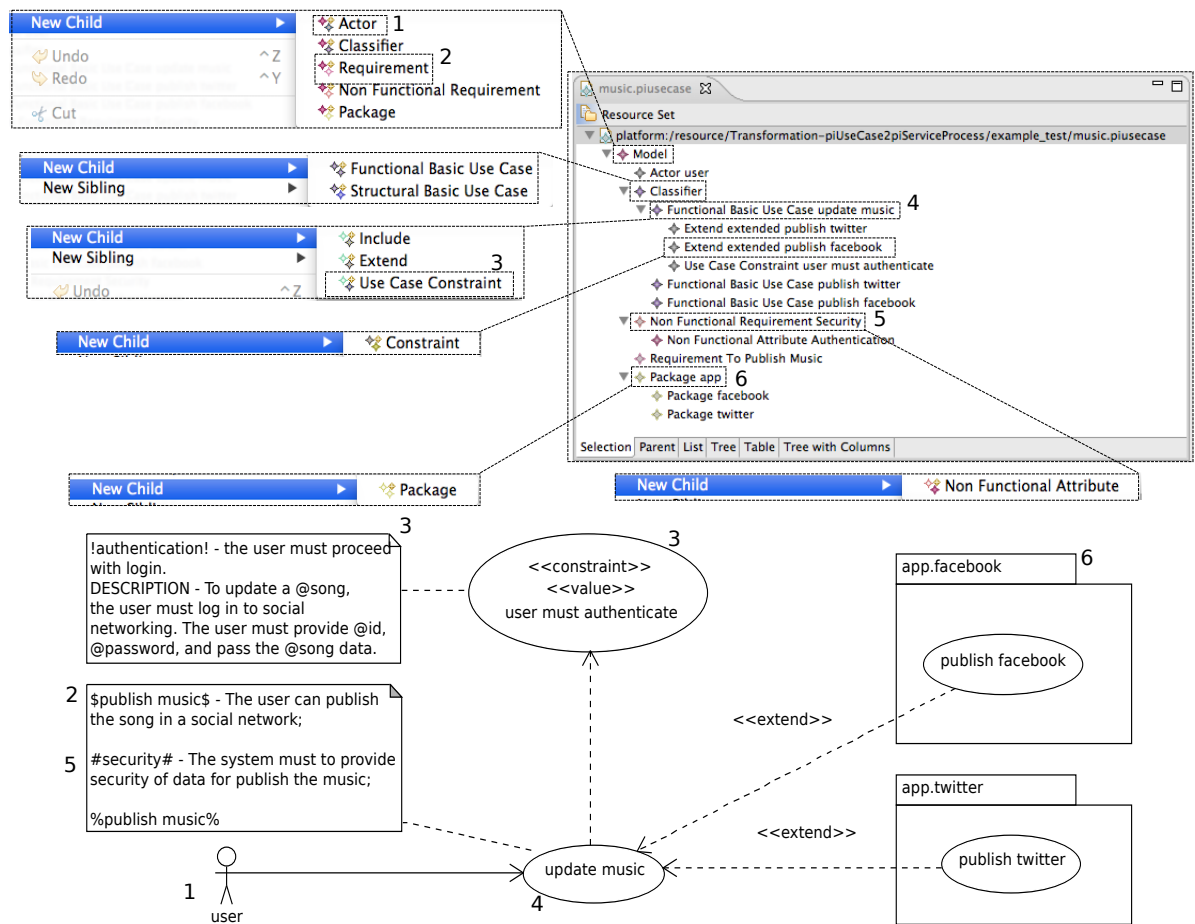


Figure 56: To Publish Music π -UseCase Environment Detail.

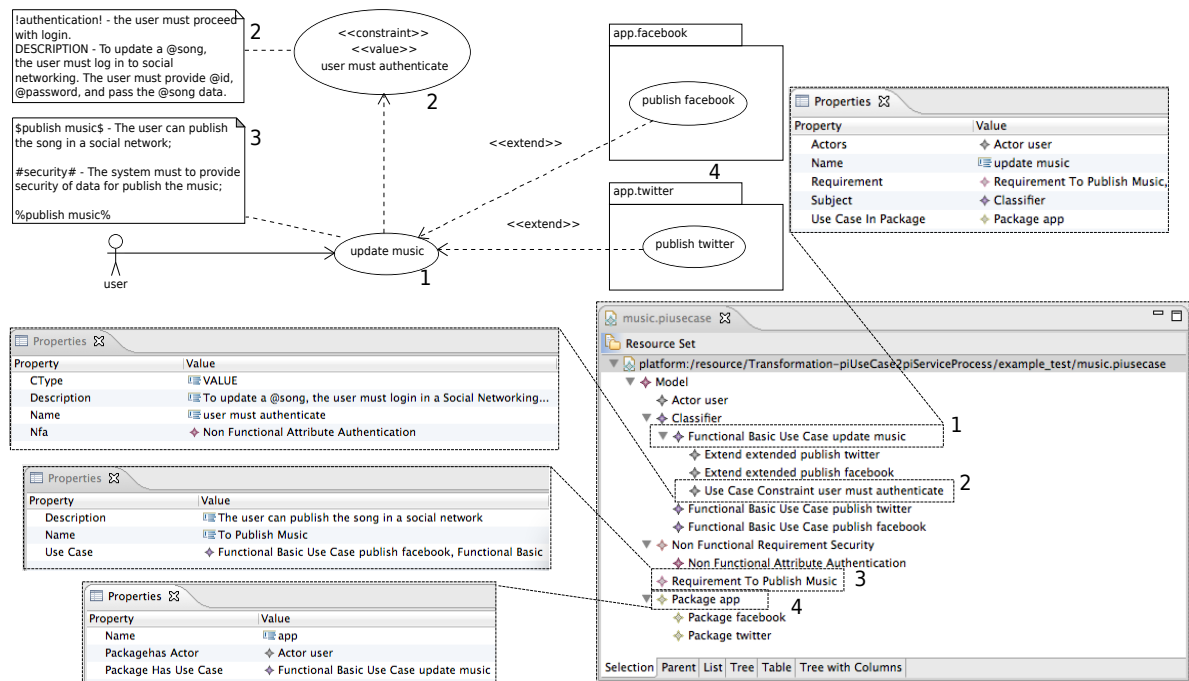


Figure 57: To Publish Music π -UseCase Environment Detail.

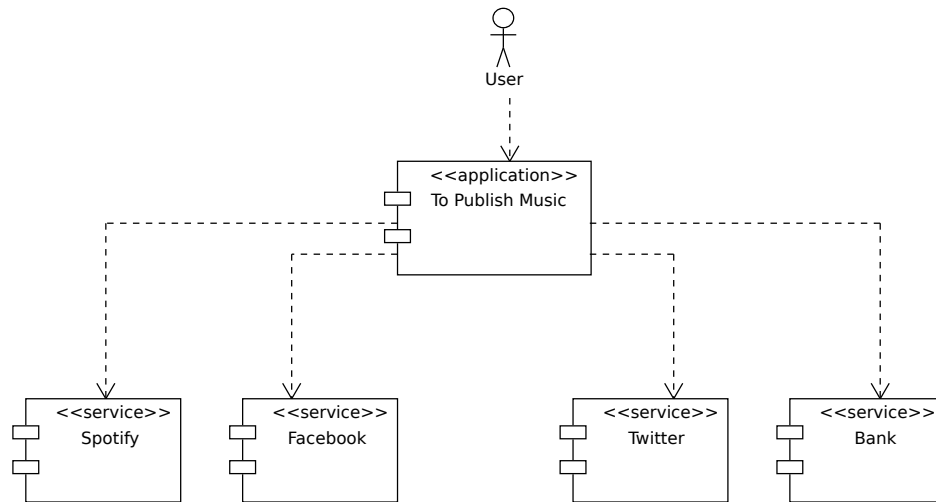


Figure 58: To Publish Music Services.

diagram. Each constraint described in the previous model is transformed into a contract in the π -ServiceProcess model.

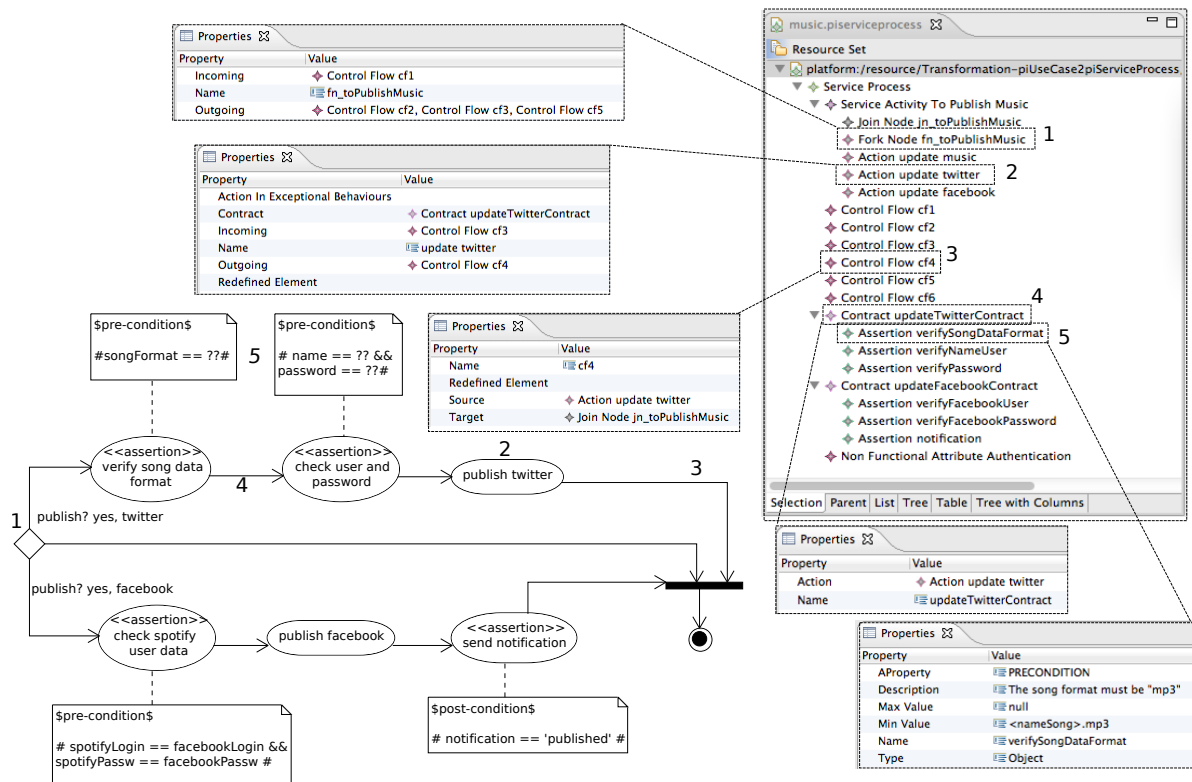


Figure 59: To Publish Music π -ServiceProcess.

Publish Twitter and *publish Facebook* (figure 59) have pre- and post-conditions assertions that are grouped into a contract for each service: (i) verify if the user data are correct; (ii) if the user is already logged in Spotify; As post-condition, it ensures the complete transaction and

verifies whether the payment authorization notification was sent to the user and to Spotify. There may be different restrictions depending on the services participating in the publish process. The user must be logged in Spotify in order to access Facebook, and Facebook must send a “successful login” acknowledge which is verified by the post-condition. For publishing a message, Twitter imposes pre-conditions on the length of the message and user data login. In contrast, Facebook only sends a confirmation acknowledge. Items 4 and 5 in figure 59 show the pre-conditions for “*publish Twitter*” concerning (i) the format of the music file and (ii) access authorization credentials, namely login and password.

The development of the π -ServiceProcess refine the constraints into contracts for the services as shown in Figure 59.

5.1.3 π -ServiceComposition Model

There are four external BUSINESS COLLABORATORS in this application: *Bank*, *Spotify*, *Twitter* and *Facebook* (items 3, figure 60). Figure 60 shows the application services that consist of a set of functions: *search music*, *select music*, *buy music*, *download music*, *listen music* and *publish music*. The *publish music* action calls two service collaborators (Facebook and Twitter), and the *buy music* action calls two functions of the Bank service collaborator.

The *To Publish Music* π -ServiceComposition model defines three policies: *transactionPolicy* (item 4, figure 60), *httpAuthPolicy* (item 2) and *authPolicy* (item 6). The *transactionPolicy* verifies the Bank data restrictions over the payment and confirmation process. The *httpAuthPolicy* policy specifies rules over the Facebook function for the authentication and notification processes. These rules are executed during the publish message function request. The *authPolicy* policy implements the open authentication protocol required for accessing the Twitter service.

From the *To Publish Music* rules and policies it is possible to model and associate non-functional properties to services’ compositions, e.g. the integration of information retrieved from different social network services, automatic generation of an integrated view of the operations executed in different social networks or for providing security in the communication channel when the payment service is called.

From this stage, the π -PEWS model (PSM) depicted in figures 31 and 32 can be generated.

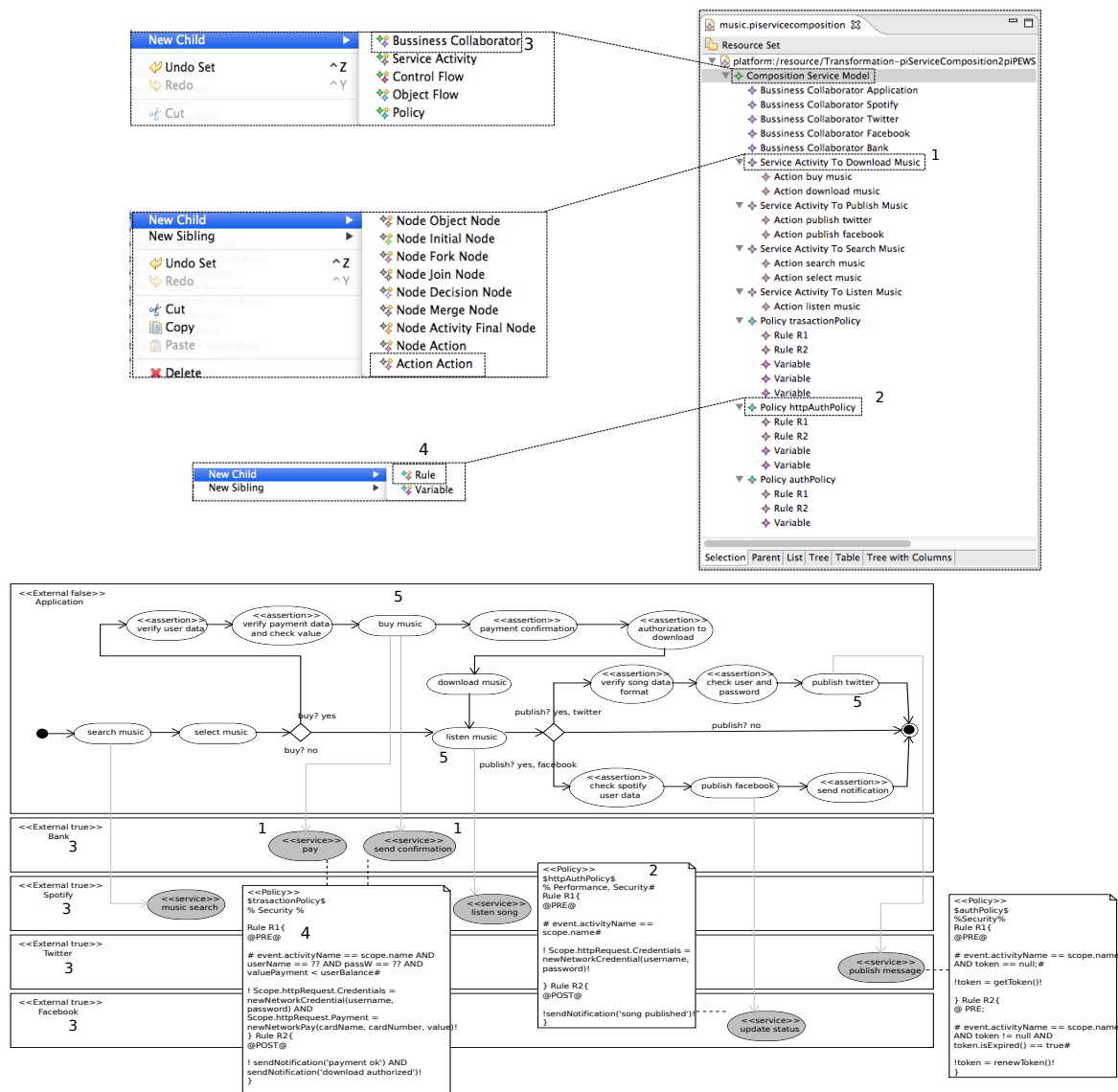


Figure 60: To Publish Music π-ServiceComposition Model.

5.2 Example 2: Crime Map

The *Crime Map* example addresses the design and implementation of a system that deals with statistical data on criminal activities. The system works in conjunction with third-party services, in order to process a search supplied by a user. The system must provide a public utility service to inform citizens about the real crime situation in a particular location or region. Making use of third party services, the application must provide a statistics crime portal. The result depends on user queries and upon a user demand results can be presented on a map. The system also interacts with micro-blogging, for looking up messages containing specific crime information. The presentation of the data concerning criminal activities is provided by the *Police service*. The presentation of a locality map crime is made by the *Google Maps service*

and messages presentation on the micro-blogging is done by the *Twitter service*. The system also has a *Post service*, for searching an address location that can be then presented in a map by Google maps. The scenarios to query data and crime statistics have the following actors: (i) *User*; (ii) *Map service*, (iii) *Post service*, (iv) *Micro-blogging service* and (v) *Police service*, to query the criminal activities (figure 61).

Considering the application functions, the crimes may be searched by types, e.g. *theft*, *murder*, *kidnapping*, etc. The search can also be made by city' district or region, for example, *center*, *west*, *south*, *north* or *east*, and it can process search temporal criteria, for example, crimes of the previous day of the last weeks or months. Finally, the service can also process conjunctive queries, e.g., *the number of kidnapping of a specific district in the last month*.

Whenever a new crime is published by the police, the system should post a message on Twitter stating the new crime, with a specific hashtag, e.g. *#crime043_kidnapping_center_05-08-2012*. The hashtag structure contains the crime number, type, local and the day it happens. From this, users can comment about the crime and these comment will appear in the application interface. All users must have an account to use the application.

The application functionalities are: *Choose search type of crime*; *see crime information*; *Receive information on the chosen query*; *See map*; *List crimes*; and *Comment about a crime*.

Non-functional requirements for this application include: *queries response performance*, *crime information validation*, *crime information presented in a map* and *ensuring crimes data security*.

5.2.1 π -UseCase Model

Figure 61 presents the Crime Map π -UseCase model. This model includes examples of *business*, *value* and *exceptional behaviours* constraints. The application accesses data from third party services to provide the user an easy way to visualize new information, these services are: *Map*, *Police*, *Postal* and *Microb-logging Services*.

The user interacts with the application that will perform the services for obtain and publish details about the crimes. The services are modeled as actors (they are external applications). Each service has a set of functions that can be performed. The π -UseCase model describes both the external services and the functionality of the application (see figure 61).

The user can search for crimes and view its occurrences in a map. The search may be performed by *type of crime*, *region* or *period*. The user can also perform a search that combines

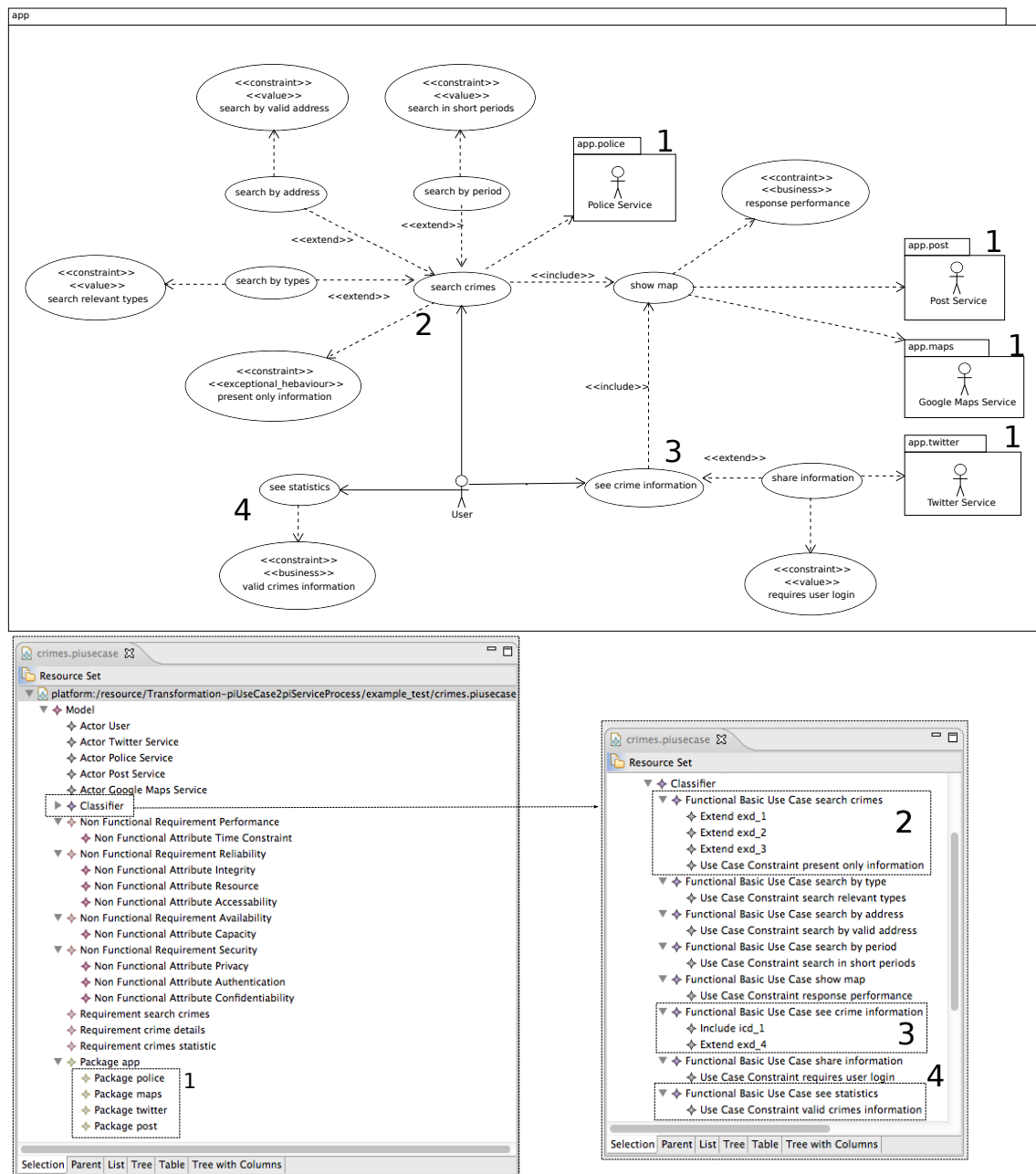


Figure 61: Crime Map π -UseCase.

these criteria. If the application’s user does not choose any option, the search will retrieve up to 30 recent occurrences. If the location maps service is unavailable or there is no response from it after 5 seconds, only the crimes information will be presented, without a map view.

Figure 61 also presents the π -UseCase model tree representation generated by the π SOD-M environment model .Figures 62 and 63 detail the use cases and constraints presented in Figure 61.

The *search crimes* use case (Figure 62) requests the *Police* service to verify the crimes according to the parameters supplied by the user. The restriction of this use case depends on

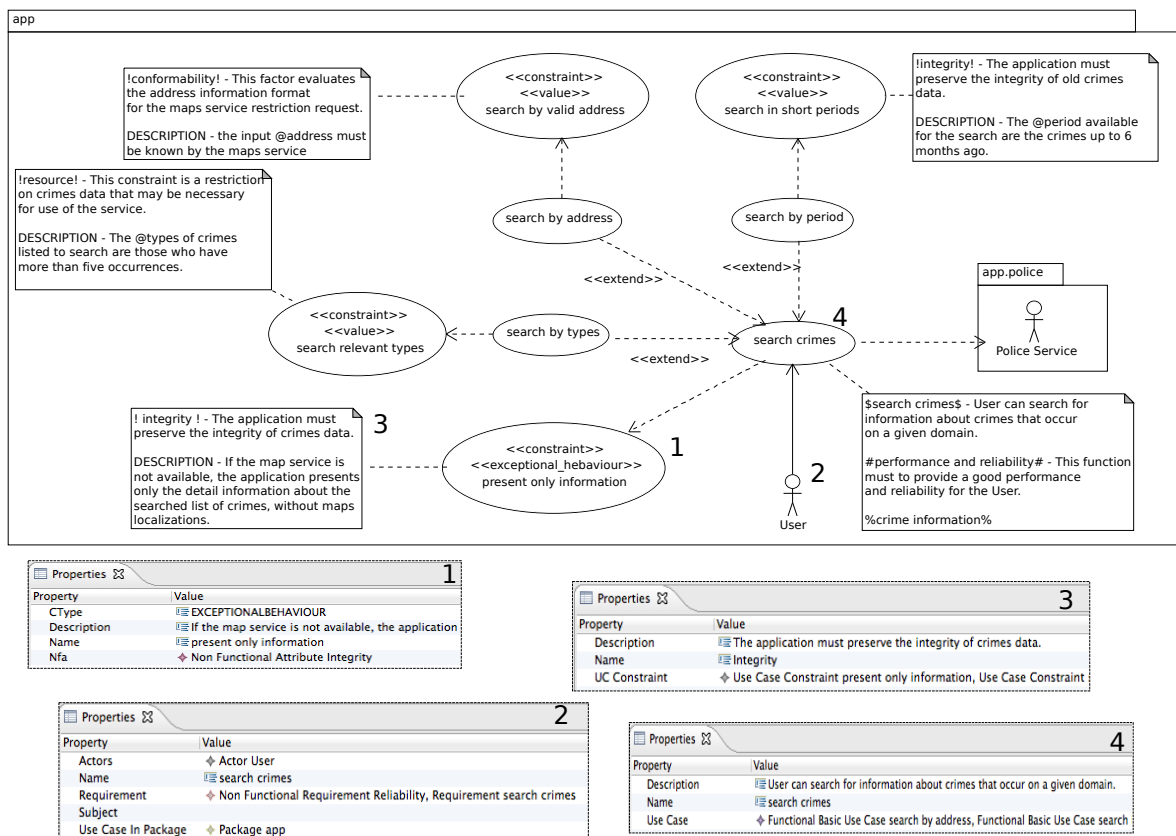


Figure 62: Search crime π -UseCase Detail.

the parameters selected by the user. If the user chooses one or several types of specific crime's type, the system will present those crimes that have more than five occurrences. For *search by address*, the address must comply with the restrictions of the police service, the research can be done by street, neighborhood, city or district. *Search by period* will provide only occurrences within the last six months.

Figure 63 describes the design of the use cases: *see crime information*, *see statistics*, *show map* and *share information*. To share information of a crime, it is necessary to access the Twitter micro-blogging service to proceed with the authentication and posting. The service requires the user id and password, however the application must ensure the User privacy information while performing authentication. To see statistics of crimes, the application should process a larger volume of data to generate statistical results. Thus, it is required to provide resources to ensure the data processing performance. The restriction on map presentation is over response time and accessibility of the *Postal* and *Google maps* services. The response time limit is 5 seconds. If the map is not processed, only the information of the crime are presented. Figures 62 and 63 also show the properties used in the π SOD-M environment and the equivalent concepts in the

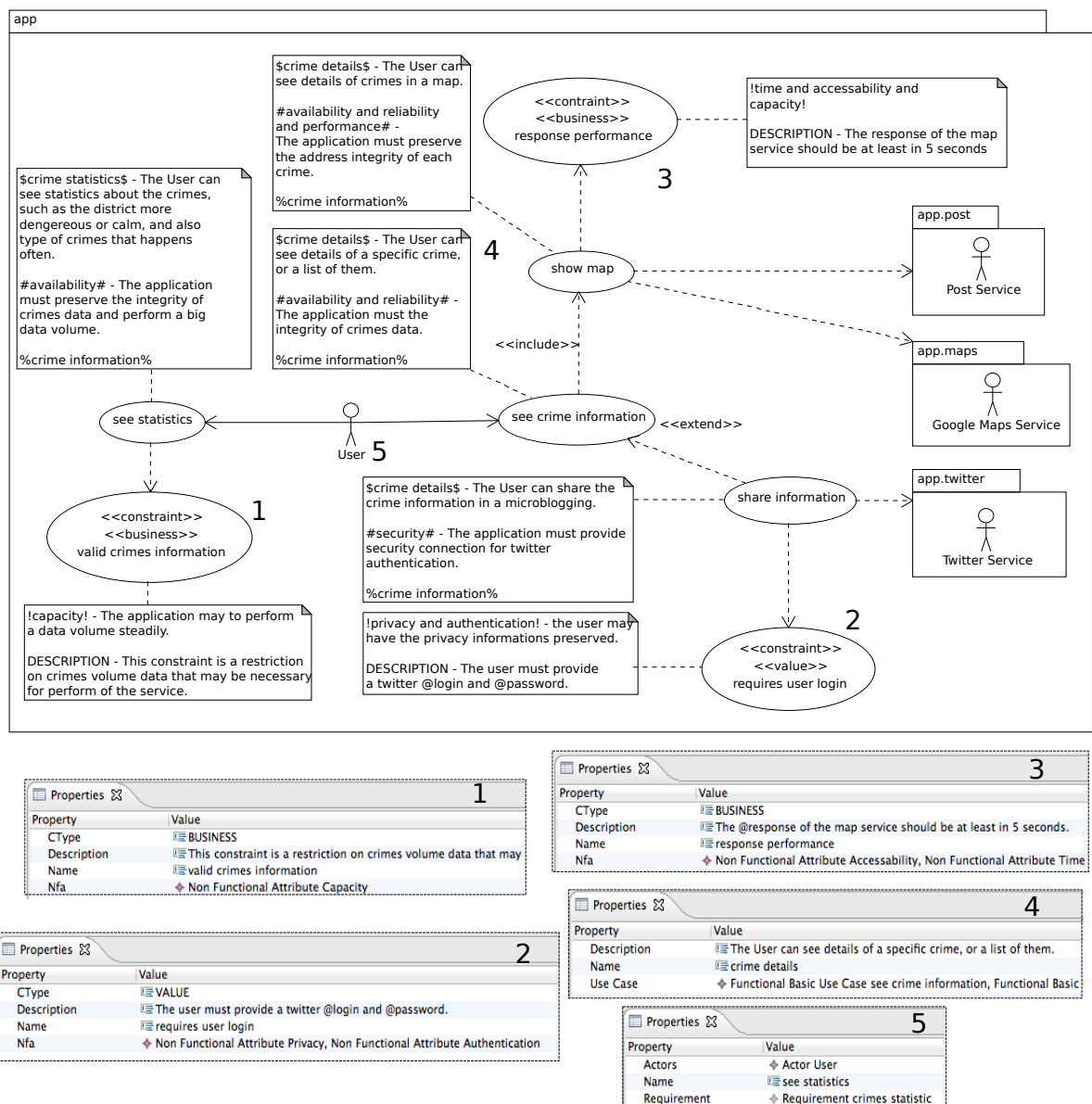


Figure 63: See crime information π -UseCase Detail.

model.

After the identification the various functions that are required by the system to perform the business services, the π -ServiceProcess model is used to represent the workflow necessary to perform a service within the system.

5.2.2 π -ServiceProcess Model

The π -ServiceProcess model is used to represent the workflow necessary to perform the system service. The service processes execution is described in two steps. Figures 65 and 67 show the flow of functions execution described in the π -UseCase models shown in Figures 62

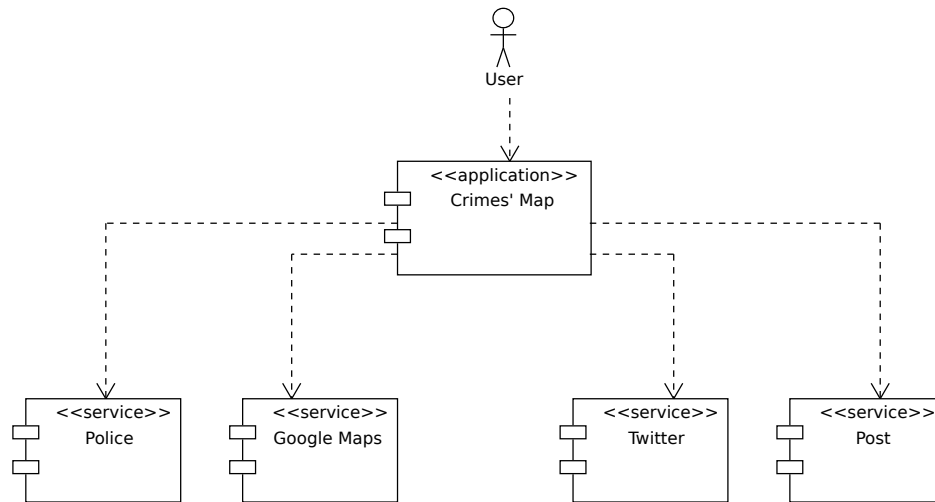
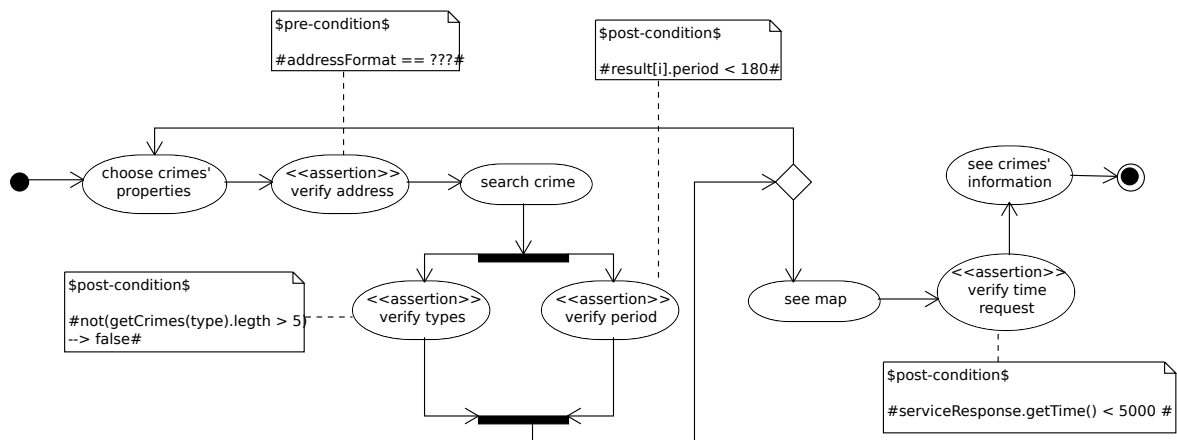


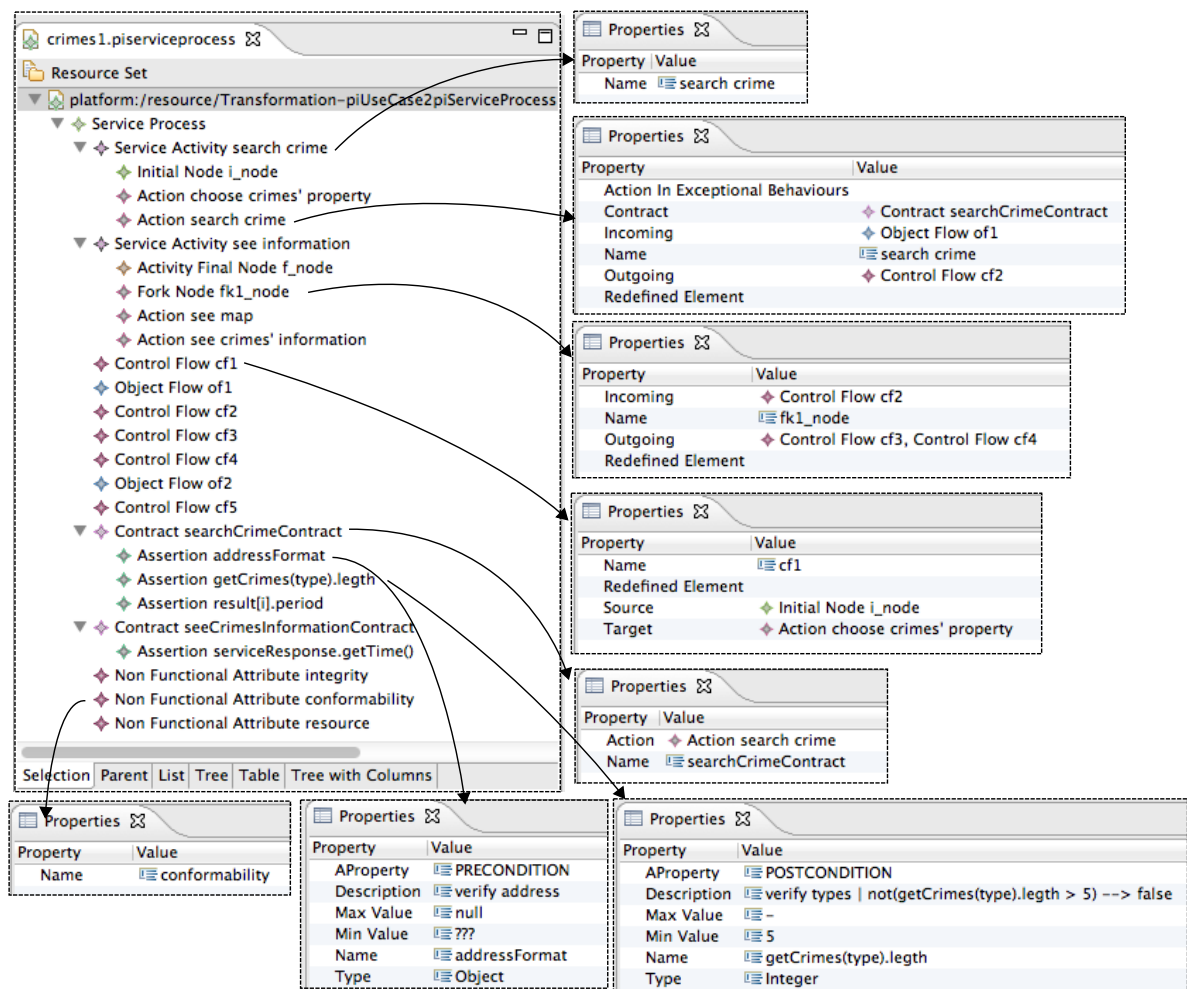
Figure 64: Crime Map Services.

and 63. Each diagram can represent different business services (see figures 62 and 63).

The diagrams of figures 65 and 67 were obtained by applying the π -UseCase2 π -ServiceProcess transformation to the diagrams of figures 62 and 63, respectively.

Figure 65: Crime information π -ServiceProcess.

Each constraint of figures 65 and 67 are transformed into assertions. For example, the assertions of address format, crime types and time are restrictions over the action *search crime*, being one pre-condition and two post-conditions. These restrictions form the contract on the *search crime* function. Similarly, for viewing maps, if the answer to the request takes longer than 5 seconds (5000 milliseconds), the map display is suspended and only the information is presented. Each assertion in this π -ServiceProcess model stems from the π -UseCase model constraints. These restrictions are shown in Figure 65. Figure 66 presents the equivalent model generated by the methodology transformation in the π SOD-M environment for the process de-

Figure 66: Crime information π -ServiceProcess Environment.

tailed in Figure 65. Each node defines the process element and its properties.

In Figure 66, *i_node* corresponds to the initial process node. ACTIONS are grouped into SERVICE ACTIVITIES, such as *choose crimes property* action that is part of the *search crime* service activity. A CONTROL FLOW is an edge that links two nodes, for example *cf1* connect the initial node *i_node* (source) with the *choose crimes property* action (target). Figure 66 also presents the assertions described in Figure 65 grouped into CONTRACTS. The contracts are: *searchCrimeContract* and *seeCrimeInformationContract*. These contracts are related with the actions *search crime* and *see crimes information*.

Figure 67 presents the assertions over the *see statistics* and *share information* actions. Both have 2 pre-conditions. To *share information* on Twitter it is necessary to verify the format of crime information (140 characters) and the Twitter id and password, for authentication. Regarding the *see statistics* action, there is a business restriction over data volume, and a value restriction over the crime information format. Thus, the presentation of the crimes statistics

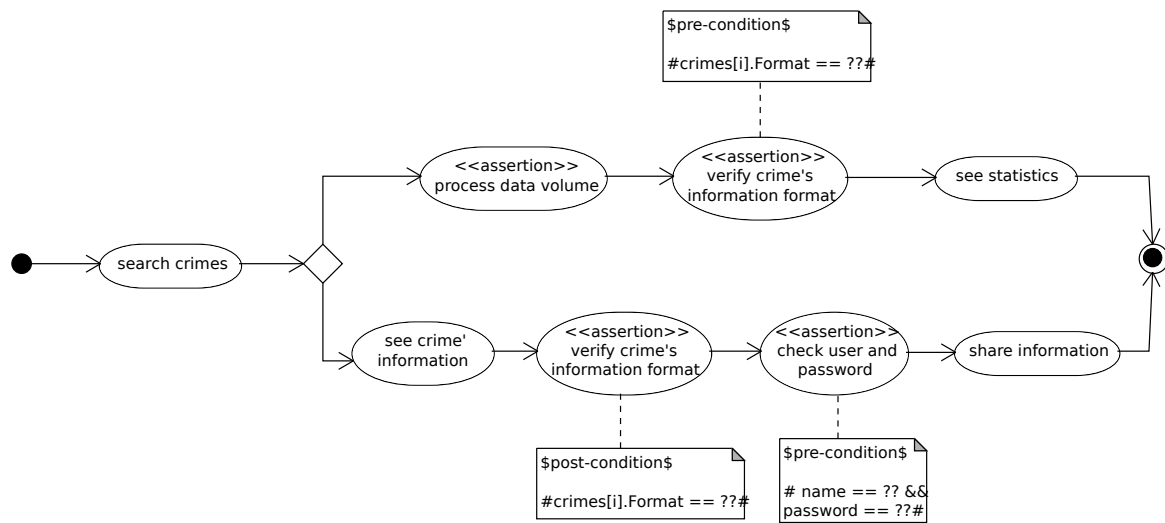


Figure 67: See crime statistic and share information π -ServiceProcess Detail.

must be done after the contract verification. Figure 68 presents the equivalent model generated by the methodology transformation in the π SOD-M environment for the process detailed in Figure 67. Each node defines the process element and its properties. Figure 68 also presents the assertions grouped into three contracts that are related with its specific actions. The contracts are: *seeStatisticsContract*, *seeCrimeInformationContract* and *shareInformationContract*. The contracts are related with the action *see statistics*, *see crimes information* and *share information*, respectively.

These models provide an overview of the business processes from the requirements described in the π -UseCase model. These models also provide a more detailed view of the execution of business processes and the possible interaction with external services. The description of this interaction is the result of the refinement of these models (π -ServiceProcess models) through the π -ServiceComposition model.

5.2.3 π -ServiceComposition Model

Figures 69 and 71 show the π -ServiceComposition models, which represent the service composition processes of each business service and additional indication of which are members of the business that execute the action. Actions are derived from service activities, identifying the set of actions that are necessary for the completion of each service.

The main partition expresses the *Application* execution (*External false*) that represents the general process. Actions have the equivalent service relation, which realize the external service, such as *Police*, *Google Maps* or *Twitter* services. They are expressed by a BUSINESS

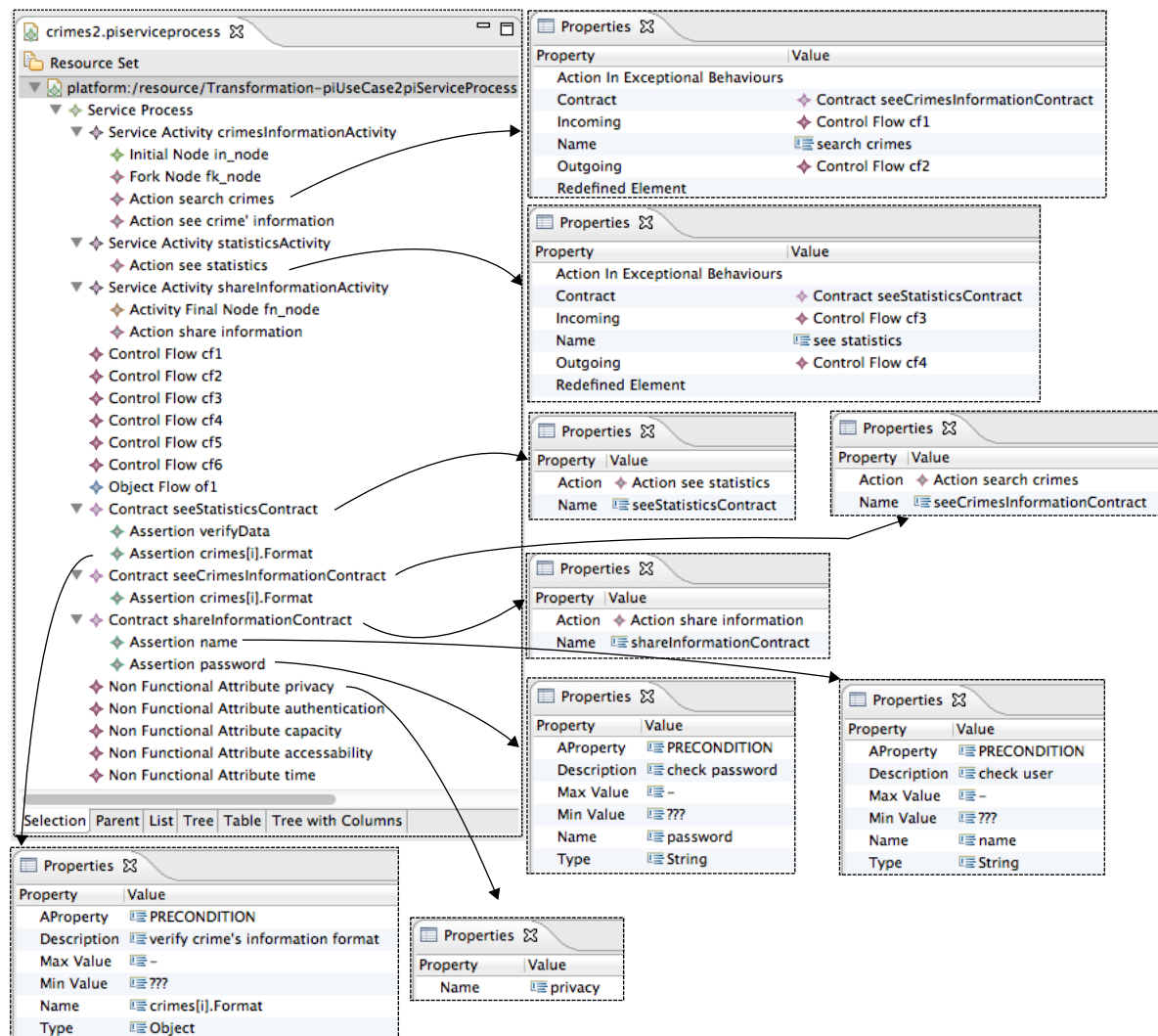
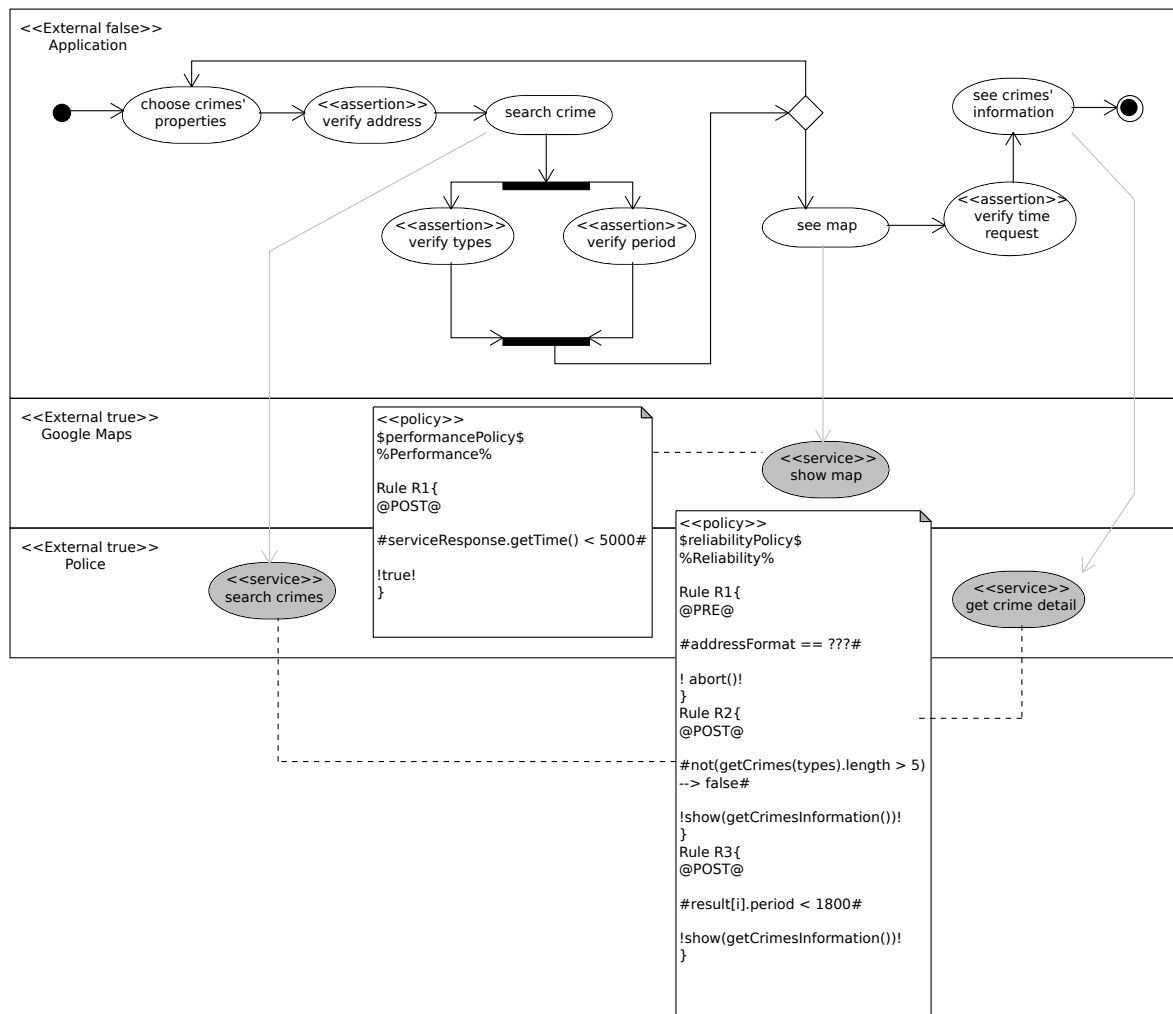


Figure 68: Crime statistic and share information π -ServiceComposition Environment.

COLLABORATOR. Both, Figure 69 and 71 present two external BUSINESS COLLABORATORS describing the application services that are invoked. These π -ServiceComposition models refine the π -ServiceProcess models, matching action with real service functions that may be executed by the system application.

For each π -ServiceComposition model, the contracts described in the π -ServiceProcess model are grouped in policies. Figure 69 describes the *performancePolicy* and *reliabilityPolicy* policies, both associated with a service activity and its actions. It is important to highlight that all policies are applied to the entire actions associated with a service activity, for example, the *performancePolicy* verifies the request time for presenting the Google maps. If the time exceeds 5000 milliseconds, the maps call is ignored and only the crime information is presented. The *reliabilityPolicy* is associated with crime data verification. There are three rules over crime address format, types of crime and the period of time in which it happens. The crime presentation

Figure 69: Crime information π -ServiceComposition.

may obey the pre- and post-condition described in this policy. Figure 70 presents the π SOD-M tool description for the model described in Figure 69.

Figure 71 refines the π -ServiceProcess model described in Figure 67 and Figure 72 presents the π SOD-M tool description for the model described in Figure 71. This model details the policies for *crime statistic* and *share information* services. The policies are *statisticPolicy* and *authenticationPolicy*, respectively.

These models provide an overview of the external services and its composition from the actions described in the π -ServiceProcess model. These models also provide a more detailed view of the system restrictions, applying policies over external services. This model refines the π -ServiceProcess models detailing the Business Collaborators that are expressed as external services, and grouping contract into policies.

The result of the transformation of this model to the π -PEWS model is given in the listing

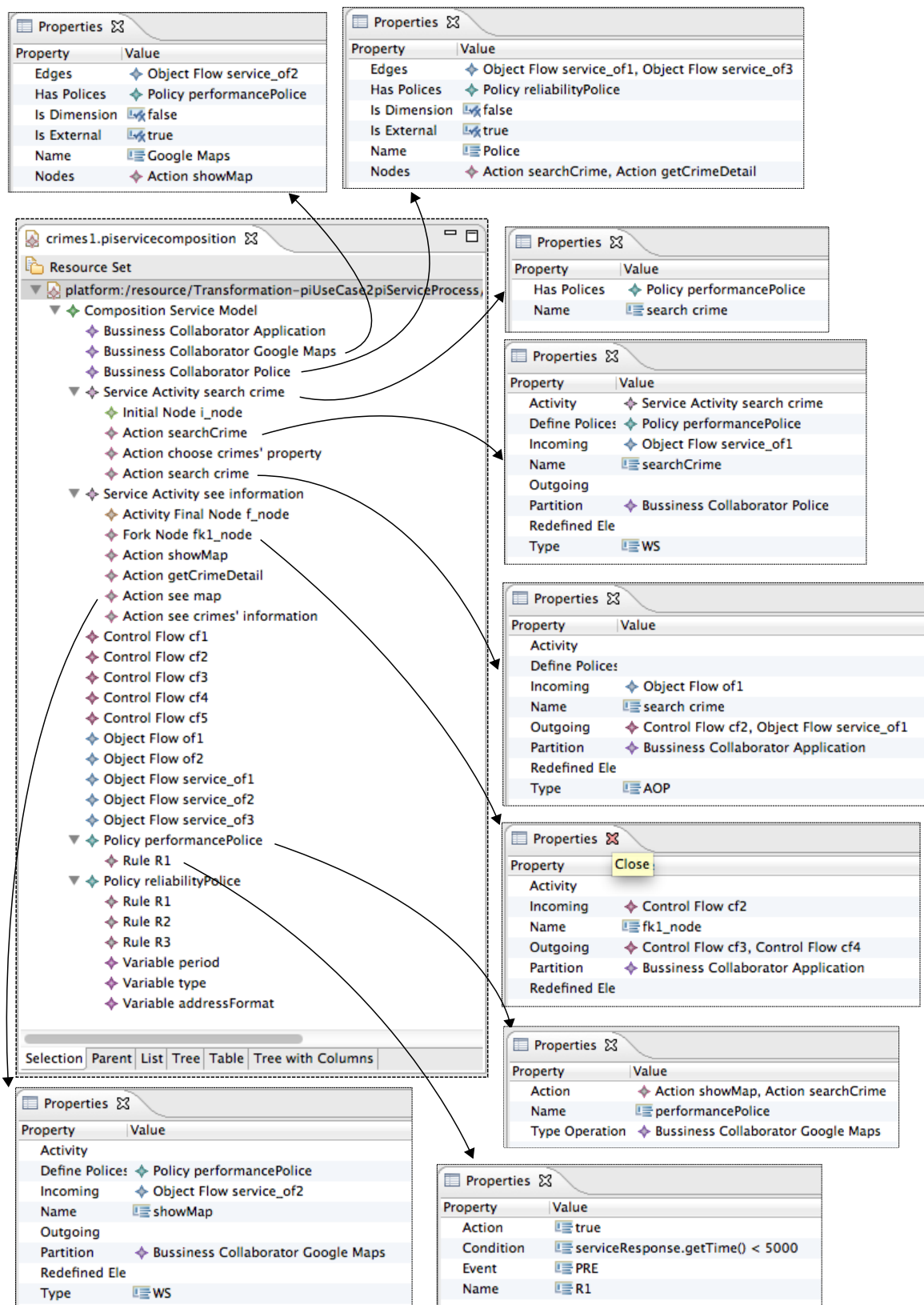
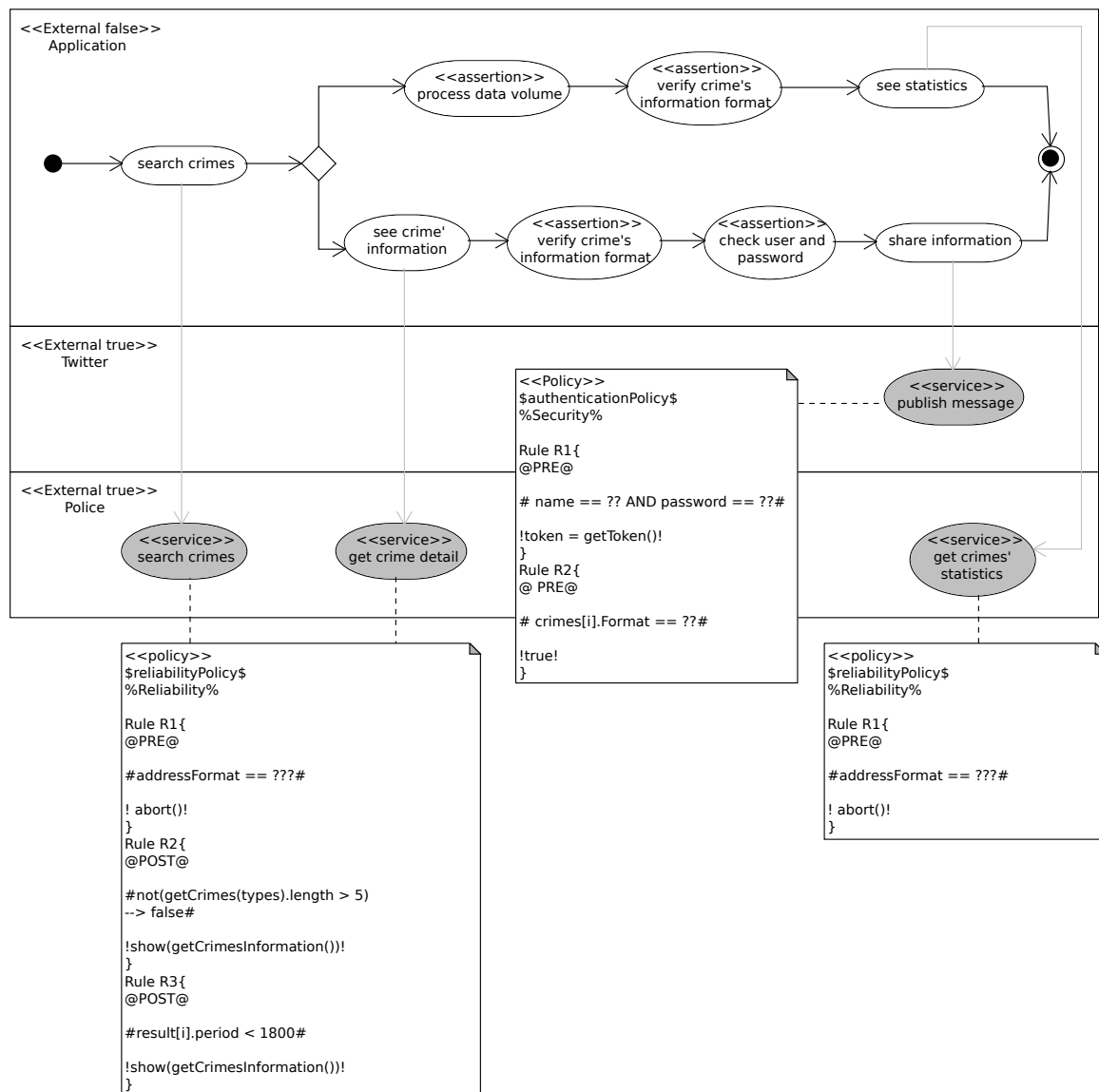


Figure 70: Crime information π -ServiceComposition Environment.

5.1 and 5.2.

Figure 71: Crime statistic and share information π -ServiceComposition.

5.3 Example 3: GesIMED Application

Aiming to perform a qualitative analysis and the validation of π SOD-M methodology, we present the *GesIMED Application*² case study.

This example was originally developed in [39] for the SOD-M methodology. Here, we adapt the requirements of the system to include NFR. This system is a management Web system that processes medical images through the Web [39]. The objective is to manage information about the creation and maintenance of scientific studies for neuroscience research. The application is designed to be used primarily by researchers in neuroscience, such as neurologists,

²If necessary, a detailed description of this case study can be found in [39].

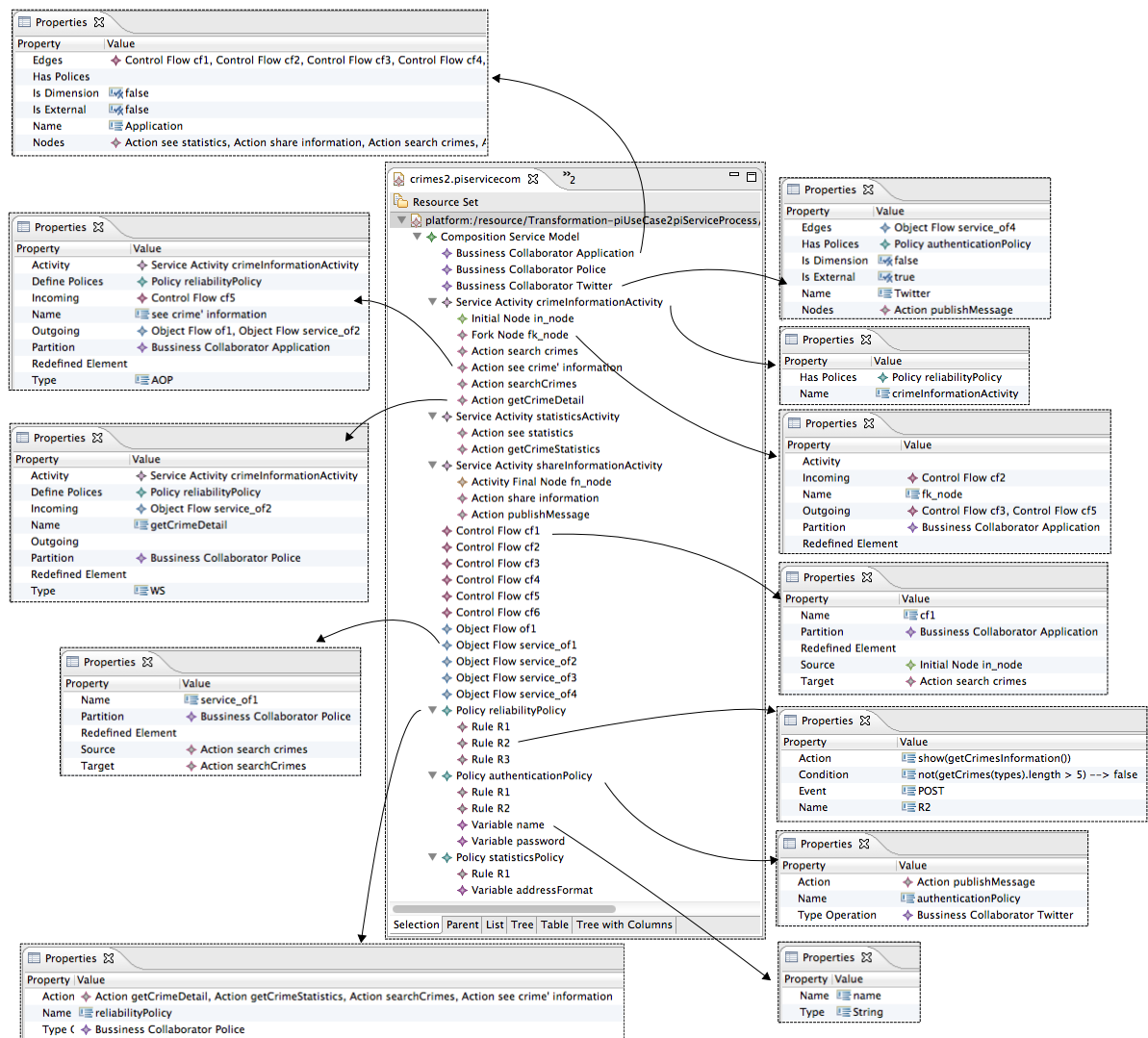


Figure 72: See crime statistic and share information π -ServiceComposition Environment Detail.

neuropsychologists and neuroradiologists who conducting research in this area.

The application requirements are the following: (i) a database of medical images that can be accessed by the user; (ii) it has an interface for querying the database; (iii) implement standard procedures for analyzing and processing stored images; and (iv) the images analysis and processing results must be also stored, so they can be used in future studies.

The Medical Image Analysis Laboratory from the Universidad Rey Juan Carlos (LAIM) offers three specific services to researchers in neuroscience. The access to services has a cost and financial revenues are assigned to LAIM. The services are:

- Storage and retrieval medical imaging service (SACim);
- Image processing service (SPim);

Listing 5.1: pi-PEWS Specification: Crime information.

```

1 ns police = "http://www.pm.rn.gov.br/service"
2 ns googleMaps = "http://maps.googleapis.com/maps/api/"

4 alias chooseCrimesProperties = portType/chooseCrimesProperties in police
5 alias searchCrimes = portType/searchCrimes in police
6 alias seeMap = portType/showMap in googleMaps
7 alias seeCrimesInformation = portType/getCrimeDetail in police

9 (chooseCrimesProperties . searchCrimes)* . seeMap . seeCrimesInformation

11 def contract searchCrimesContract{
12   isAppliedTo: searchCrimes;
13   requires: addressFormat == ??
14   (onFailureDo: call(chooseCrimesProperties) );
15   ensures : !(getCrimes(types).length > 5) ==> false &&
16             result[i].period < 1800
17   (onFailureDo: show(getCrimesInformation()));
18 }

20 def contract showMapContract{
21   isAppliedTo: seeMap;
22   ensures : serviceResponse.getTime() < 5000
23   (onFailureDo: skip);
24 }

```

Listing 5.2: pi-PEWS Specification: Crime statistic and share information.

```

1 ns police = "http://www.pm.rn.gov.br/service"
2 ns twitter = "https://dev.twitter.com/docs/api/"

5 alias searchCrimes = portType/searchCrimes in police
7 alias seeCrimesInformation = portType/getCrimeDetail in police
4 alias seeStatistics = portType/getCrimeStatistics in police
6 alias shareInformation = portType/publishMessage in twitter

9 searchCrimes . ((seeCrimesInformation . shareInformation) |
10                seeStatistics)

12 def contract searchCrimesContract{
13   isAppliedTo: searchCrimes;
14   requires: addressFormat == ??
15   (onFailureDo: call(chooseCrimesProperties) );
16   ensures : !(getCrimes(types).length > 5) ==> false &&
17             result[i].period < 1800
18   (onFailureDo: show(getCrimesInformation()));
19 }

21 def contract shareInformationContract{
22   isAppliedTo: shareInformation;
23   requires: name == ?? AND password == ?? &&
24             crimes[i].Format == ??
25   (onFailureDo: call(shareInformation) );
26 }

28 def contract seeStatisticsContract{
29   isAppliedTo: seeStatistics;
30   requires: crimes[i].Format == ??
31   (onFailureDo: call(seeStatistics) );
32 }

```

- Image visualization service (SVim);

From the functions offered by these services we will detail the application characteristics. The application uses these services offered by LAIM for modeling the application.

Given the description of the business requirements and the expected functionality of the system, π SOD-M is used for developing this application.

We will detail the *perform image processing* business service. In π SOD-M, the π -UseCase model describes the application functions, restrictions and its non-functional attributes. This

business service (*perform image processing*) is sufficiently rich to describe the peculiarities of π SOD-M compared to the original model of SOD-M.

5.3.1 π -UseCase Model

Figure 73 shows the π -UseCase model. This model defines four constraints: *authenticate*, *response time*, *data format* and *validate payment card*. These constraints are restrictions over the functions of the *GesIMED* application. In the description of each constraint it is necessary to describe the constraint type (*business* or *value*). The model has one business constraint and three value constraints. It is possible to describe constraints for each use case application.

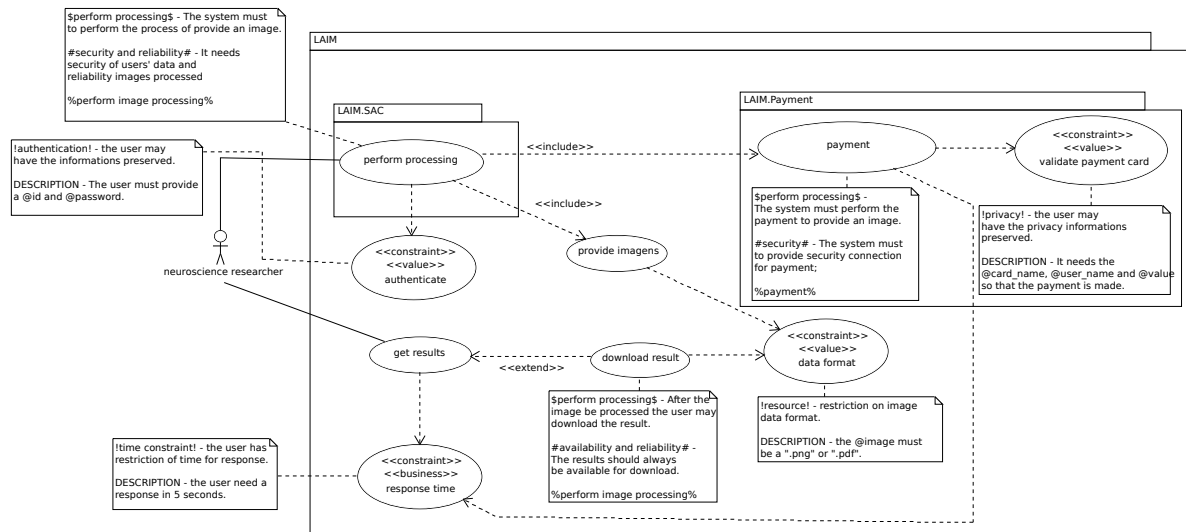


Figure 73: π -UseCase: Perform Image Processing.

Besides the constraints, the model details the non-functional attributes related to each constraint, for example, the *response time* constraint. For value constraints, it is necessary identify candidates variables to be verified, through the use of @. For example, the authentication constraint, has the @id and @password identifiers that must be verified.

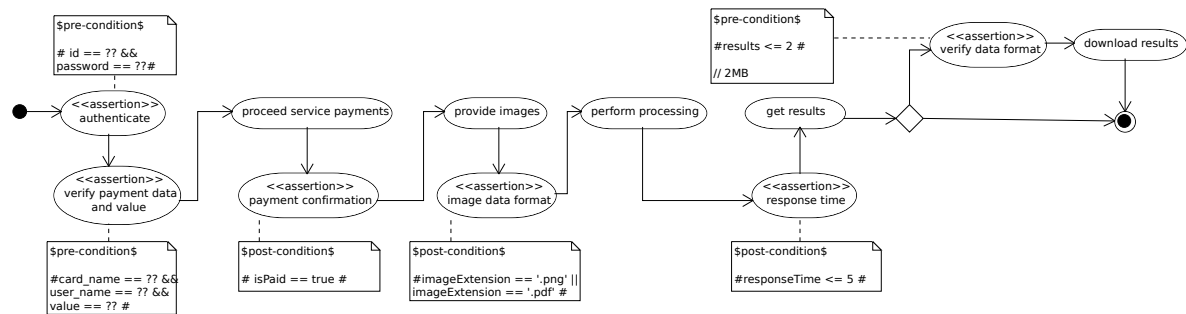


Figure 74: π -ServiceProcess: Perform Image Processing.

The main feature of our approach is the association of non-functional requirements with use cases. For example, the user can identify that the quality requirement related to a payment process is *transactional* or *data privacy*. So, through the process, it is possible to refine this information, considering other details provided.

The result of the π -UseCase model is a detailed list of constraints, its respective use case, system requirement information and non-functional attributes required for each service.

5.3.2 π -ServiceProcess Model

Figure 74 shows the π -ServiceProcess model. This model transforms the use cases into actions, and constraints into assertions. The resulting workflow describes pre or post-conditions on workflow execution. The assertions are wrappers around the action. All the assertions related to a specific action become a contract. For example, the *proceed payments service* action has a pre- and a post-condition. The pre-condition is a restriction over the payment information (*card number, name* and *value*), while the post-condition defines the final state after the payment. The other assertions of this model are the result of the transformation of each constraint described in the π -UseCase model.

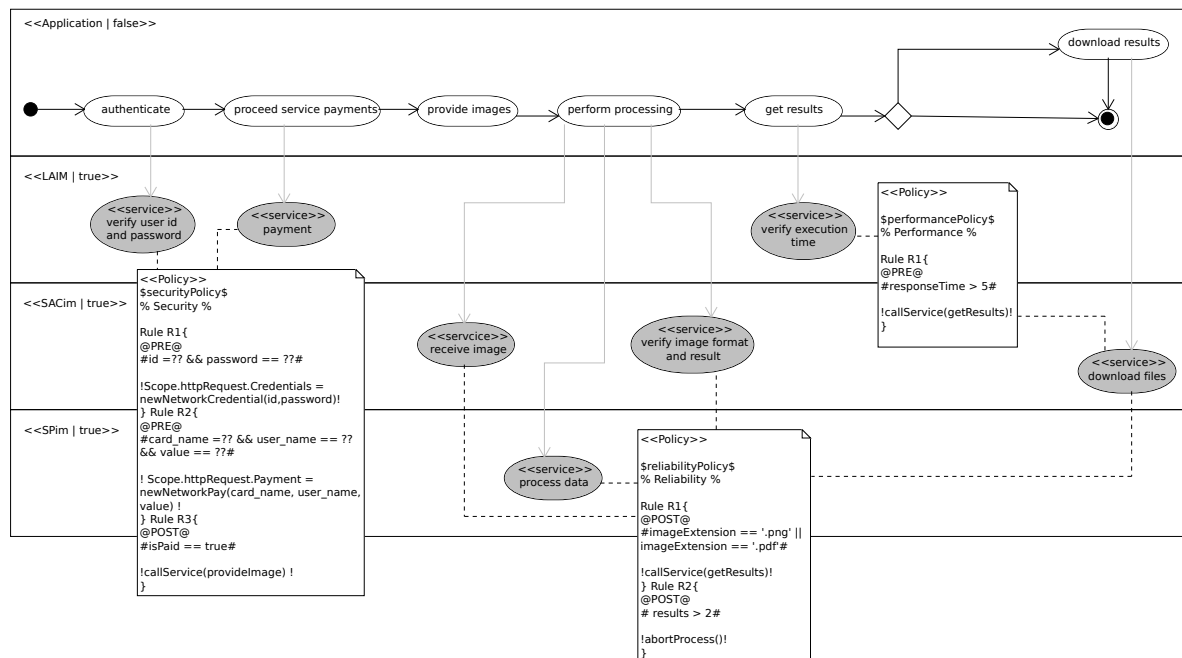


Figure 75: π -ServiceComposition: Perform Image Processing.

Listing 5.3: pi-PEWS Specification: Perform Image Processing.

```

1 ns app = "http://www.neuro.laim.org"
2 ns LAIM = "http://www.neuro.laim.org/laim-service.wsdl"
3 ns SACim = "http://www.neuro.laim.org/sacim-service.wsdl"
4 ns SPim = "http://www.neuro.laim.org/spim-service.wsdl"

6 alias authenticate = portType/verifyUserAndPassword in LAIM
7 alias proceedServicePayment = portType/payment in LAIM
8 alias provideImage = portType/provideImage in app
9 alias receiveImage = portType/receiveImage in SACim
10 alias processData = portType/processData in SPim
11 alias verifyImageFormatAndResult = portType/verifyImageFormatAndResult in
12 SACim alias getResult = portType/verifyExecutionTime in LAIM
13 alias downloadResults = portType/downloadFiles in SACim

15 service performProcessing =
16     receiveImage . processData . verifyImageFormatAndResult

18 authenticate . proceedServicePayment . provideImage .
19     performProcessing . [act(getResult).time > 5]getResult .
20     {downloadResults}

21 def contract authenticateContract{
22     isAppliedTo: authenticate;
23     requires: id == ?? && password = ??;
24     (onFailureDo: call(authenticate) )
25 }

27 def contract paymentContract{
28     isAppliedTo: proceedServicePayment;
29     requires: card_name == ?? &&
30             user_name == ?? &&
31             value == ??;
32     ensures: isPaid == true ;
33 }

35 def contract performProcessingContract{
36     isAppliedTo: performProcessing;
37     requires: imageExtension == '.png' || imageExtension == '.pdf'
38     (onFailureDo: skip );
39     ensures: resultValue <= 2 ;
40 }

```

5.3.3 π -ServiceComposition Model

The π -ServiceComposition model presented in figure 75 refines the π -ServiceProcess model. This model describes the real services and their associated collaborators. The information of the business collaborator comes from the package description. Contracts with the same non-functional requirements become service policies. The policies define rules, which are a direct transformation of the assertions in the previous model.

Listing 5.3 shows the π -PEWS specification that was generated from the π -ServiceComposition model. This specification considers the operations (lines 6-16), its business collaborators (namespaces in the lines 1-4), the main workflow (lines 18-20) and services restrictions (described as contracts in the lines 21-40). The final result of our approach is the specification that describes the structure and sequence of execution to be performed by an orchestrator. π -PEWS specification defines a set of contracts to represent the application restrictions. Thus, using π SOD-M it is possible to refine the models until the code generation.

5.4 Lessons Learned

This section summarizes the experience we obtained after the use of π SOD-M for the three examples. In this section we also present some results collected from the evaluation made between the two approaches.

5.4.1 Example 1: To Publish Music

In this case study we can see that the restrictions on web services can be modeled from the early stages of development, not leaving it to the programming language, or to the programmer to resolve this type of problem. More specifically, this case study had restrictions *performance* and *security*. Since the π -*UseCase* model to the π -*ServiceComposition* model, non-functional requirements and their attributes might be described without serious impact on the modeling of other application requirements.

The restrictions on the functions provided by the *Spotify* service was basically *performance*. This type of restriction is not easy to represent in the implementation, because it relies on the network available to the user. We described the constraints and refined them into a *performance* policy for updating the music. The security restrictions have been described for the service that provides banking information. The security policy generated was described for two functions of the Bank service, *pay* and *send confirmation* functions. This type of specification can be refined incrementally by applying the modelling steps defined by the π SOD-M methodology.

It does not cover all the possible restrictions in an application but it shows that it is possible to guarantee quality requirements refining information at each iteration of the modeling, such as constraints of *performance* and *security*.

The difficulty in modeling this example was the definition of policies for each type of non-functional requirement. For example, the *httpAuthPolicy* policy for the authentication on Facebook service has restrictions from both types, *performance* and *security*.

5.4.2 Example 2: Crime Map

The most important process when designing a system is the requirement analysis process, especially identifying the non-functional requirements and the system restrictions. In order to ease the coding process and let the developer focus on technical issues, the system architecture must be automatically generated, from the system design.

Using π SOD-M has enabled the explicit specification of functional and non-functional requirements and their refinement along the different phases of the development process.

In this example we can see the restrictions on four web services: *Police*, *Google Maps*, *Twitter* and *Post* service. As these services are independent, the constraints of the application being developed must comply with the service APIs. If a restriction is inconsistent with a function, there is no guarantee that it will be respected. For example, if the police offers crimes information on a particular region by type or day, it is not possible to present the restrictions of a particular time of day, for example, crimes that happen in the afternoon.

This example has two *reliability* policies and one *security* policy. Policies on data are common, because they are restrictions on the service interface. We notice in the development of this example that function restrictions, especially the execution time and time response are more difficult to ensure.

5.4.3 Example 3: GesIMED

The π SOD-M methodology helps to define a detailed description of the application, its restrictions and non-functional requirements associated with each service. The refinement of the quality requirements of each level helps to improve the development detail in order to produce a more reliable application. The NFRs are not only described in a general way, but the restrictions are designed to each service individually.

Considering the modeling of this example in SOD-M, the first step of the is to list the global requirements (business case). Figure 76 shows the use case model for the application. This model is very simple and describes the main application requirements. The actor is represented as the “neuroscience researcher”, which is the final consumer of the services to be implemented. Furthermore, the model is represented as use cases, which describes the main system business services: “perform image processing³”, “perform image view” and “perform query”.

The next model is the the extended use cases model. This model is used to model the functionality required by the system to perform each business services. Business services described in the previous model, become more detailed in the extended use case model. In this model are described both the application’s features such as the possible services to be performed. Figure 77 presents the extended use case model for *perform image processing* business service. This model is detailed in 6 use cases that are necessary for the image processing. The use cases are:

³During our analysis, we will detail the *perform image processing* requirement, as a way to better understand the particularities of both methods.

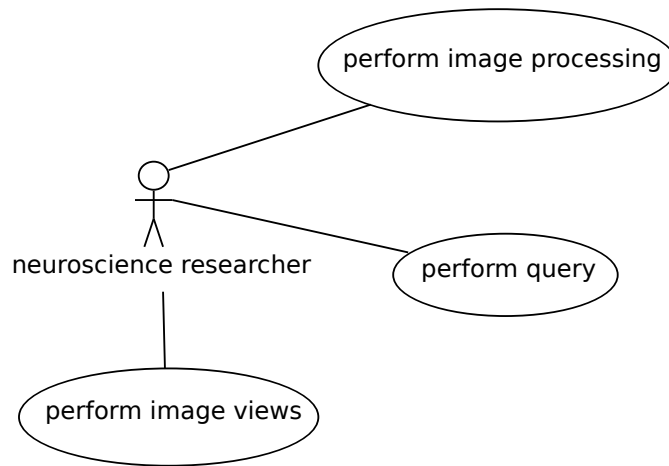


Figure 76: Use Case Model [39].

(i) perform image processing; (ii) proceed with the service payment; (iii) provide images; (iv) authenticate; (v) get results; and (vi) download results.

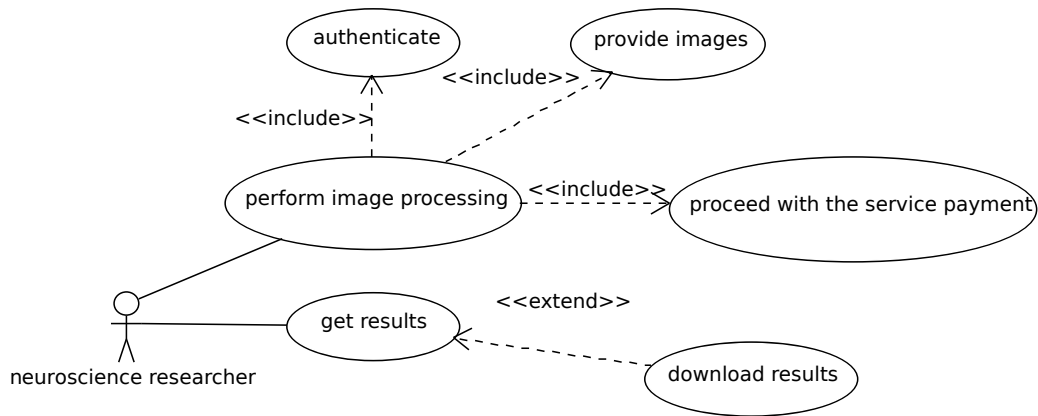


Figure 77: Extended Use Case Model [39].

Once identified the different functionalities that are required by the system to perform business services, the service process model is used to represent the workflow necessary to perform a business service. SOD-M represents each business service in different diagrams, which are shown in the following figures. Figure 78, for the business service “*perform image processing*”. The activities represented in these figures are derived from the basic use cases represented in the model extended use case.



Figure 78: Image Processing - Service Process Diagram [39].

The service composition models represent the processes of composition of the various actions that are necessary to perform each business service, detailing which business collaborator performing each action.

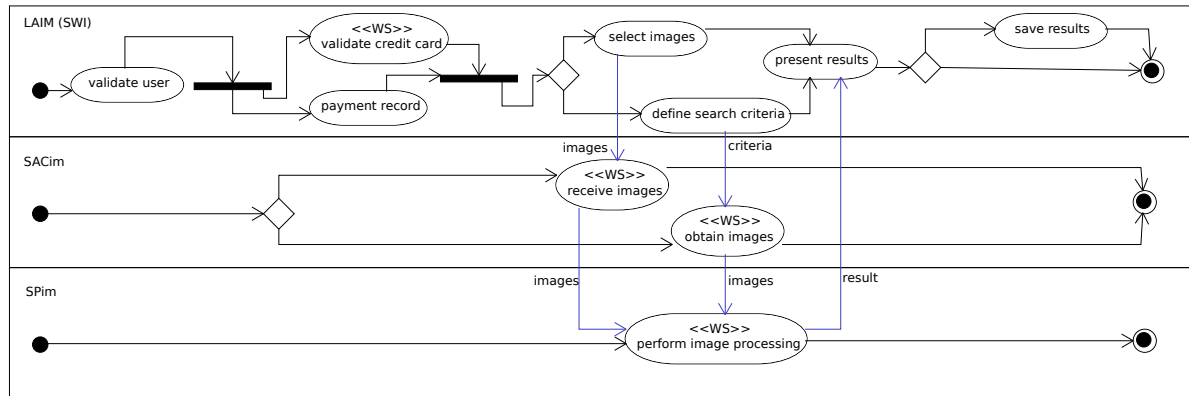


Figure 79: Image Processing - Service Composition Diagram [39].

Figure 79 present the details of the services composition for the business service we are modelling. It is important to realize that the service processes described previously, are refined by the execution of these service. Notice that SOD-M creates a workflow for each business service, independently. This workflow is a sequence of services that can be run considering the system features. Figure 79 presents that the workflow uses the services offered by LAIM to perform each function, according the SACim, SPim and SVim services. It is also important to note that the external application services are marked with the stereotype «WS».

The result of applying the SOD-M method is a detailed service composition model from the business use case, in our specific case the “*perform image processing*” business service.

5.5 Measurement and Evaluation of the π SOD-M Methodology

Based on [17], the measurement and evaluation process requires a mechanism for determining what data is to be collected; why it is to be collected; and how the collected data is to be interpreted. The purpose of the measurement and evaluation flows from the needs of the organization; the need to better understand resource utilization in order to improve cost estimation; or the need to evaluate the benefits, advantages and drawbacks of a research project.

The data collection process itself requires a basic paradigm that traces the goals of the collected process, *i.e.* the reasons the data are being collected. [17, 63] describe that the need for information must be quantified whenever possible and the quantification analyzed as to

whether or not it satisfies the needs. This quantification of the goals should then be mapped into a set of data that can be collected.

Each evaluation *goal* generates a set of questions that attempt to define and quantify the specific goal which is at the root of its goal tree. The *goal* is only as well defined as the *question* that it generates. Each *question* generates a set of *metrics* or *distributions of data*.

The paradigm proposed by [17] defines a set of six steps to the data collection:

1. Generate a set of goals;
2. Derive a set of questions of interest or hypotheses which quantify those goals;
3. Develop a set of data metrics and distributions which provide the information needed to answer the question of interest;
4. Define a mechanism for collecting the data;
5. Perform a evaluation of the data;
6. Analyze the data collected to answer the question posed.

Aiming to collect data that help in an initial analysis of our methodology through three examples described in this chapter, we will use the approach proposed by Basili [17] to extract these data and analyze if the goals of our proposal have been achieved and how it is possible improve the π SOD-M methodology.

Generate a set of goals: First we present the goals we have, when we intend to measurement and evaluate the π SOD-M methodology. The goals are:

1. Verify if the identification of non-functional requirements at initial levels of development improves the modeling of reliable web services.
2. Verify how to refine application constraints into specification code, and if it improves the software quality.
3. Verify if it is possible to reuse code generated for requirements non-functional.
4. Verify how the non-functional requirements are specified in the methodology.

Table 8: π SOD-M's Goals and Questions

Goals	Questions
1. Verify if the identification of non-functional requirements at initial levels of development improves the modeling of reliable web services.	<p>1. What are the levels of modeling presented in the methodology? And how the NFRs are modelled?</p> <p>2. What is the goal of each modeling level?</p> <p>3. Which kind of artifact is generated at each level of development?</p>
2. Verify how to refine application constraints into specification code, and if it improves the software quality.	<p>1. What kind of modelling approach is used in the methodology?</p> <p>2. How are modeled the non-functional requirements?</p>
3. Verify if it is possible to reuse code generated for requirements non-functional.	<p>1. Which level of development the code is generated? and what kind of code is generated?</p> <p>3. The code is generated automatically or manual?</p>
4. Verify how the non-functional requirements are specified in the methodology.	<p>1. How are classified the non-functional requirements?</p>

Set of questions of interest which quantify the goals: Following the approach proposed in Basili [17], the questions are related to each goal. Table 8 present the questions related to the goals of the π SOD-M evaluation analysis.

Set of data metrics and distributions: Still following the approach proposed by Basili [17], we identify metrics with the objective of quantify the questions in order to generate data that can be analyzed and evaluated in order to verify that the goals of the evaluation were satisfied, or not. The metrics are listed by goal/question (for example #1.2. This metric is related to *goal* 1, and *question* 2 previously identified). The statements for measure the data are shown in table 9.

Collecting the data: After the development of metrics and the types of data to be collected, the extraction was performed considering the modeling steps of π SOD-M for each case study presented in this chapter. Table 10 presents the results of the data collection.

Table 9: π SOD-M's Metrics/Distributions

Goal/Question	Metrics/Distributions
#1.1	1. How many levels the methodology defines? 2. How many models are generated at each level?
#1.2	1. How many restrictions were modeled (by level)?
#1.3	1. How many artifacts were generated (by level)?
#2.1	1. How long (in days) was spent for the modeling of each case study?
#2.2	1. How many RNFs were modeled using the approach?
#3.1	1. How many lines of code were generated for specify the non-functional requirement identified?
#3.2	1. What is the total of line of code generated?
#4.1	1. How many types of non-functional requirements were identified? 2. Which non-functional requirements were identified?

Table 10: Results: π SOD-M's Metrics/Distributions

Metrics/Distributions	π SOD-M		
	Example 1	Example 2	Example 3
#1.1.1	4	4	4
#1.1.2	1-1-1-1	2-2-2-2	3-1-1-1
#1.2.1	5c-8a-6r-2co	7c-9a-7r-5co	4c-6a-6r-3co
#1.3.1	1	1	1
#2.1.1	8	10	5
#2.2.1	2	3	3
#3.1.1	9	37	20
#3.2.1	29	56	40
#4.1.1	2	3	3

Evaluation of the data: Tables 9 and 10 present the questions and data generated from the evaluation of each example. All examples were modeled in accordance with the proposed methodology and yielded the following results.

The methodology defines 4 levels of modeling (*#1.1.1*), however the case studies generated a different number of models (π -UseCase, π -ServiceProcess, π -ServiceComposition e π -PEWS). In *Example 1* was generated 1 model of each type, in *Example 2* were generated two models of each type, and in *Example 3* were generated 3 π -UseCase models and 1 referring to the others (*#1.1.2*). We found that this variation has no involvement with the identification of the number of non-functional requirements.

Regarding the types of restrictions modeled at each level, we find that the number constraints identified implies an increase in the number of lines of code. The *Example 1* set five

constraints in the π -UseCase model, 8 *assertions* in the π -ServiceProcess model, 6 *rules* in the π -ServiceComposition model and 2 *contracts* in the π -PEWS model (5c-8a-6r-2Co⁴). This example generated 29 lines of code specification. However, the *Example 2* presents the following data 7c-9a-7r-5co, generating 56 lines of specification code. It was observed that the number of restriction increases the number of lines specification. The *Example 3* presents the following data, 4c-6a-6r-3co, generating 40 code lines specification (#1.2.1, #3.1.1 and #3.2.1).

According to the data analysis, we also found that the number of lines of code for the definition of contracts increase according to the numbers of rules identified (#1.2.1).

The number of days spent for modeling also varied according to the number of constraints. Thus, the time spent with identification and modeling of the constraints implies a greater number of days of development. *Example 2* was modeled in 10 days (#2.1.1) and this example showed a greater number of restrictions and non-functional requirements. We assume that if these restrictions were not modeled since the initial stages, the time spent on maintenance could be higher. *Example 3* had the fewest number of restrictions, and therefore took less time to be modeled. Thus the number of non-functional requirements of an application implies the time spent for the project development using π SOD-M (#2.2.1 and #4.1.1).

Conclusions: We conclude that the identification of NFRs in initial levels of modelling helps the development of services, mainly because the restrictions can be refined and transformed into code specification. This favors the prior identification of errors, and help in the classification of the non-functional requirements.

Classify restrictions as constraints, then refine this concept into assertions, and then group it into policy improves identifying types of non-functional requirements, providing reusability of services and better use of the code specification.

We also found that the use of the method provides identification of different types of NFRs. These requirements are translated into contracts and can be checked at runtime. We can see that the number of lines of code generated for each type of non-functional requirement does not have a regular variation, and it does not generates apparent side effects.

5.5.1 Comparison of the SOD-M and π SOD-M approaches

We also perform a comparative analysis between π SOD-M and SOD-M, considering other aspects. Table 11 presents the data related with the three examples used for the π SOD-M

⁴5c = 5 constraints, assertions 8th = 8; 6r = 6 rules; 2Co contracts = 2

Table 11: Comparison: SOD-M and π SOD-M

	SOD-M			π SOD-M		
	Ex. 1	Ex. 2	Ex. 3	Ex. 1	Ex. 2	Ex. 3
Use Case	9	8	5	13	14	9
Action	7	7	5	15	12	11
Business Collaborator	4	3	3	4	3	3
NFRs modelled	-	-	-	2	3	3
Lines of Code	20	19	20	29	56	40
Transformations	2	2	2	3	3	3

evaluation: (i) To Publish Music example (*Ex. 1*), (ii) Crime Map example (*Ex. 2*), and (iii) GesIMED example (*Ex. 3*). The result reflects the difference between the two approaches (SOD-M and π SOD-M) with the intention of verifying how π SOD-M can improve the modeling process of service-oriented applications in the presence of non-functional requirements.

The aspects analyzed and presented in table 11 are: (i) the number of use cases required for modeling both, the restrictions and requirements of the application, (ii) how many actions are required for the representation of the business process, (iii) how many business collaborators are needed to the complete execution of the workflow, (iv) the number of non-functional requirements that were identified during the modeling of the application, (v) how many lines of code were generated from the service composition model of each example, and (vi) how many model transformations held throughout the development process.

Considering the number of use cases for modeling the examples presented in this chapter, the models from the π SOD-M approach present more use cases than the models in SOD-M, because using π SOD-M it is possible to model the restrictions of the application through stereotyped use cases (*constraint*). Due to this reason, example 1 has 9 use cases using SOD-M, while using π SOD-M, the model has 13 use cases. The difference (4 use cases) between both approaches represents the amount of restrictions that have been identified at this level through the π -UseCase model. In example 2, it has 8 use cases using SOD-M while using π SOD-M it has 13 use cases. A difference of 5 use cases. In the third example, it presents 5 use cases using SOD-M, and 9 cases through π SOD-M. π SOD-M presents a greater number of use cases because this model also represent the constraints of the application, even in the most abstract level. We found that in spite of a relatively greater effort modeling, π SOD-M describes functional and non-functional aspects, while SOD-M models only functional aspects.

In relation to the number of actions, the π SOD-M approach provides a larger number, be-

cause it needs to describe pre- and post-conditions for the services. The larger number of actions in π SOD-M favors the verification of restrictions during the execution of the workflow. These actions are the result of a model transformation. The use cases marked with *constraint* stereotype are transformed into actions with the *assertion* stereotype. It is important to realize in the table that in the first example, the number of action ranges from 7 to 15, the second 7 to 12 and in the third example ranges from 5 to 11. The constraint checking through assertions favors the representation of non-functional requirements necessary for the workflow execution.

Considering the number of *Business Collaborator*, this value does not suffer any variation between both approaches, because of the services to be performed in the 3 examples in both approaches are the same. Although the number of use cases and actions vary, it does not imply the variation of the amount of external services (*Business Collaborator*) required for execute the application.

Since the goal of SOD-M and π SOD-M approaches are different, analyze the number of non-functional requirements modelled is not comparable, however points to an evolution and improvement of the SOD-M approach. Using SOD-M, it is not possible to identify any non-functional requirement during the modeling. However, using π SOD-M, it is possible to model different non-functional requirements and refine them. From this point of view, both approaches are not comparable because they have different focus of development, otherwise the proposal presented in π SOD-M adds reasonable value considering the modeling of non-functional requirements for service-oriented applications.

Regarding the specification code for each example, we analyzed the difference of lines of code (π -PEWS language) generated in both approaches. SOD-M only considers the specification of functional aspects, and π SOD-M consider both aspects, functional and non-functional, thereby we compare the π -PEWS specification for each example, considering the generation of the workflow (SOD-M) and the generation of contracts (π SOD-M). The number of lines of specification grows in π SOD-M due to the fact that this approach generates contracts which defines pre- and post-conditions for the services. The growth depends on the number of constraints identified during the modeling. There is no standard concerning the difference in terms of the number of lines generated. For example, in the first example the difference between the approaches is 9 lines of code, while in the second example the difference is 37 lines. In the example 3 the difference is 20 lines of code, where 20 lines were generated using the SOD-M approach and 40 lines using π SOD-M. We can conclude is that the number of non-functional requirements modeled implies a greater number of lines of specification code, to realize the examples 2 and 3, both has three non-functional requirements and it implied a larger number of

lines in π -PEWS specification, 56 e 40 respectively.

Another factor to be considered is the number of transformations in each approach. In the modeling of all the examples in SOD-M were performed two transformations at PIM level, whereas in the modeling of the examples using π SOD-M were performed three transformation, 2 in the PIM level and 1 on PSM level. The larger number of transformations in π SOD-M represents a step of refinement and the generation of code for a specific platform.

Despite the numbers presented in table 11 show that there is a larger effort in the use of the π SOD-M methodology, compared with SOD-M, this is due to the fact that non-functional requirements and restrictions on services are made from the most abstract levels of modeling. This favors the early identification of system restrictions and improves the refinement of transversal aspects and constraints of the application. The π SOD-M methodology adds reasonable value and new concepts regarding SOD-M, improving and facilitating the description of quality requirements for service-oriented applications.

5.6 Conclusions

This chapter presented the use of the π SOD-M methodology, as a form of empirical validation. We described, modeled and implemented three example of application development to use concepts, model properties and detail the process development. The methodology and its concepts supported the development of applications that use service compositions, mainly in the identification and refinement of quality requirements. The use of π SOD-M and its environment can help to identify the non-functional requirements and improve the system modeling and specification process.

We conducted, in this chapter, two types of evaluations, the first followed the proposal based on Basili [17] for data evaluation. And the second was a comparison between π SOD-M and SOD-M, considering different aspects.

The development of applications by using SOD-M favors and improves the modelling of services in more abstract levels of development and help to identify non-functional requirements related with each of these services.

The transformations of models enables the semi-automatic development of applications. Owing to the generation of the system specification code from the service composition model, the designer and developer can focus on the description and detail of functional and non-functional requirements. The generated code contains the expression of the interaction with

services and their associated namespaces. This specification can be executed after the model transformation.

As the process is iterative and incremental, after the generation of each model it is possible to make adjustments to the generated model. For example, after the π -UseCase model design and the generation of π -ServiceProcess model, the designer can make adjustments and improve the model so that it becomes more expressive.

6 *Conclusions*

Nowadays , access to software, hardware and network resources is done more and more through services. From this point of view, building applications consists in composing services according to a set given requirements. In order to ensure a desired level of quality, applications must comply to specified requirements and reliability properties. Ensuring these properties for applications composed by heterogeneous services provided under different conditions is not an easy task and imposes challenges for systems developers.

Non-functional requirements are related to business rules associated to the general semantics of the application. In the case of service-based applications, they also concern the compliance constraints imposed by the services. Having such business rules associated to the compound service can help to ensure that the resulting application fulfills requirements of the user.

The main goal of this work was to provide tools for helping the development of reliable service-based applications. Our work targets the specification and programming web application, including non-functional aspects (i.e., atomicity, security, exception handling, persistence). In contrast to approaches such as WS-* [30], our work specifies *policies* for service compositions in an orthogonal way, *i.e* separating the specification of non-functional requirements from the main functionalities of the application. The use of the WS-* standards suppose that non-functional requirements are implemented according to the knowledge that a programmer has of a specific application requirements, without deriving them in a methodological way, thus leading to *ad-hoc* solutions that can be difficult to reuse. In our approach, once defined policies for a given application, they can be reused and/or specialized for another application with similar requirements.

Our work also enables the modeling of non-functional properties and their associated recovery actions and the way they are associated to a service composition expressed as a workflow. In the modeling phase of our methodology, non-functional properties are seen as business rules and constraints (e.g. privacy requirements with respect to data used within the process, temporal constraints on the communication with a service) specified semi-formally. These requirements

are part of a business process specification (e.g., online purchase process). Along the phases of the methodology these business rules and constraints are modeled using more concrete concepts that lead to their implementation in executable code.

In general, existing approaches dealing with non-functional requirements in the distributed systems, the database and the software engineering domains, suppose a good knowledge of the programmer who is in charge of implementing and running the non-functional services according to given application requirements. Our work benefits from these previous results in order to provide a software oriented methodology that can include in the different development phases the specification of non-functional properties, their modeling and implementation. Thereby, our methodology provides meta-concepts that can be specialized for modelling different non-functional properties (security, atomicity, exceptions, persistency) and associated to a service composition.

6.1 Main Contributions

This thesis presented a methodology for specifying and designing reliable service-based applications. Our proposal deals with functional and non-functional requirements separately. We model and associate policies that represent both systems' cross-cutting aspects and constraints stemming from the component services.

π SOD-M is proposed for the development of service-oriented applications, helping to ensure quality requirements. This work also proposes a categorization of concepts for the modeling and refinement of non-functional properties, e.g. *constraint*, *contract*, *assertion*, *policy*, *rules*, *non-functional requirement* and *non-functional attribute*.

We evaluate the methodology by means of three case studies, showing that the development process based on π SOD-M produces satisfactory results with respect to the modeling and refinement of quality requirements. We can conclude that grouping contracts with non-functional properties into a single policy, produces a more effective result in the generation of the specification, and therefore in the verification of these properties.

Another contribution of our proposal is the integration of the methodology concepts in a MDA-based development. π SOD-M is a method that proposes meta-models at different levels (CIM, PIM and PSM) and extending the PSM meta-models. π SOD-M is aimed at the design and development of reusable service-based applications.

A set of transformations at the different levels are also proposed in our methodology. These

transformations refine the models from an abstract representation of requirements into programming code.

We extended the PEWS language [105] (into π -PEWS) to add contracts specification and temporal modeling restrictions for services.

We provide a platform to give support to our methodology. The π SOD-M environment is implemented by a set of Eclipse plugins. All the transformation defined in our methodology are supported by the implementation. Our tool performs static verification in order to check the existence of the functions of the services of the composition [104].

6.2 Future Work

This thesis proposes a comprehensive methodology that can offer a range of possibilities for extension in future work. Future work is organized into *validation*, *development* and *research tasks*:

Considering the evaluation and validation of the proposed methodology the future work should address a more detail evaluation of π SOD-M methodology in order to perform a more quantitative analysis. This will require compare the generated code and implementation specifications with different languages, and this language can be added to meta-models structure in the PSM level. Thus, the methodology will generate executable code for different languages. This will requires adjustments meta-model with the PIM level.

Describe software product lines related with reliable services applications that could be generated from the use of π SOD-M may also be considered as future work. As well as, analyze the variability of non-functional requirements proposed by π SOD-M.

Considering the development area, the future work should address the improvement of the development environment that support the methodology, with visual tools and with execution environment, running in different platforms. This will provide a more usable environment. Another aspect to be considered is the development of the π -PEWS environment (back-end) to execute the specification generated by the methodology.

Finally, considering research tasks, future work should address the formalization of the methodology transformation rules at different levels. Another aspect to be added to the methodology is the definition of new meta-models in the CIM level to represent business and system requirements, despite the original proposal offer *e-value* and *BPMN* models. It would be important to do a more thorough search in regarding these requirements to represent the independent

level computing.

References

- [1] Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*, page i, 1998.
- [2] *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP: A Practitioner's Guide to the RUP*. Addison-Wesley, 2003.
- [3] S. Abiteboul, O. Benjelloun, and T. Milo. Towards a flexible model for data and web services integration. In *proc. Internat. Workshop on Foundations of Models and Languages for Data and Objects*, 2001.
- [4] J. R. Abrial, M. K. O. Lee, D. Neilson, P. N. Scharbach, and I. H. Sorensen. The b-method. In Soren Prehn and W. J. Toetenel, editors, *VDM Europe (2)*, volume 552 of *Lecture Notes in Computer Science*, pages 398–405. Springer, 1991.
- [5] M. E. Adiba and J. M. Andrade. Update consistency and parallelism in distributed databases. In *International Conference on Distributed Computing Systems*, ICDCS, pages 180–187, 1981.
- [6] S. Agarwal, S. Lamparter, and R. Studer. Making web services tradable: A policy-based approach for specifying preferences on web service properties. *J. Web Sem.*, 7(1):11–20, 2009.
- [7] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
- [8] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeranwarana. Business process execution language for web services. Available at <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>, 2003.
- [9] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimek. Web service choreography interface. Technical report, World Wide Web Consortium, 2002.
- [10] A. Arsanjani. SOMA: Service-Oriented Modeling and Architecture. Technical report, IBM, Available in [<http://www.ibm.com/developerworks/library/ws-soa-design1/>](http://www.ibm.com/developerworks/library/ws-soa-design1/), 2004.
- [11] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Gariapathy, and K. Holley. Soma: a method for developing service-oriented solutions. *IBM Syst. J.*, 47(3):377–396, July 2008.

- [12] D. Austin, A. Barbir, E. Peters, and S. Ross-Talbot. Web services choreography requirements. Available at <http://www.w3.org/TR/2004/WD-ws-chor-reqs-20040311/>, March 2004. W3C Working Draft.
- [13] C. Ba, M. A. Carrero, M. Halfeld-Ferrari, and M. A. Musicante. PEWS: A New Language for Building Web Service Interfaces. *J. UCS*, 11(7):1215–1233, 2005.
- [14] C. Ba, M. Halfeld-Ferrari, and M. A. Musicante. Composing web services with PEWS: A trace-theoretical approach. In *IEEE European Conference on Web Services (ECOWS)*, pages 65–74, 2006.
- [15] S. M. Babamir, S. Karimi, and M. R. Shishechi. A broker-based architecture for quality-driven web services composition. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4, dec. 2010.
- [16] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web services conversation language (wscl) 1.0. Available at <http://www.w3.org/TR/2002/NOTE-wscl10-20020314/>, 2002.
- [17] V. Basili. Quantitative evaluation of software methodology. Technical Report TR-1519, University of Maryland, July 1985.
- [18] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, January 2006.
- [19] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Owl web ontology language. Technical report, W3C - <http://www.w3.org/TR/owl-ref/>, 2004.
- [20] K. Belhajjame, C. Collet, and G. Vargas-Solar. A flexible workflow model for process-oriented applications. In *Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01) Volume 1 - Volume 1*, WISE '01, pages 72–, Washington, DC, USA, 2001. IEEE Computer Society.
- [21] M. Bell. *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*. John Wiley, 2008.
- [22] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi, and S. Guinea. A guided tour through savvy-ws: A methodology for specifying and validating web service compositions. In *Lipari Summer School*, pages 131–160, 2007.
- [23] E. Börger and A. Cisternino, editors. *Advances in Software Engineering, Lipari Summer School 2007, Lipari Island, Italy, July 8-21, 2007, Revised Tutorial Lectures*, volume 5316 of *Lecture Notes in Computer Science*. Springer, 2008.
- [24] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (soap). Technical report, World Wide Web Consortium, 2000.
- [25] A. Brown. SOA Development Using the IBM Rational Software Development Platform: A Practical Guide. In *Rational Software*, 2005.

- [26] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll. An overview of JML tools and applications. *Int. J. Softw. Tools Technol. Transf.*, 7(3):212–232, 2005.
- [27] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '08, pages 261–272, New York, NY, USA, 2008. ACM.
- [28] S. Ceri, F. Daniel, M. Matera, and F. M. Facca. Model-driven development of context-aware web applications. *ACM Trans. Internet Technol.*, 7(1), February 2007.
- [29] S. Chollet and P. Lalanda. An extensible abstract service orchestration framework. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 831–838, Washington, DC, USA, 2009. IEEE Computer Society.
- [30] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. Technical report, World Wide Web Consortium, 2001. Available in <http://www.w3.org/TR/wsdl>.
- [31] L. Chung. Representation and utilization of non-functional requirements for information system design. In *International Conference on Advanced Information Systems Engineering*, CAiSE, pages 5–30, 1991.
- [32] L. Chung and J. C. Leite. Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.
- [33] L. Chung, B. Nixon, E. S. K. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, 1999.
- [34] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Requirements Engineering, 14th IEEE International Conference*, pages 39–48, sept. 2006.
- [35] L. M. Cysneiros, J. C. Leite, and J. M. Sabat Neto. A framework for integrating non-functional requirements into conceptual models. *Requir. Eng.*, 6(2):97–115, 2001.
- [36] U. S. da Costa, A. M. Moreira, M. A. Musicante, and P. A. Souza Neto. Jcml: A specification language for the runtime verification of java card programs. *Sci. Comput. Program.*, 77(4):533–550, 2012.
- [37] A. D'Ambrogio. A model-driven wsdl extension for describing the qos of web services. In *ICWS*, pages 789–796, 2006.
- [38] L. de Alfaro and T. A. Henzinger. Interface automata. volume 26, pages 109–120, New York, NY, USA, September 2001. ACM.
- [39] M. V. de Castro. *Aproximación MDA para el Desarrollo Orientado a Servicios de Sistemas de Información Web: Del Modelo de Negocio al Modelo de Composición de Servicios Web*. PhD thesis, Universidad Rey Juan Carlos - Escuela Técnica Superior de Ingeniería de Telecomunicación, 2007.

- [40] V. De Castro, E. Marcos, and J. M. Vara. Applying cim-to-pim model transformations for the service-oriented development of information systems. *Inf. Softw. Technol.*, 53(1):87–105, January 2011.
- [41] V. de Castro, E. Marcos, and R. Wieringa. Towards a service-oriented mda-based approach to the alignment of business processes with it systems: From the business model to a web service composition model. *International Journal of Cooperative Information Systems*, 18(2), 2009.
- [42] W. Derks, W. Jonker, J. Dehnert, and P. Grefen. Customized atomicity specification for transactional workflows. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications*, CODAS '01, pages 140–, Washington, DC, USA, 2001. IEEE Computer Society.
- [43] N. Dhyanes, G. C. Vineel, and S. V. Raghavan. Devise: A methodology for building web services based infrastructure for collaborative enterprises. In *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '03, pages 12–, Washington, DC, USA, 2003. IEEE Computer Society.
- [44] G. Di Modica, O. Tomarchio, and L. Vita. Dynamic slas management in service oriented environments. *J. Syst. Softw.*, 82(5):759–771, May 2009.
- [45] V. Diamadopoulou, C. Makris, Y. Panagis, and E. Sakkopoulos. Techniques to support web service selection and consumption with qos characteristics. *J. Netw. Comput. Appl.*, 31(2):108–130, April 2008.
- [46] J. A. Espinosa-Oviedo, G. Vargas-Solar, J. L. Zechinelli-Martini, and C. Collet. Non-functional properties and services coordination using contracts. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*, IDEAS '09, pages 307–310, New York, NY, USA, 2009. ACM.
- [47] J. A. Espinosa-Oviedo, G. Vargas-Solar, J. L. Zechinelli-Martini, and C. Collet. Policy driven services coordination for building social networks based applications. In *In Proc. of the 8th Int. Conference on Services Computing (SCC'11), Work-in-Progress Track*, Washington, DC, USA, July 2011. IEEE.
- [48] J. Fabra, V. De Castro, P. Alvarez, and E. Marcos. Automatic execution of business process models: Exploiting the benefits of model-driven engineering approaches. *Journal of Systems and Software*, 85(3):607 – 625, 2012. <ce:title>Novel approaches in the design and implementation of systems/software architecture</ce:title>.
- [49] G. Feuerlicht and S. Meesathit. Towards software development methodology for web services. In *Proceedings of the 2005 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the fourth SoMeT_w05*, pages 263 – 277, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.
- [50] H. Fujita and M. Mejri, editors. *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Fifth SoMeT 2005, September 28-30, 2005, Tokyo, Japan*, volume 129 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2005.

- [51] M. P. Gervais. Towards an mda-oriented methodology. In *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, COMPSAC '02, pages 265–270, Washington, DC, USA, 2002. IEEE Computer Society.
- [52] M. Glinz. Rethinking the notion of non-functional requirements. In *in Proceedings of the Third World Congress for Software Quality (3WCSQ'05)*, pages 55–64, 2005.
- [53] J. Gordijn and H. Akkermans. Value based requirements engineering: Exploring innovative e-commerce ideas. *REQUIREMENTS ENGINEERING JOURNAL*, 8:114–134, 2002.
- [54] J. Greene and A. Stellman. *Applied software project management*. O'Reilly, first edition, 2005.
- [55] ATLAS Group. Atl: Atlas transformation language. Technical report, ATLAS Group, LINA & INRIA, February, 2006.
- [56] C. Gutiérrez, D. G. Rosado, and E. Fernández-Medina. The practical application of a process for eliciting and designing security in web service systems. volume 51, pages 1712–1738, Newton, MA, USA, December 2009. Butterworth-Heinemann.
- [57] R. Hamadi and B. Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference - Volume 17*, ADC '03, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
- [58] R. Heckel and M. Lohmann. Towards contract-based testing of web services. In Mauro Pezzé, editor, *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)*, volume 116, pages 145–156, 2005.
- [59] Ariba Inc., IBM Corp., , and Microsoft Corp. Universal description, discovery, and integration (uddi). Technical report, UDDI.org, 2000.
- [60] B. Jeong, H. Cho, and C. Lee. On the functional quality of service (fqos) to discover and compose interoperable web services. *Expert Syst. Appl.*, 36(3):5411–5418, April 2009.
- [61] J. Jeston and J. Nelis. *Business Process Management, Second Edition: Practical Guidelines to Successful Implementations*. Elsevier, 2008.
- [62] B. Kitchenham, H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Softw. Engg.*, 13(1):97–121, February 2008.
- [63] J. Kontio, G. Caldiera, and V. Basili. Defining factors, goals and criteria for reusable component evaluation. In *CASCON*, page 21, 1996.
- [64] G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D. R. Cok. How the design of jml accomodates both runtime assertion checking and formal verification. In *FMCO*, pages 262–284, 2002.
- [65] G. T. Leavens, J. R. Kiniry, and E. Poll. A jml tutorial: Modular specification and verification of functional behavior for java. In *CAV*, page 37, 2007.

- [66] S. Liu, R. Khalaf, and F. Curbera. From DAML-S Processes to BPEL4WS. *Research Issues in Data Engineering, International Workshop on*, 0:77–84, 2004.
- [67] D. Mairiza, D. Zowghi, and N. Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 10th ACM Symposium on Applied Computing, SAC '10*, pages 311–317, New York, NY, USA, 2010. ACM.
- [68] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. Payne. Owl-s: Semantic markup for web services. W3C Member Submission 22, 2004.
- [69] H. Martin, M. E. Adiba, and B. Defude. Consistency checking in object oriented databases: a behavioral approach. In *Selected Papers from the First International Conference on Information and Knowledge Management, Expanding the Definition of Database, CIKM '92*, pages 53–68, London, UK, UK, 1993. Springer-Verlag.
- [70] R. Mendes, P. F. Pires, F. C. Delicato, and T. V. Batista. Webflowah: an environment for ad-hoc specification and execution of web services-based processes. In *SAC*, pages 692–693, 2009.
- [71] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992.
- [72] B Meyer. Eiffel: Version 3 and beyond. In *Eiffel*, page 5, 1992.
- [73] B. Meyer. *Object-Oriented Software Construction, 2nd Edition*. Prentice-Hall, 1997.
- [74] B. Meyer and J. M. Nerson. *Object-Oriented Applications*. Prentice-Hall, 1993.
- [75] N. Milanovic. Contract-based web service composition framework with correctness guarantees. In *ISAS*, pages 52–67, 2005.
- [76] N. Milanovic. *Contract based Web service composition*. PhD thesis, Humboldt-Universitat - Berlin, 2006.
- [77] N. Milanovic. Service engineering design patterns. In *Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering, SOSE '06*, pages 19–26, Washington, DC, USA, 2006. IEEE Computer Society.
- [78] N. Milanovic and M. Malek. Architectural support for automatic service composition. In *Services Computing, 2005 IEEE International Conference on*, volume 2, pages 133 – 140 vol.2, july 2005.
- [79] N. Milanovic and M. Malek. Search strategies for automatic web service composition. *Int. J. Web Service Res.*, 3(2):1–32, 2006.
- [80] J. Miller and J. Mukerji. Mda guide. 2003.
- [81] R. Mohanty, V. Ravi, and M. R. Patra. Web-services classification using intelligent techniques. *Expert Syst. Appl.*, 37(7):5484–5490, July 2010.
- [82] M. A. Musicante and E. Potrich. Expressing workflow patterns for web services: The case of pews. 12(7):903–921, jul 2006.

- [83] M. A. Musicante, E. Potrich, and M. A. Carrero. A programming environment for web services. In *Proceedings of the 2008 ACM symposium on Applied computing, SAC '08*, pages 2363–2367, New York, NY, USA, 2008. ACM.
- [84] J. Musset, E. Juliot, and S. Lacrampe. *Acceleo référence*. Technical report, Obeo et Acceleo, 2006.
- [85] J. Mylopoulos, L. Chung, and B. A. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Software Eng.*, 18(6):483–497, 1992.
- [86] OMG. Business process model and notation (bpmn). Technical report, OMG, <http://www.omg.org/spec/BPMN/2.0>, 2011.
- [87] OMG. Business process model and notation (bpmn). Technical report, OMG, <http://www.omg.org/spec/BPMN/2.0>, 2011.
- [88] OMG. Omg unified modeling languagetm (omg uml), infrastructure. version 2.4.1. Technical report, OMG, August 2011.
- [89] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, April 2007.
- [90] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho. Knowledge based quality-driven architecture design and evaluation. *Information & Software Technology*, 52(6):577–601, 2010.
- [91] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11), 2007.
- [92] M. P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, WISE '03*, pages 3–, Washington, DC, USA, 2003. IEEE Computer Society.
- [93] M. P. Papazoglou and W. J. Heuvel, Van Der Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, July 2006.
- [94] M. P. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, editors. *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*, volume 6500 of *Lecture Notes in Computer Science*. Springer, 2010.
- [95] J. L. Pastrana, E. Pimentel, and M. Katrib. Qos-enabled and self-adaptive connectors for web services composition and coordination. *Computer Languages, Systems & Structures*, 37(1):2 – 23, 2011.
- [96] P. F. Pires, M. R. F. Benevides, and M. Mattoso. Building reliable web services compositions. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, pages 59–72, London, UK, UK, 2003. Springer-Verlag.

- [97] A. Portilla, T. Hanh, and J. A. Espinosa-Oviedo. Building reliable mobile services based applications. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 121–128, april 2008.
- [98] E. Ramollari, D. Dranidis, and A. J. H. Simons. A survey of service oriented development methodologies.
- [99] N. S. Rosa, P. F. Cunha, and G. R. Justo. An approach for reasoning and refining non-functional requirements. *Journal of the Brazilian Computer Society*, 10(1):59–81, 2004.
- [100] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer. *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley Publishing, 2008.
- [101] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *Proceedings of the IEEE International Conference on Web Services, ICWS '04*, pages 43–, Washington, DC, USA, 2004. IEEE Computer Society.
- [102] B. Schmeling, A. Charfi, and M. Mezini. Composing non-functional concerns in composite web services. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 331–338, july 2011.
- [103] I. Sommerville. *Software Engineering 6th Edition*. Addison Wesley, 2008.
- [104] P. A. Souza Neto, H. B. Medeiros, and R. Hallais Neto. Plugin extrator para verificação de composições pews. *HOLOS*, V.3:84–106, 2012. Available in <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/>.
- [105] P. A. Souza Neto, M. A. Musicante, G. Vargas-Solar, and J. L. Zechinelli-Martini. Adding Contracts to a Web Service Composition Language. *LTPD - 4th Workshop on Languages and Tools for Multithreaded, Parallel and Distributed Programming*, September 2010.
- [106] A. Stellman and J. Greene. *Applied software project management*. O’Reilly, 2005.
- [107] S. Thatte. Xlang: Web services for business process design. Technical report, 2001.
- [108] D. Thißen and P. Wesnarat. Considering qos aspects in web service composition. In *Computers and Communications, 2006. ISCC '06. Proceedings. 11th IEEE Symposium on*, pages 371–377, june 2006.
- [109] R. Tongrunrojana and D. Lowe. Wied: A web modelling language for modelling architectural-level information flows. *J. Digit. Inf.*, 5(2), 2004.
- [110] A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos. Extending sysml to explore non-functional requirements: the case of information system design. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 1057–1062, New York, NY, USA, 2012. ACM.
- [111] A. Uszok, J. M. Bradshaw, and R. Jeffers. Kaos: A policy and domain services framework for grid computing and semantic web services. In *iTrust*, pages 16–26, 2004.
- [112] W. M. P. Van der Aalst. Don’t go with the flow: Web services compositions standards exposed. *Issue of IEEE Intelligent System*, Jan/Feb 2003.

- [113] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.
- [114] H. M. W. Verbeek and W. M. P. Van der Aalst. Analyzing BPEL processes using Petri nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78, Miami, Florida, USA, 2005. Florida International University.
- [115] A. Watson. A brief history of MDA, 2008.
- [116] H. Xiao, B. Chan, Y Zou, J. W. Benayon, B. O’Farrell, E. Litani, and J. Hawkins. A framework for verifying sla compliance in composed services. In *ICWS*, pages 457–464, 2008.
- [117] G. Yeom, T. Yun, and D. Min. Qos model and testing mechanism for quality-driven web services selection. In *Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA’06)*, pages 199–204, Washington, DC, USA, 2006. IEEE Computer Society.
- [118] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE ’97, pages 226–, Washington, DC, USA, 1997. IEEE Computer Society.
- [119] X. Zhang, F. Parisi-Presicce, R. S. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.
- [120] L. Zhu and Y. Liu. Model driven development with non-functional aspects. In *Proceedings of the 2009 ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design*, EA ’09, pages 49–54, Washington, DC, USA, 2009. IEEE Computer Society.

APPENDIX A - Source Selection and Analysis

Method: Non-Functional Requirements for Service-Based Applications

We selected the sources proposed by [62] for searching primary studies. These sources contain the works published in journals, conferences and workshops which are of recognized quality within the research community. The search for bibliography was performed in the engines are: (i) *IEEE Computer*; (ii) *ACM Digital Library*; and (iii) *Science Direct*. For each of the the selected sources, we used the following search query criteria:

```
((("non functional properties") OR ("non functional requirements")) AND "web service" AND "composition"))
```

After define the sources' selection, we identify those works that provided direct evidence with regard to the research questions. Deciding for the inclusion and exclusion criteria for filtering the corpus works selection, we selected those related to non-functional requirements/properties, and quality for web service based applications. Initially, the selection criteria were interpreted liberally and clear exclusions were only made with regard to title, abstract and introduction.

Based on the guidelines mentioned in [62], we established a three-step with different selection criteria:

- Step 1 - the search string must be run on the selected search engine. An initial set of studies was obtained by filtering of title, abstract, and if necessary, introduction. All the studies were selected according to the inclusion and exclusion criteria. Studies which were not clearly related to any aspect of the research questions were excluded.

	IEEE Explorer	ACM Library	Science Direct	Total
Total results	65	271 (75 ¹)	166	502 (306 ²)
Step one - results selected	19	10	20	49
Step one - results selected (%)	29.23%	13.33%	12%	16%
Step two - results selected	7	3	9	19
Step two - results selected (%)	10.76%	4%	5.42%	6.20%

Table 12: Summary of the studies selected at each step.

- Step 2, the exclusion criteria were based on the following practical issues: non-English papers, non- International Conference papers and non-International Workshop papers. Specifically in the case of ACM library, we considered only the transaction journal works.
- Step 3, the papers selection process was based on detailed research questions (RQ_1 to RQ_7).

The information for each step was collected considering the 3 searchers and the the query presented previously. the results of each step were: (i) for each source a list of all the studies that fulfilled the query; (ii) a list of studies for each source which contained all the works that did not fulfill the second stage inclusion criteria; and (iii) the last step produced a list of works for each source which contained all the studies that fulfilled the second step (table 12).

The extraction of information was based on the research questions, and each work extraction question included the following items: (i) where the paper was found; (ii) identification of the title and main subjects; (iii) summary of the research; (iv) inclusion and exclusion criteria; (v) objective and result; and (vi) subjective results.

Table 12 shows a summary of the studies selected in each stage of the selection procedure for each source. The “Total results” were obtained by running the search string on the selected sources. The next four rows show the results obtained after applying stages one (2 first rows) and two (2 last rows) of the studies selection procedure.

In the first step, respecting the filters described, the 65 articles collected from IEEE, only 29.23% of them were in accordance with the criteria described early, representing 19 articles. In ACM Library, from 75 works collected, only 13.33% passed in the first stage filter, representing 10 articles. In Science Direct had the lowest percentage, totaling 20 of the 166 articles collected by the query, thus representing 12% of the total. Despite being the lowest relative value, the Science Direct had the largest absolute result, with 20 works in the first step. In the second stage, the percentage dropped further, and the relevant works and with accordance to the criteria have been collected as the final result. The results were respectively, 10.76%, 4% and 5.42% of total

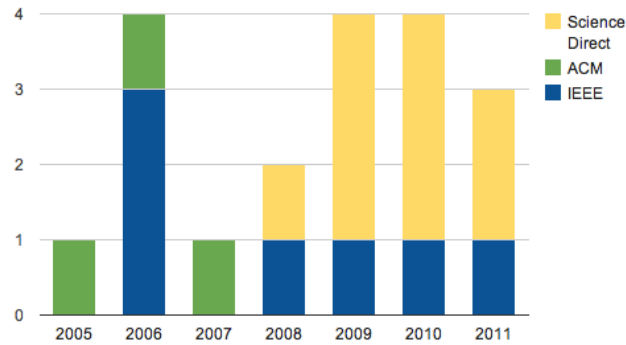


Figure 80: Publications per year.

from the IEEE, ACM and Science Direct. The highest percentage was among the works from IEEE, while the largest number of results, in absolute terms, was collected from Science Direct. The approaches resulting from this last stage were studied in depth and information concerning the detailed research questions and other fields of the extraction forms was extracted from each work we selected. 49 works were selected in the first stage, and, only 19 works in the second stage. It represents 6.20% of the total amount of works.

Figure 80 shows the publications per year, from 2005 to 2011. 12 of 19 articles were selected in the systematic review published in 2006, 2009 and 2010, being four in each year and 6 in Science Direct source, 5 in IEEE, and only 1 at the ACM. All nine of Science Direct were published in the last 4 years. Figure also shows that the number of publications that consider classification of NFR once again increased from 2008.

APPENDIX B - Service-Based Non-Functional Requirement Concepts

- **NON-FUNCTIONAL (NF) ATTRIBUTE** - An attribute that describes the quality or characteristics of a functional requirement. For example *confidentiality* and *privacy* may be non-functional attributes for the functional requirement *user registration*.
- **NON-FUNCTIONAL (NF) REQUIREMENT** - A group of semantically correlated non-functional attributes (NFA). For example, *security* is an NF Requirement that comprises attributes such as *confidentiality* and *integrity*.
- **CONSTRAINT** - A constraint prevents the system from achieving more than its goal. With the definition of constraints, the system can be more robust, and unexpected problems can be solved before they happen. For example, in a banking system, the customer can only withdraw money if they have positive balance in the account.
- **CONSTRAINT TYPE** - Represents the types of constraints that could be expressed, the types are: business and data (*value) constraints. When modeling a system requirement the analyst can identify if there are restrictions on business, or data, or both.
- **CONTRACT** - Is the formalization of obligations (requires) and benefits (ensures) of a function, service activity or component. The following questions can be used to define contracts: *What does it expect? What does it guarantee?* Contract can be crucial to software correctness that they should be part of the design process.
- **EXCEPTIONAL BEHAVIOUR** - Are alternative execution paths if any condition or restriction is not respected. For example, if a user's password is not correct after three attempts, the user's account is locked for security.
- **POLICY** - A policy is a set of rules applied to a particular scope. This scope can be defined as an action, an activity, a function or a workflow. A policy is a composition of contracts

applied to a non-functional application requirement. For example, a security policy of a system constraint includes authentication, access, data privacy, and so on.

- **USE CASE** - Represents a behavior that can be executed in order to realize (parts of) a functional requirement.
- **REQUIREMENT** - A requirement is a super type for functional and non-functional requirements. Thus, the use cases can be related to both types of requirements.
- **SERVICE ACTIVITY** - Represents a function of a software system or its component that are implemented through services. A Service Activity may be calculations, technical details, data manipulation and processing and other specific functionality that are available to be accessed on the Internet.

APPENDIX C - π -PEWS Language

π -PEWS specifies assertions that are checked during the execution of a program. These assertions describe pre- and post-conditions for a given operation or compound service. In the case these conditions are not verified, the contract defines correcting actions (described as a PEWS path expression). The grammar defining the new language is defined as follows: A π -PEWS program is similar to a PEWS program, but with the possibility of adding contract definitions at the end of the program. Path expressions, defining the service workflow, are described by the *Service* non-terminal of the grammar below. Since contracts and workflow are separate concerns, the syntax of path expressions remain unchanged, from the original version of the language.

(1) $Program ::= \llbracket (\text{"var"} \text{ id "=" } ArithExp)^* (\text{"service"} \text{ id "=" } Service)^* Service \text{ Contract}^* \rrbracket$

(2) $Service ::= \llbracket \text{id} \rrbracket \mid \llbracket Service \text{"."} Service \rrbracket \mid \llbracket Service \text{"+"} Service \rrbracket \mid \llbracket Service \text{"||"} Service \rrbracket$
 $\mid \llbracket Service \text{"*"} \rrbracket \mid \llbracket \text{"["} BooleanCondition \text{"]"} Service \rrbracket \mid \llbracket \text{"\{"} Service \text{"\}" } \rrbracket$

The definition of contracts is given below. It includes a name for the contract, as well as its four component sections:

- Target service of the contract: This section specifies the service to which the contract applies. This can be an operation or a compound service (which also defines a scope for the contract).
- Preconditions and their actions. This section defines the assertions that will be verified before the target service is executed.
- Post-conditions and their actions. This section defines the assertions that will be verified at the end of the target service execution.
- Time constraints for the services on the contract scope.

(3) $Contract ::= \llbracket \text{"def"} \text{"Contract"} Id \text{"\{"} IsAppliedTo \text{ Requires}^* \text{ Ensures}^* (\text{TimeConstraint})? \text{"\}" } \rrbracket$

The directive “**isAppliedTo:**” defines the target service and scope of the contract. This service is given by the identifier appearing next to the keyword. The target service can be a simple operation or a compound service. In the former case, the contract cannot contain any time restriction. In the case of a contract defined for a compound service, the operations and services that form the contract can participate of the time constraint expressions defined by the contract.

(4) $isAppliedTo ::= \llbracket \text{“isAppliedTo” “:” ident “;”} \rrbracket$

The “**requires**” and “**ensures**” parts of a contract define the pre-conditions (resp. post-conditions) to be checked for each contract. In the case of pre-conditions, they will be verified before the service is executed. Post-conditions will be checked after the execution of the service. In both cases, when the assertion fails, the associated action will be executed. Actions are defined as services, and written as PEWS path expressions.

(5) $Requires ::= \llbracket \text{“requires” BooleanCondition (“(” onFailureDo “)” }? \text{“;”} \rrbracket$

(6) $ensures ::= \llbracket \text{‘ensures’ BooleanCondition (“(” onFailureDo “)” }? \text{“;”} \rrbracket$

(7) $onFailureDo ::= \llbracket \text{‘onFailureDo’ Service} \rrbracket$

Time Constraints are defined as relational expressions build from the operators.

(8) $timeConstraint ::= \llbracket \text{‘timeConstraint’ ‘:’} (\llbracket \text{“meet”} \rrbracket \mid \llbracket \text{“start”} \rrbracket \mid \llbracket \text{“overlap”} \rrbracket) \llbracket \text{‘(’} (opName \mid timeConstraint) \text{‘,’} (opName \mid timeConstraint) \text{‘)’} \rrbracket$

The model of temporal relations proposed here follows the main ideas presented in [7, 20]. The model is used to impose additional restrictions to the order in which operations of a web service are performed at execution time. For example, the confirmation of the bank authorization request must arrive at most one minute after the service call.

A service composition time relation is represented by the following model: Let Ω be a set of web services $(\omega_1, \dots, \omega_n)$ to be composed and Θ , a set of temporal relations r_t that can be applied to Ω . Temporal relations are defined by predicates $r_t : \Omega \times \Omega$, where $r_t \in \{ \textit{meet}, \textit{overlap}, \textit{start} \}$.

For example, the expression $\textit{meet}(\omega_1, \omega_2)$ states that the execution of operation ω_2 will begin as soon as ω_1 finishes. The temporal relations are described below:

Meet. The restriction specified by $\textit{meet}(\omega_1, \omega_2)$ specifies that ω_2 will be executed immediately after ω_1 finishes.

Overlap. The constraint given by $overlap(\omega_1, \omega_2)$ states that execution of the two services overlap in time. The restriction also states that the service ω_1 initiates before ω_2 .

Start. The specification of $start(\omega_1, \omega_2)$ states that the two services have their execution in parallel and that they start at the same time.

APPENDIX D - To Publish Music Case Study Diagrams

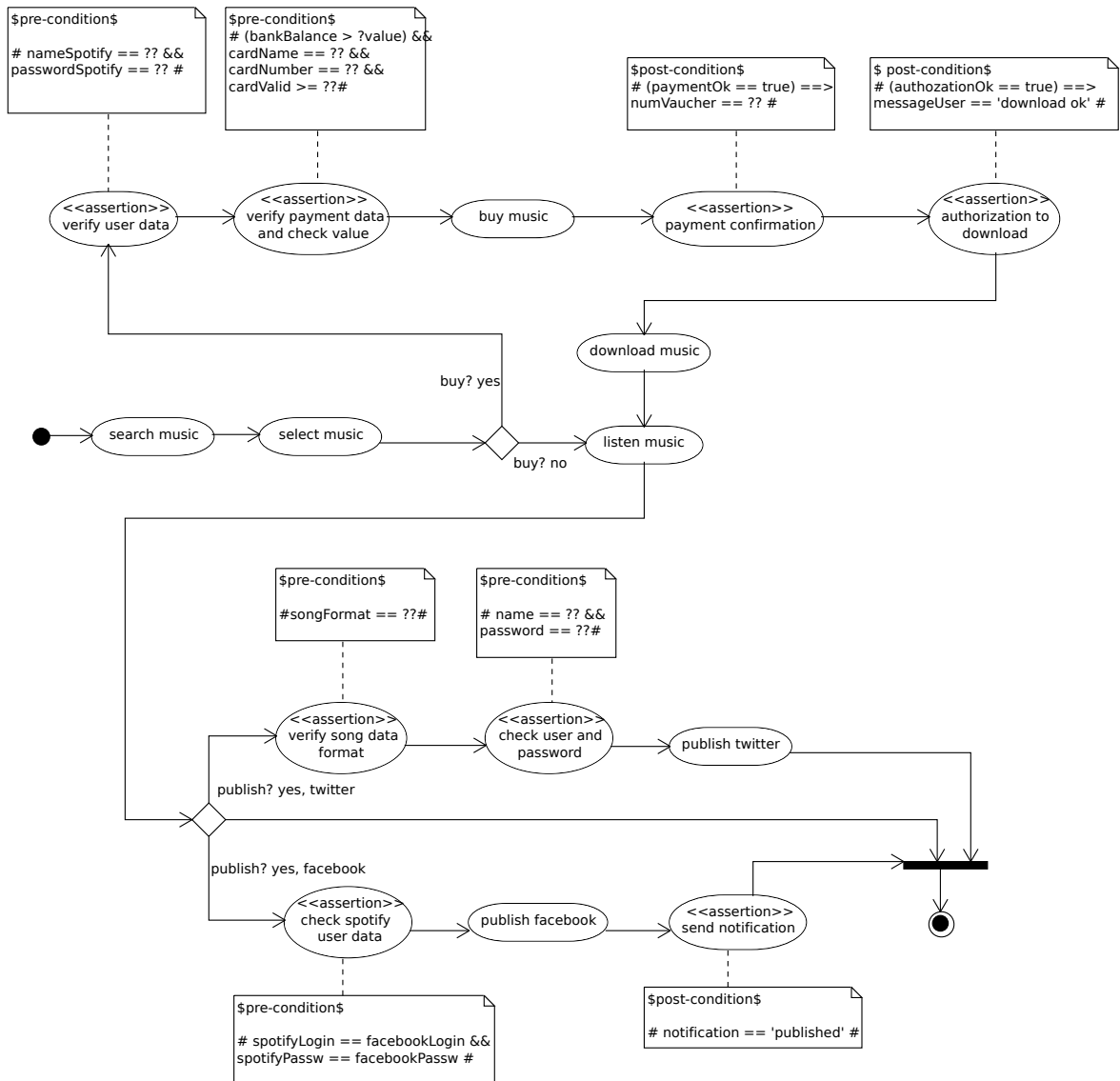


Figure 81: π -ServiceProcess - To Publish Music.

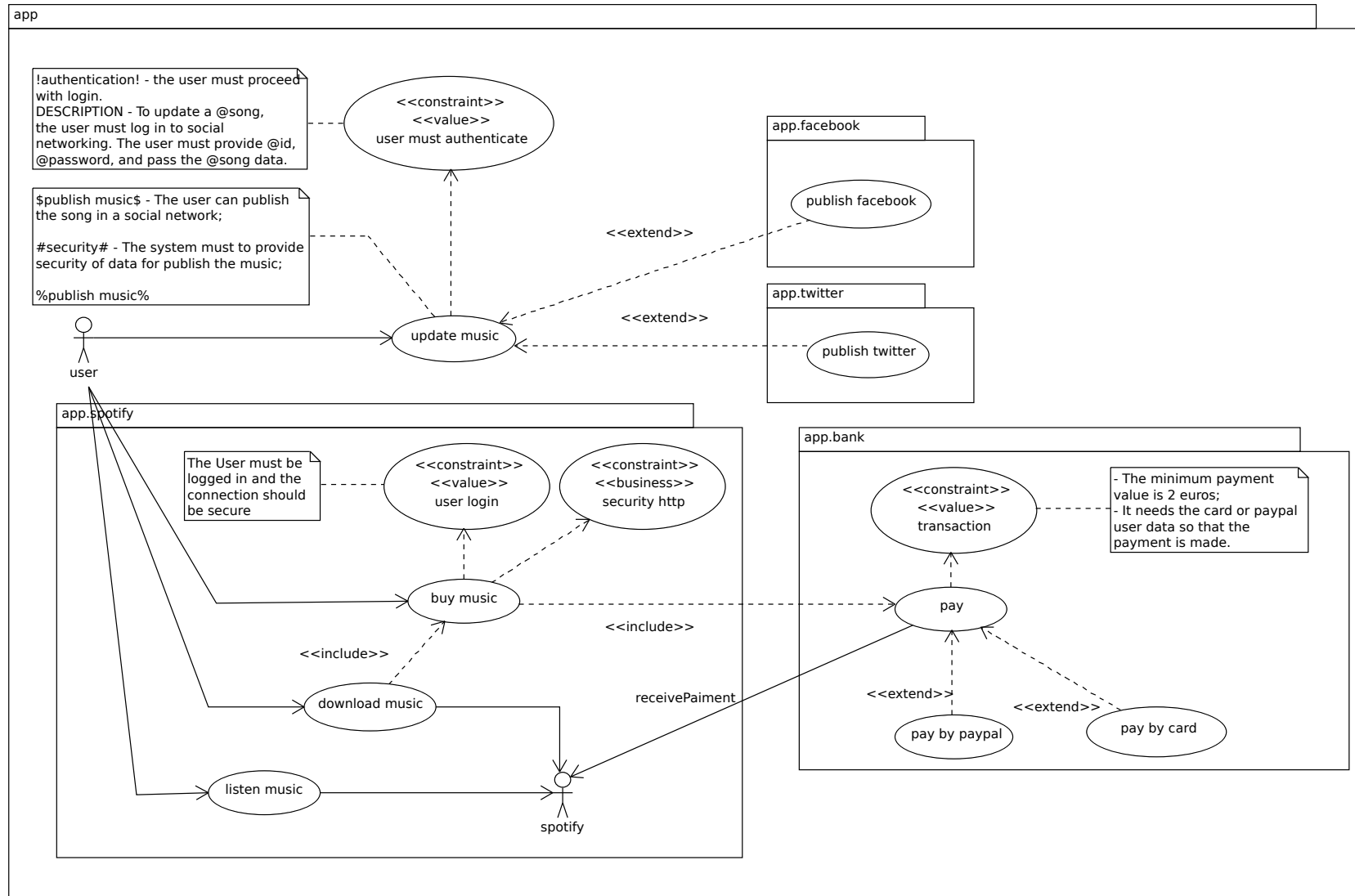


Figure 82: π -UseCase - To Publish Music.

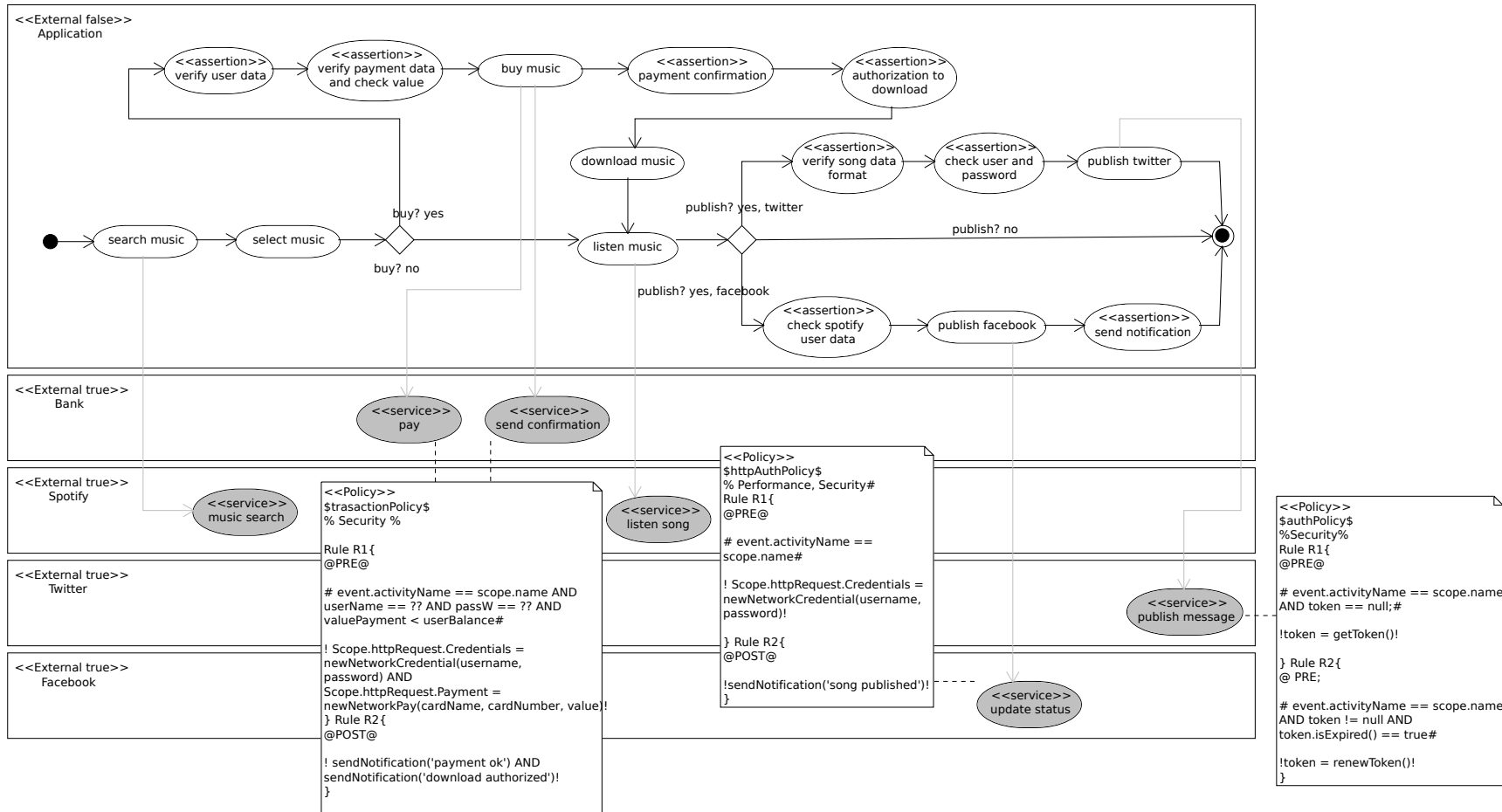


Figure 83: π -ServiceComposition - To Publish Music.