A proposal for the evolution of model-driven software engineering

Thiago Gottardi

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

Thiago Gottardi

A proposal for the evolution of model-driven software engineering

Doctoral dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Rosana Teresinha Vaccare Braga

USP – São Carlos
February 2018

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

```
       Gottardi, Thiago
G685p     A proposal for the evolution of model-driven
       software engineering / Thiago Gottardi; orientadora
       Rosana Teresinha Vaccare Braga. -- São Carlos, 2018.
          297 p.


          Tese (Doutorado - Programa de Pós-Graduação em
       Ciências de Computação e Matemática Computacional) --
        Instituto de Ciências Matemáticas e de Computação,
       Universidade de São Paulo, 2018.


          1. Engenharia de Software. 2. Processo de
       Software. 3. Engenharia de Software Dirigida por
       Modelos. 4. Engenharia de Software Experimental. 5.
       Paradigma de Programação. I. Braga, Rosana Teresinha
       Vaccare, orient. II. Título.
```

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Thiago Gottardi

Uma proposta para a evolução da engenharia de software dirigida por modelos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Prof. Dr. Rosana Teresinha Vaccare Braga

USP – São Carlos
Fevereiro de 2018

ACKNOWLEDGEMENTS

Dear Reader, thank you very much for reading this thesis. Hopefully, this thesis is going to be useful for your research and general interest. This required a lot of efforts because I intended it to be more than a usual thesis that only focuses on a specific research problem. I would be grateful if my efforts are useful outside the academia and also encourage future research projects.

Thanks to Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES – process numbers DS-8428398/D and BEX 3482-15-4) and Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP – process number 2016/05129-0) for grant received during the development of this work.

This is a list of all professors who have officially collaborated during my doctorate course. Thank you very much for your time! In chronological order:

- Prof. Dr. Rosana Teresinha Vaccare Braga (advisor, comittee member and teaching supervisor);
- Prof. Dr. Jon Whittle (internship supervisor).
- Prof. Dr. Daniel Lucrédio (comittee member);
- Prof. Dr. Paulo Cesar Masiero (comittee member and teaching supervisor);
- Prof. Dr. Toacy Cavalcante de Oliveira (comittee member);
- Prof. Dr. Elisa Yumi Nakagawa (teaching supervisor);
- Prof. Dr. Fabio Moreira Costa (comittee member);
- Prof. Dr. Renata Pontin de Mattos Fortes (comittee member).

Thanks to all professors, students and faculty workers at ICMC-USP; Thanks to all professors, students and faculty workers who accepted me at Lancaster University during my internship studies.

This is a list of collaborators who performed notable contribution to the development of the research that is reported in this thesis. Thank you very much for your time! In chronological order:

• MSc. Stevão Alves de Andrade	(Performance Studies);
• Prof. Dr. Rafael Serapilha Durelli	(Systematic Review);
• Dr. Silvana Morita Melo	(Process Discovery Study);
• Prof. Dr. Draylson Micael de Souza	(Empirical Studies and Case Study Requirements);
• MSc. Daniel G. Feloni	(Related Architecture for Reuse);
• MSc. Karen Rabelo Pacini	(Related Architecture for Reuse);
• MSc. Iohan Gonçalves Vargas	(Related Architecture for Systems of Systems);
• Prof. Dr. Kleinner Silva Farias de Oliveira	(Empirical Studies);
• BSc. Danillo Ferreira dos Reis	(Empirical Studies);
• Dr. Elias Adriano Nogueira da Silva	(Web Services Consulting);
• Prof. Dr. Júlio Cézar Estrella	(Web Services Consulting);
• MSc. Lilian Passos Scatalon	(Experimental Studies).

Thanks to all others who have helped, I apologize that I had to keep this list short. I am extremely grateful to all study participants, your contribution was fundamental to the completion of this thesis. Your identity will be kept anonymous as promised.

Many thanks to all attendants of the qualification exam and thesis defense. Thank you anonymous reviewers, your suggestions and criticism have helped significantly; including the detractors, who encouraged the most. We will never know if they expected this thesis to be completed. Thanks to the conference chairs and workshop participants who provided feedback on this work. Besides the original focus on Software Engineering, I am honored that researchers on Statistics and Distributed Systems are also interested on my efforts.

Upon starting my doctorate course I did not expect it could be risky. I am glad to have survived the huge storm on December $5^{t}h$, 2015 in Lancaster, UK. Many thanks to all friendly people who have helped when there was nowhere to live. It is noteworthy that despite this issue I still had to write a paper draft by hand beside emergency lighting, as there was no power. This draft eventually became the first paper that is based on the main proposal of this thesis.

In special, I am grateful to God and my parents that we have survived all the adversities that happened during this course, for many that could not make it. Goodbye to all my grandparents and also my aunt. Thanks for their encouragement. I am really sorry they could not see the completion of my doctorate, specially because it is the first one in our family. Farewell to my dog as well, despite working with so many bright people, I am still honored to have met such smart living being. My best wishes to every one, wherever you are.

In conclusion, thanks to everyone who has contributed directly or indirectly. Please rest assured that your help will not be forgotten. Thanks again to all readers. You are invited to read this thesis and additional documents[1].

[1] <http://tiny.cc/gottardi-doc>

RESUMO

GOTTARDI, T. **Uma proposta para a evolução da engenharia de software dirigida por modelos**. 2018. 297 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

No contexto da Engenharia de Software Dirigida por Modelos (MDSE), a produção de software pode ser realizada por meio de definições de modelos. Apesar dos benefícios desse método de desenvolvimento, diferentes domínios exigem a especificação de linguagens de modelagem e ferramentas específicas, que, por sua vez, precisam ser desenvolvidos em conjunto com o software final. Desta forma, desenvolvedores encontram problemas ao utilizar este método. Este trabalho possui duplo objetivo: 1) identificar os problemas mais críticos deste método; 2) discutir e fornecer possíveis soluções aos problemas. A identificação de problemas foi realizada por meio de um mapeamento sistemático, estudos empíricos, colaborações e entrevistas com especialistas. Foi identificado que MDSE, de acordo com a literatura básica, possui um nível de abstração excessivamente alto, acarretando em carência de processos adequados e de treinamento de desenvolvedores que vão além de problemas de necessidade de ferramentas de modelagem. Portanto, nesta tese, discute-se a necessidade de evoluir processos de MDSE que permita aos desenvolvedores uma nova forma de tratar modelos e código-fonte. Para tanto, neste trabalho também é descrito um novo método de desenvolvimento, descrito como uma possível evolução concreta do MDSE, o qual define um paradigma para desenvolver software. Este método é exemplificado em várias aplicações dentro deste trabalho. Após conduzir estudos analíticos e experimentais, concluiu-se que estas aplicações também possibilitam uma contribuição significativa no domínio de software orientado a serviços que podem ser empregadas em software do mundo real.

Palavras-chave: Engenharia de Software, Processo de Software, Engenharia de Software Dirigida por Modelos, Engenharia de Software Experimental, Paradigma de Programação.

ABSTRACT

GOTTARDI, T. **A proposal for the evolution of model-driven software engineering**. 2018. 297 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

In the Model-Driven Software Engineering (MDSE) context, software production can be performed by defining models. Despite the advantages of this methodology, different domains require specific support tools and modeling languages, which, in turn, must be developed along with the final software. Because of this, developers face problems when applying the method. The objective of this work is twofold: 1) to identify the most critical problems when developing with this method; 2) discuss and provide possible solutions to those problems. The critical problems were identified by performing a systematic mapping, empirical studies, collaborations and interviews with specialists. It has been identified that MDSE, according to basic literature, has an excessively high abstraction level which leads to a lack of adequate processes and developer training, besides the need for modeling tools. A new method is necessary to allow developers to treat models and source-code differently. Therefore, in this thesis, the need for evolving MDSE processes is discussed. In this manner, this work introduces a new development method described as a possible concrete evolution of MDSE that defines a paradigm for software development. This method is defined along with domain specific languages, a tool-chain and sample software systems. After conducting analytic and experimental studies, it has been concluded that these applications also represent a valuable contribution for implementing service-oriented systems which can be employed in real world applications.

Keywords: Software Engineering, Software Process, Model-Driven Software Engineering, Experimental Software Engineering, Programming Paradigm.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ATL	Atlas Transformation Language or ATL Transformation Language
BSc	Bachelor of Science
CDO	Connected Data Objects
CIM	Computation Independent Model
Dr	Doctor
DSL	Domain Specific Language
EMF	Eclipse Modeling Framework
EMP	Eclipse Modeling Project
ESS	Extraction-Selected Studies
GQM	Goal Question Metric
HTTP	Hyper-Text Transfer Protocol
MBSE	Model-Based Software Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MDSE	Model-Driven Software Engineering
MO	Model-Oriented or Model Orientation
MOD	Model-Oriented Development
MOF	Meta Object Facility
MOFM2T	MOF Model-to-Text
MOP	Model-Oriented Programming
MOS	Model-Oriented Software
MOSS	Model-Oriented Software System
MOWS	Model-Oriented Web Service
MOWSDL	Model-Oriented WSDL
MRT	Model(s) at Run-Time
MSc	Master of Science
MTL	Model-to-Text Language
MVC	Model-View-Controller
OO	Object-Oriented or Object Orientation
PDM	Platform Definition Language

PICO	Population, Intervention, Comparison, Outcomes
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query View Transformation
REST	REpresentative State Transfer
RQ	Research Question
SM	Systematic Mapping
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPEM	Software & Systems Process Engineering Metamodel
SR	Systematic Review
UML	Unified Modeling Language
WADL	Web Application Description Language
WS	Web Service
WSDL	Web Service Definition Language or Web Service Description Language
XMI	XML Meta-data Interchange
XML	eXtensive Mark-up Language
XSD	XML Schema Definition

CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Context

Software development methods are defined to improve software quality and/or decrease development effort. Among these methods, Model-Driven Software Engineering (MDSE) is a software development method in which models are not only employed for documentation or software representation, instead they can be employed to drive software development (BRAMBILLA; CABOT; WIMMER, 2012). Models are also used as input by transformation tools which are capable of generating code for the actual software. This allows to improve productivity and to increase understanding by representing the software at higher abstraction levels than source code, while also avoiding low level coding mistakes (FRANCE; RUMPE, 2007).

MDSE is a specific case of Model-Driven Engineering (MDE) applied to software engineering. In these development methods, models are employed within the development phases in order to produce the final product, which is a software in case of MDSE, but might be something else for MDE. As these models are not just used to describe designs and concepts, they are considered as the artifacts that effectively drive the product production (or software development) (BRAMBILLA; CABOT; WIMMER, 2012).

This use is justified within the literature (SCHMIDT, 2006; PASTOR; MOLINA, 2007), as modeling allows to represent problem concepts more effectively, while the transformations are capable of providing the solution for these problems. This requires to use modeling languages that allow the creation of machine readable models, which, in turn, can be executed or transformed into the final software. In this manner, it is possible to replace the source-code by models that may represent the software at higher abstraction levels, tightening the semantic gap between problem and its software solution (FRANCE; RUMPE, 2007).

Besides adequate modeling languages, development teams also require specific tools to edit, validate and transform models, which may be specific to the project or domain. Then, it is possible to categorize development teams that use MDSE into two groups. The first group is composed by developers working on specific domains that have suitable tools to use MDSE since the inception of their development projects. However, the second group works on domains with no previous tool definition, so they must either adopt general purpose tools or develop their own MDSE tools by employing existing methodologies, for instance, MDA (Model-Driven Architecture) (Object Management Group, 2010a). For this second group, it is necessary to create specific transformers for tailoring the adequate solution.

With the advancement and availability of tools and frameworks to create model transformation tools and modeling languages, it was expected that the effort required to produce new tools and languages would be justified thanks to the increased productivity (PASTOR; MOLINA, 2007). After several years following this expectation, it was not possible to confirm it. Throughout this thesis, there are studies on how this evolution could occur in reality.

1.2 Problem

In 2007, some authors in this topic (PASTOR; MOLINA, 2007) have described that MDSE's major problem is caused by the lack of adequate tools. From their perspective, once new adequate tools become available, the initial effort on adapting them for creating specific transformation tools and model editors would provide a large productivity gain that justifies this initial effort.

A few years later, Brambilla, Cabot and Wimmer (2012) have described the availability of a set of tools that can be employed during different activities involved in developing software with MDSE. These tools could indicate that the problems described by Pastor and Molina (2007) had been solved.

Afterwards, Whittle *et al.* (2013) have discussed that while the tools are part of the problem, there are other factors that cause projects to fail, which contradict the initial statement by Pastor and Molina (2007). Beyond lack of proper tools, these factors also include adequate training for developers, who are supposed to create their own tools when required by some MDSE projects.

The work documented in this thesis involves further studies on the challenges of this method that go beyond the tool problem and discussing other challenges of the method, which motivate the proposal of new treatments within this research.

1.3 Motivation

Pastor and Molina (2007) discussed that during forty years, software processes were created with the focus on how to produce quality source-code. Whenever models are present, they are treated as less important artifacts.

They also claim, however, that after advances on Model-Driven Development techniques and tools, process support and model edition have lagged behind. In this manner, available processes and their support might not be adequate to take advantage of the benefits empowered by MDSE.

In this thesis, the research is focused on the hypothesis that tool support is not the main problem that hinders developers from benefiting from the advantages of MDSE. Therefore, studies were conducted to further identify the problem, as well as proposing development methods that push beyond the established MDSE methods.

1.4 Justification

MDSE is a development method focused on raising representativity of documentation and increasing the development productivity (SCHMIDT, 2006; FRANCE; RUMPE, 2007). As previously written in this introduction, software development methods are defined to improve software quality and/or decrease development effort. This project also involves pointing how to extend the method to improve software quality instead of only increasing development productivity.

The objective of increasing the software quality is justified since it is perceived by the end-users, while the increase of development productivity might not be enough to encourage the use of the method. It is also discussed how techniques initially created for supporting MDSE could be made useful for the end-users.

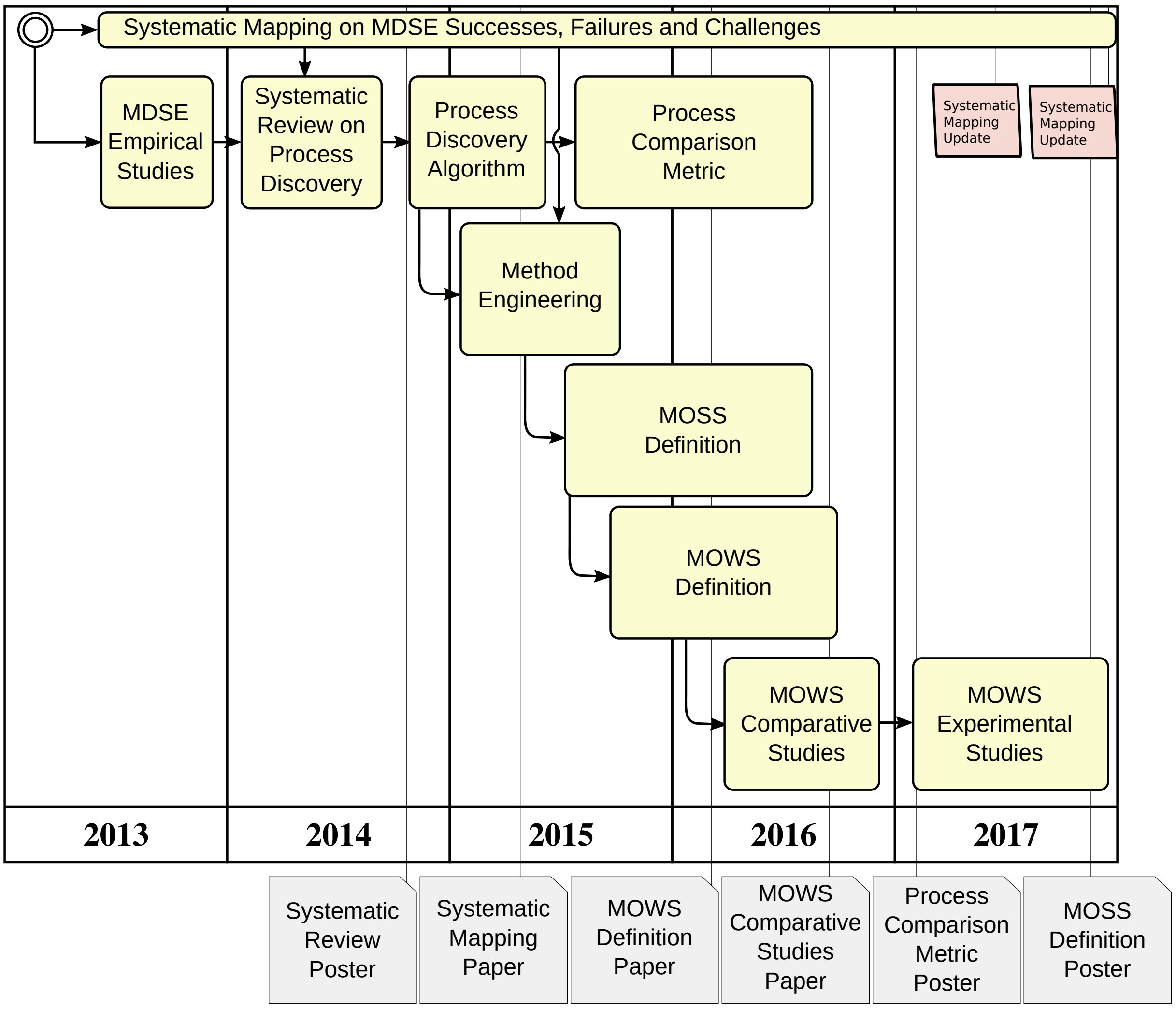
In this project, MDSE and derivative techniques are studied with the intent of discussing how the method could be evolved into a software paradigm in which the resulting products have visible quality improvement.

1.5 Methods

Different methods have been employed during the research efforts which led to the writing of this thesis. It is important to point out that these efforts are not linear, including different attempts throughout the project.

In Figure 1 there is a graphic representing the overall development of the research project reported in this thesis. This figure contains a time-line the horizontal axis include the years when the research has been conducted. Each research task is represented by rounded rectangles. Parallelograms represent the updates while the cards at bottom represent publications. The vertical axis has no importance in this figure, however, the rounded rectangles at top represent the starting studies: the Systematic Mapping and MDSE empirical studies are the starting studies that were used to advance the rest of research. All rounded rectangles are linked with arrows to indicate the research tasks that followed. This graphic does not include the future works.

Figure 1 – Research Methods Time-Line


Systematic Mapping on MDSE Successes, Failures and Challenges
MDSE Empirical Studies
Systematic Review on Process Discovery
Process Discovery Algorithm
Process Comparison Metric
Method Engineering
MOSS Definition
MOWS Definition
MOWS Comparative Studies
MOWS Experimental Studies
Systematic Mapping Update
Systematic Mapping Update
2013
2014
2015
2016
2017
Systematic Review Poster
Systematic Mapping Paper
MOWS Definition Paper
MOWS Comparative Studies Paper
Process Comparison Metric Poster
MOSS Definition Poster

Source: Created by the author

This research project was composed by comprehensive literature reviews, including a systematic mapping and a systematic review. The systematic mapping was performed to identify success domains of MDSE and existing challenges by reviewing 4859 studies collected from search engines, in which 3727 are unique. According to this mapping, MDSE has been successfully applied to several application domains, including Business Information Systems, Web Applications, and Computer Networks. For domains that are not covered by successful MDSE tools, there are some challenges that are left unsolved, as for example: need for adequate software development processes, tool dependency and need for support, and maintenance issues when developers need to modify the generated code or source models. The challenges are further discussed as part of Chapter 3.

After the initial search sessions of the systematic mapping, studies on these challenges were conducted with the intent of further learning how these challenges could be solved. Initially, students who attended a class on MDSE were interviewed (informally). Preliminary conclusions encouraged work on empirical research, which included two studies to assess the difficulties that developers would face when using MDSE. Another systematic review was conducted, this time in the context of process discovery techniques. A process discovery algorithm was formalized and extended as part of this research project. The process discovery technique was used to analyze small software projects using MDSE in which students worked on.

The analysis of these projects encouraged exploratory research efforts with the goal of establishing a definition of a software category which is easier to build by using MDSE since project inception. This software category along with development techniques and tools are provided as the main contribution of this thesis.

This software category was validated in case studies, comparative studies and experimental studies, which are presented within this thesis, indicating their feasibility and applicability for software development.

We also attempted to study challenges related to tools for MDSE. One hypothesis was related to the lack of proper usability by modelling tools employed in MDSE processes. In this manner, usability studies were also conducted, but these were cancelled during the research project since their hypotheses were treated by related works selected during the SM execution.

Further details on the completed and cancelled studies that were carried out during this project are presented in Chapter 4.

1.6 Objective of this Thesis

The objective of this document is to present the outcome of a research effort which involves secondary studies related to primary studies conducted in the context of MDSE. Problems were identified and possible solutions are also presented and validated.

Among these solutions, an evolution of the MDSE method is discussed in conjunction with more recent developments. Despite arguments that the lack of tool support is not the major problem in this context, further tool support was also devised for this new proposal, with the intent of assisting developers to apply this paradigm.

Therefore, there are primary and secondary objectives to be listed for the thesis, described herein: The primary objective is to describe a possible, feasible and validated software category with its development method that is evolved from the legacy of MDSE techniques, tools and methods.

This software category was created in a manner to avoid tool dependency and make models useful for the end-users. Therefore, MDSE tools are optional and yet useful for the developers and end-users. This category is also more specific allowing to specify a simple but concrete development method.

Among secondary objectives, there are:

1. Document employed research methods;
2. Document MDSE development studies;
3. Document software project method analysis;
4. Document and provide developed languages and tools to support the proposal;
5. Document a systematic mapping;
6. Document a systematic review;
7. Document experimental studies;
8. Document case studies;
9. Document process discovery and comparison algorithm.

In order to attain the primary objective, several research efforts were conducted, which are described in this thesis for future documentation. Therefore, the following chapters include descriptions on the conducted studies.

1.7 Thesis Outline

This thesis is divided into eight chapters. Besides the introduction, the organization of this document is structured as follows: The theoretical foundation required for proper understanding of the basics required for the establishment of this thesis are presented in Chapter 2. In Chapter 3, a comprehensive systematic mapping on model-driven software engineering successful domains and challenges is presented. Chapter 4 includes the history of employing process analysis and method synthesis which were used for the proposal presented within this thesis, which is described within Chapter 5 as a software system category. This proposal was validated in a set of studies. Chapter 6 contains comparative studies, which compare the proposed software systems with traditional systems. Chapter 7 contains experimental studies, which evaluate development efforts to build the proposed software systems. Finally, Chapter 8 contains the main thesis conclusions, outlooks for future works as well as list of related publications. This thesis also includes a set of appendix chapters for further reference.

CHAPTER

2

THEORETICAL FOUNDATION

2.1 Initial Remarks

This chapter is composed by theoretical foundation necessary for understanding this thesis. Section 2.2 contains the principles related to MDSE, including modeling and meta-modeling languages, tool examples and processes. Section 2.3 introduces method engineering, which allows to create new methods based on existing method fragments. Section 2.4 consists of an introduction on process discovery techniques, presenting approaches and algorithms to analyze and generate documentation for previously executed processes. Section 2.5 includes basic definitions on Markov models, a statistical method that has been used in process discovery techniques. Section 2.6 contains principles of Models at Run-time. An existing programming paradigm evolved from MDSE is described in Section 2.7. Section 2.8 contains definitions for web services. Finally, in Section 2.9, this chapter is concluded.

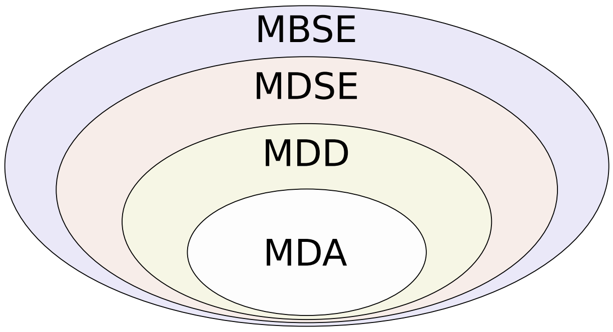
2.2 Model-Driven Software Engineering

Models are artifacts that can be used to represent problems, real world concepts and/or solutions at high abstraction levels, while a software system source-code is written to express the implementation of a solution. Models can be also employed within software engineering during development efforts, in a method named as Model-Driven Software Engineering (MDSE) (BRAMBILLA; CABOT; WIMMER, 2012). MDSE is made possible by employing generators or interpreters, which take models as input artifacts to generate or execute the final software (PASTOR; MOLINA, 2007).

MDSE is related to other methods that employ models. According to the diagram shown in Figure 2, there are four main methods that employ models: MBSE (Model-Based Software Engineering), MDSE, MDD (Model-Driven Development) and MDA (Model-Driven Architecture). MBSE is the broader case of any software process that employs models as artifacts during development, regardless of the importance of these models. Therefore, MBSE involves both MDSE and MDD.

MDD is the specific case of MDSE in which models are employed to generate the final software, therefore, in the figure, MDSE involves MDD completely (BRAMBILLA; CABOT; WIMMER, 2012). MDA, in turn, is a specific case of MDD in which models have specific abstraction levels. It was defined as a standard by the Object Management Group (Object Management Group, 2010a), with the objective of establishing a structured approach for MDD.

Figure 2 – Sets of Methods that Employ Models



Source: Brambilla, Cabot and Wimmer (2012)

MDD is the combination of generative programming, domain specific languages and software transformation. Its objective is to reduce the gap between problem and implementation/solution by using abstraction levels higher than source-code, protecting developers from the complexity of the implementation platform (FRANCE; RUMPE, 2007).

This objective is attained when the models are employed to express concepts from the problem domain more effectively than source-code. These models can be instances of Domain Specific Languages (DSL), which are modeling languages created specifically to express concepts for the given domain. Specific tools are also used to create the software solution for the problems declared in the models (SCHMIDT, 2006; PASTOR; MOLINA, 2007).

A successful MDSE process requires specific tools to transform the related models into the solution. These transformations are categorized according to the type of input and output artifacts.

When a text artifact is generated from a model, this transformation is referred as “Model-to-Text”. In the same fashion, models can also be generated from other models, in a transformation named as “Model-to-Model”. It is also possible to have other combinations of transformations, including “Text-to-Model” and “Text-to-Text”.

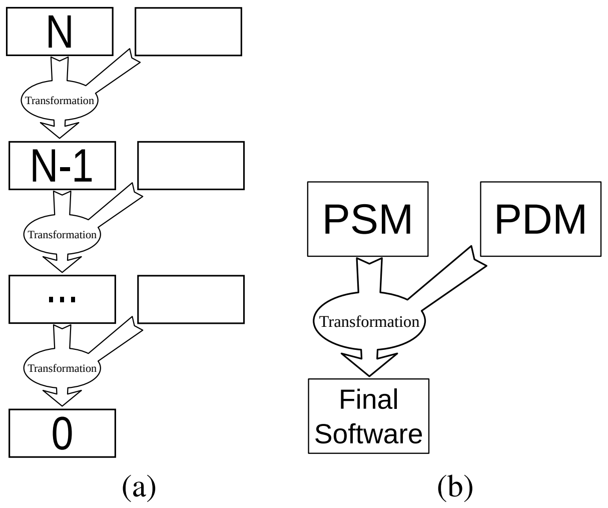
The generated artifact does not need to be the final software. These can also be intermediate artifacts which are sequentially transformed until reaching the intended abstraction level. This technique is used within MDA. According to the MDA specification, models are categorized into three levels of abstraction: Computation-Independent Model (CIM), Platform-Independent Model (PIM) and Platform-Specific Model (PSM).

CIM is the most abstract model, which has no assumption related to computation. In this manner, it must only represent the problem without defining its solution. PIM is an intermediate model that is platform independent. While it may represent how to implement the solution, it must not be tied to a specific implementation platform. The lowest level is named as PSM and is specific to a single implementation platform.

MDA is based upon the principle that transformations may also generate intermediate artifacts. These transformations are then employed sequentially in order to gradually lower the abstraction level, until the lowest level (final software source-code) is attained.

As illustrated in Figure 3 (a), a development process employing MDA could involve several levels of modeling artifacts. The first (most abstract) level is referred as level *N* and is sequentially transformed into lower level models until level zero is attained. The blank boxes in Figure 3 (a) were added to suggest that the transformations may also merge more models during their execution.

Figure 3 – Illustration of Different Modeling Levels



Source: Object Management Group (2010a)

There is also another type of model in MDA named as Platform-Definition Model (PDM), which is intended to map PSM models to the final source-code (Object Management Group, 2010a). This transformation is illustrated in Figure 3 (b).

As represented in Figure 3 (a), model transformation in MDA involves multiple divisions. These divisions improve the reuse opportunities of each model level. For instance, in order to generate different systems that share the same domain or to migrate the target platform, it is only necessary to replace a few models at intermediary levels.

2.2.1 Metamodels, Meta-metamodels and Metamodeling

Metamodels are models that define a modeling language either formally or semi-formally. A metamodel is composed by “metaelements” or “metaobjects”. The modeling language of a metamodel is defined as a model as well, which is referred as “meta-metamodel”.

In order to explain the relationships among models, meta models and “meta-metamodels”, let A, B and C be all models for developing with MDSE. A represents the most abstract modeling language and defines its own language; A is also used to define the language of B. In the same manner, B is used to define the language of C (BRAMBILLA; CABOT; WIMMER, 2012).

Therefore, A is a “meta-metamodel”, B is a “metamodel” and C is simply a model. Being in a meta level does not exclude the fact that they are all models. This is because metamodels are also models. Meta-metamodels are also metamodels, which in turn are also models.

ECore (Eclipse Foundation, 2011) and Meta Object Facility (MOF) (Object Management Group, 2010b) are examples of meta-metamodels. MOF, for instance is used to define the Unified Modeling Language (UML), a general purpose modeling language for object-oriented software including analysis, design and documentation (BOOCH; RUMBAUGH; JACOBSON, 2005). Similarly to UML itself, MOF also includes classes, relationships and properties, though at the meta-level.

For example, a portion of the metamodel of UML 2 is represented in Figure 4 as a MOF metaclass diagram. This diagram includes “Class”, “Property” and “Operation” metaclasses. These metaclasses define the UML element types Class, Property and Operation (Object Management Group, 2010b).

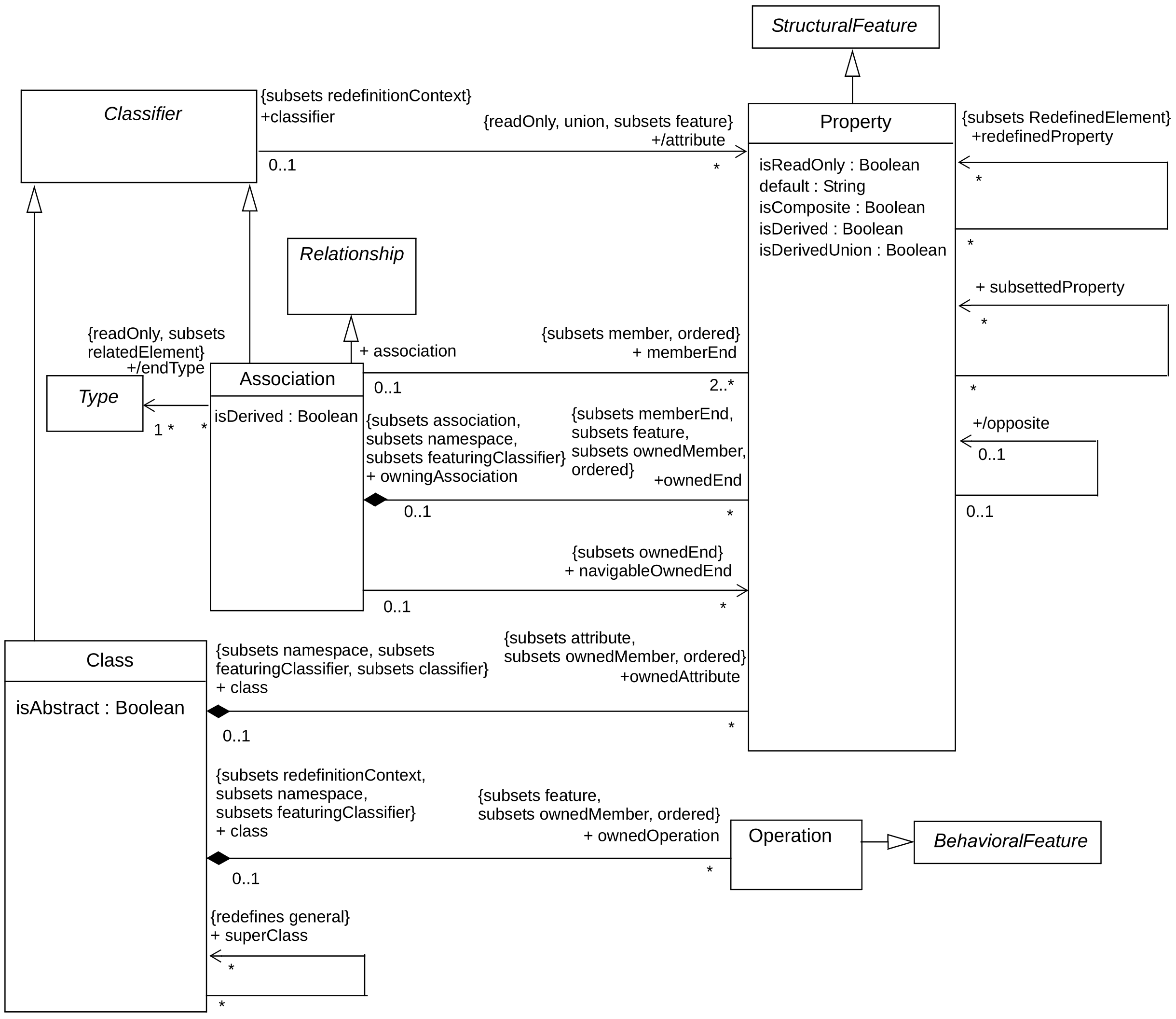
It is important to point out that MOF also includes metaclasses associations and properties. For instance, it is visible in the figure that “Class” is associated with “Property”. This relationship indicates that classes may contain several attributes, while each attribute must be owned by at most one class (as defined by the multiplicity numbers “0..1”).

Another important detail to point out is that MOF also includes generalizations. In Figure 4 it is possible to visualize that “Class” has a generalization relationship towards “Classifier”, which indicates that “Class” inherits every meta-level property.

ECore is another meta-metamodeling language (Eclipse Foundation, 2011). As well as MOF, ECore is used to specify modeling languages by using metaelements. These metaelements include “EClass” for metaclasses, “EAnnotation” for metamodel annotations, “EAttribute” for metaclass properties, “EEnum” for enumerations and “EPackage” for packages.

The advantage of employing ECore instead of MOF is that ECore is simpler. While having less meta-metaelements indicates that ECore has less features than MOF, a complex meta-metamodeling has more features that may cause ambiguity in the modeling languages (BRAMBILLA; CABOT; WIMMER, 2012).

Figure 4 – MOF Metaclass Diagram with a portion of UML Specification


StructuralFeature
Classifier
{subsets redefinitionContext}
+classifier
{readOnly, union, subsets feature}
+/attribute
Property
isReadOnly : Boolean
default : String
isComposite : Boolean
isDerived : Boolean
isDerivedUnion : Boolean
{subsets RedefinedElement}
+redefinedProperty
+ subsettedProperty
+/opposite
Relationship
{readOnly, subsets relatedElement}
+/endType
Type
Association
isDerived : Boolean
+ association
{subsets member, ordered}
+ memberEnd
{subsets association, subsets namespace, subsets featuringClassifier}
+ owningAssociation
{subsets memberEnd, subsets feature, subsets ownedMember, ordered}
+ownedEnd
{subsets ownedEnd}
+ navigableOwnedEnd
Class
isAbstract : Boolean
{subsets namespace, subsets featuringClassifier, subsets classifier}
+ class
{subsets attribute, subsets ownedMember, ordered}
+ownedAttribute
{subsets redefinitionContext, subsets namespace, subsets featuringClassifier}
+ class
{subsets feature, subsets ownedMember, ordered}
+ ownedOperation
Operation
BehavioralFeature
{redefines general}
+ superClass

Source: Object Management Group (2010b)

2.2.2 MDSE Technical Requirements

Pastor and Molina (2007) have discussed how software development projects using MDSE depend on adequate tools. More recently, Brambilla, Cabot and Wimmer (2012) have described that several tools were made available. For example, The Eclipse Modeling Project (Eclipse Foundation, 2011) encompasses a set of tools to support developers working with models. It also involves languages and frameworks to empower the development of modeling tools and transformers.

Among instances of languages to support model-to-model transformation, there is the Atlas Transformation Language (ATL), and the Query View Transformation (QVT). Among languages to support model-to-text transformation, there is MOF Model to Text Language (MTL), Java Emitter Template (JET) and XPand, all of which have tool implementations available (Eclipse Foundation, 2011).

Another technical requirement is related to configuration management involving modeling artifacts. For this concern, repository managers were developed, e.g. Connected Data Objects (CDO) (Eclipse Foundation, 2013a) and Morsa (ESPINAZO-PAGáN; CUADRADO; GARCíA-MOLINA, 2011). Model-specific comparison tools have also been developed, e.g. EMFCompare (Eclipse Foundation, 2011) and SiDiff (KEHRER *et al.*, 2012).

Depending on the relationship among artifacts during the development process, different models may be affected by changes in parallel. These parallel changes might need to be synchronized, a concern that requires adequate tools. Among examples of tools to synchronize these parallel changes, Eadapt (Eclipse Foundation, 2013b) allows the developers to define rules to automatize the migration of models after modifying their referred ECore metamodels. This project is part of the Eclipse Project (Eclipse Foundation, 2011).

2.2.3 Model Management

An MDSE process employs models as the main artifact throughout the development (PASTOR; MOLINA, 2007). Models may also refer to related models and, thus, they can present dependencies. This indicates the need for documenting these relationships, describing how each model affects other artifacts. The management of these dependencies would support project managers and developers to acknowledge how a model modification could impact other artifacts.

Among proposals to support model management, "megamodels" were created. "Megamodel" is a category of domain specific language intended to document how the models participate during the life-cycle of model-driven software development (BRAMBILLA; CABOT; WIMMER, 2012). This documentation may also be flexible enough to support other model based software processes that are not model-driven.

For example, the following megamodels were made available: AM3, a metamodel for graphical megamodels; MoScript, a toolset which includes a domain specific language for megamodels. These metamodels support the creation of model instances that can be employed to record the relationship of other modeling artifacts within a MDSE transformation chain. Both instances are part of the MoDisco (Eclipse Foundation, 2013c). It is important to advise that despite the fact that megamodels were defined for creating documentation for the software process and the participation of the artifacts, they are not development methods and do not include guidelines for developers.

2.2.4 MDSE Specific Software Processes

Development methods include guidelines and instructions for developers on how to act during the development of a solution (or product). Among these methods, there are software processes. "Software Processes" or "Software Development Processes" contain a sequence of activities and their related expected results for producing a software product (SOMMERVILLE, 2015).

It is important to remind that since MDSE software processes involve models as active artifacts besides the source-code, there are different possible sequences on which artifact should be defined first; for simplicity, these processes are referred to in this thesis as either **model-first** or **code-first**. According to MDSE and code generation literature, software processes are traditionally focused on writing code (PASTOR; MOLINA, 2007). There are code generation techniques that were devised focusing on reverse-engineering previously existing code into models (HERRINGTON, 2003; DURELLI *et al.*, 2014; HESS, 2005), i.e., code-first methods. These are development methods in which code is written before model transformation is performed. However these methods do not constitute a process that involves the complete life cycle of a software project.

On the other hand, model-first processes start with the definition of models before code, since these processes cannot be based upon traditional code writing techniques. The remainder of this section is focused on model-first approaches. Therefore, in this section, we identify processes defined specifically for MDSE, which are: Almeida *et al.* (2003), Belaunde *et al.* (2004), Chitforoush, Yazdandoost and Ramsin (2007) and Asadi, Ravakhah and Ramsin (2008). All these processes are initiated from model definition, before any source-code artifact is produced. They employ higher to lower abstraction level evolution as recommended by MDA. Although MDA includes guidelines on how to involve models within a software development project (Object Management Group, 2010a), it constitutes a method but does not encompass a process, since it lacks a software development activity sequence.

The oldest MDSE-specific method specification that has been identified during a systematic literature mapping performed during this project (further described in Chapter 3) is the process created by Almeida *et al.* (2003). These authors have not defined this method as a complete process for software development, it is simply a set of guidelines to support developers while defining models at the PIM and PSM abstraction levels.

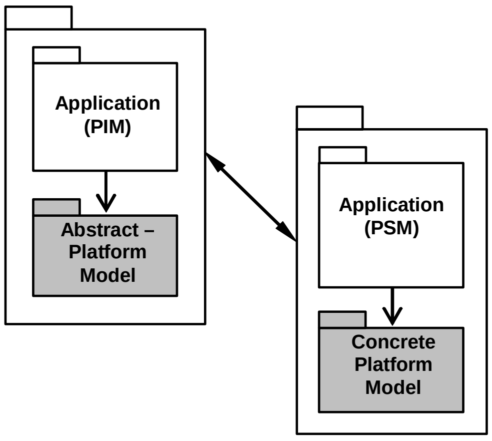
In Figure 5 there is a simplified diagram for illustrating the method created by Almeida *et al.* (2003). In their proposal, there are relationships to indicate which features of PIM models must comply with the related PSM models. Their method also supports activities on how to define models. However, there are no specific instructions on how to create metamodels.

In a posterior project in which Almeida was also involved, a complete software process for MDSE has been specified. This process was named as “MODA-TEL”, after the consortium with the same name[1]. This consortium was managed by the EURESCOM[2].

The process is illustrated as a diagram in Figure 6. This diagram includes the process phases, represented by rectangles. Arrows represent the sequence flows and dependencies

[1] http://www.modatel.org/
[2] http://www.eurescom.eu/

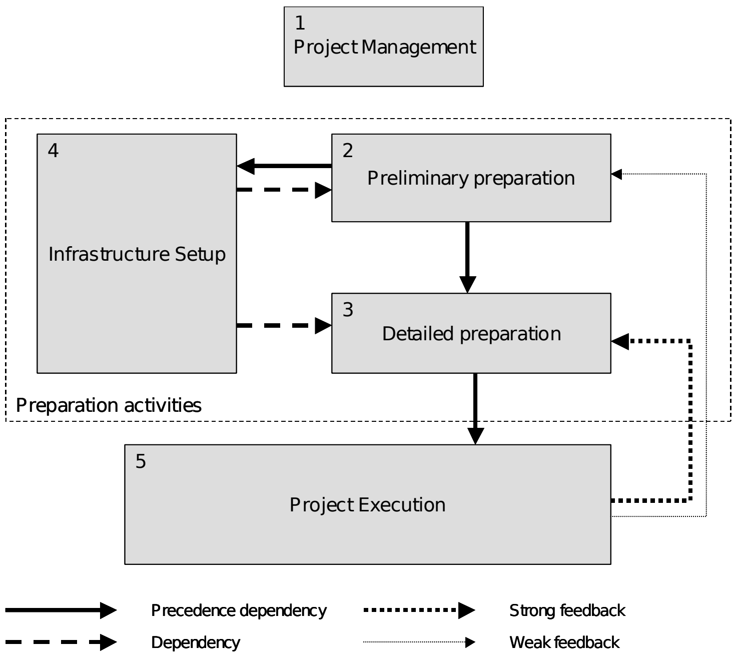
Figure 5 – The Method by Almeida *et al.* (2003)



Source: Almeida *et al.* (2003)

between a pair of phases. Sequence flows may be blocking, indicating that the following phase must be interrupted to allow the previous one to be completed. This implies that if any following activities were being executed, they must be interrupted in order to return to the preceding activity. Sequence flows may also be auxiliary, indicating that the related phase may cooperate on the same product without interrupting.

Figure 6 – MODA-TEL Process Diagram



Source: Belaunde *et al.* (2004)

During the "Project Management" phase, the activities are related to project planning from a managerial perspective. The MDSE activities are inserted into two specific phases, referred as “Preparation Activities”. “Preliminary Preparation” is the name of the first preparation phase, which includes activities related to the requirement analysis of modeling and transformations. Following this specification, during the second phase, named as “Detailed Preparation”, the actual model formats and transformations are defined in detail.

The resulting specifications created during both “Preparation Activities” are used during the following phase, referred as “Infrastructure Setup”. Its activities include decisions regarding the adoption of modeling and transformation tools, as well as the definition of how to manage the models throughout the process.

During the “Project Execution” phase, the goal is to develop the final software artifacts, which includes creating all input models and source-code, testing, validation, transformer execution, integration, transition and maintenance.

Despite the availability of these phases, it is important to remind that the MODA-TEL process was created for a specific software domain (telecommunications) and does not include specific guidelines for the model/metamodel specification activities. It does not employ CIM models and has no guidelines for creating model documentation.

In another proposal, Chitforoush, Yazdandoost and Ramsin (2007) have considered the limitations of MODA-TEL and defined a framework for MDSE development. This framework was further expanded into an actual software development process by Asadi, Ravakhah and Ramsin (2008), which is the most complete MDSE process identified during literature reviews performed within this project. Their process is based upon MDA, including activities for defining CIM models and documentation.

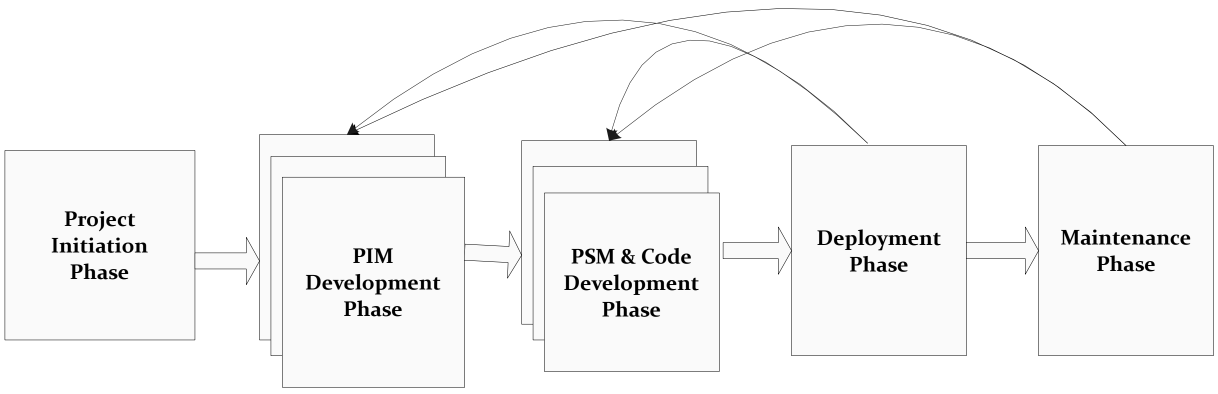
By following the MDA, their process is composed by activities related to the definition of CIM, PIM and PSM models. Figure 7 contains a diagram to illustrate the process created by Asadi, Ravakhah and Ramsin (2008), which is divided into five phases. The first phase is related to project initiation (or inception). During this phase, the activities include definition or adoption of a CIM model, definition of the requirements model, establishing the development team and selecting the support tools, e.g., editors and transformers.

The second phase involves PIM model specification. During this phase, CIM models and requirements documents are used as base to create a conceptual PIM model, an architectural model and, subsequently, a detailed PIM model. The detailed PIM model must be verified before it is used as input for model transformers.

The third phase involves PSM model development along with source-code. The first model transformation is executed during this phase by transforming the detailed PIM model into PSM. There are no specific instructions on how to create the transformers. The authors simply mention that this is dependent on the tools chosen during the first phase. Following this activity, PSM is transformed into source-code. After the source-code is available, the software undergoes testing activities.

The fourth phase involves deployment, while the fifth phase is the maintenance. Both phases are related to the software transition from the developer to the client. Further documentation writing is also included during the maintenance phase.

Figure 7 – Process Diagram by Asadi, Ravakhah and Ramsin (2008)



Source: Asadi, Ravakhah and Ramsin (2008)

It is important to inform that every process identified during the literature review performed within this research project starts from the highest abstraction level towards the lowest. In the same manner, Asadi, Ravakhah and Ramsin (2008) process starts from the CIM model and evolves sequentially towards PIM, PSM and source-code (model-first process). Loops allow developers to revisit past activities in iterations.

As a disadvantage, this process includes strict assumptions on the process model and application domain, which discourages it from being adapted into process models in the state of practice. Another disadvantage is that the process lacks specific guidelines for how to define the models, metamodels and support tools whenever they are not previously available.

This lack of detail has encouraged the author of this thesis to conduct further research on how to study existing processes to encourage discovery of successful process executions that were conducted without a documented process. The main goal of studying existing processes was to pinpoint why each activity was required in a MDSE process. These activities eventually led to the creation of a software category described as a possible evolution of MDSE.

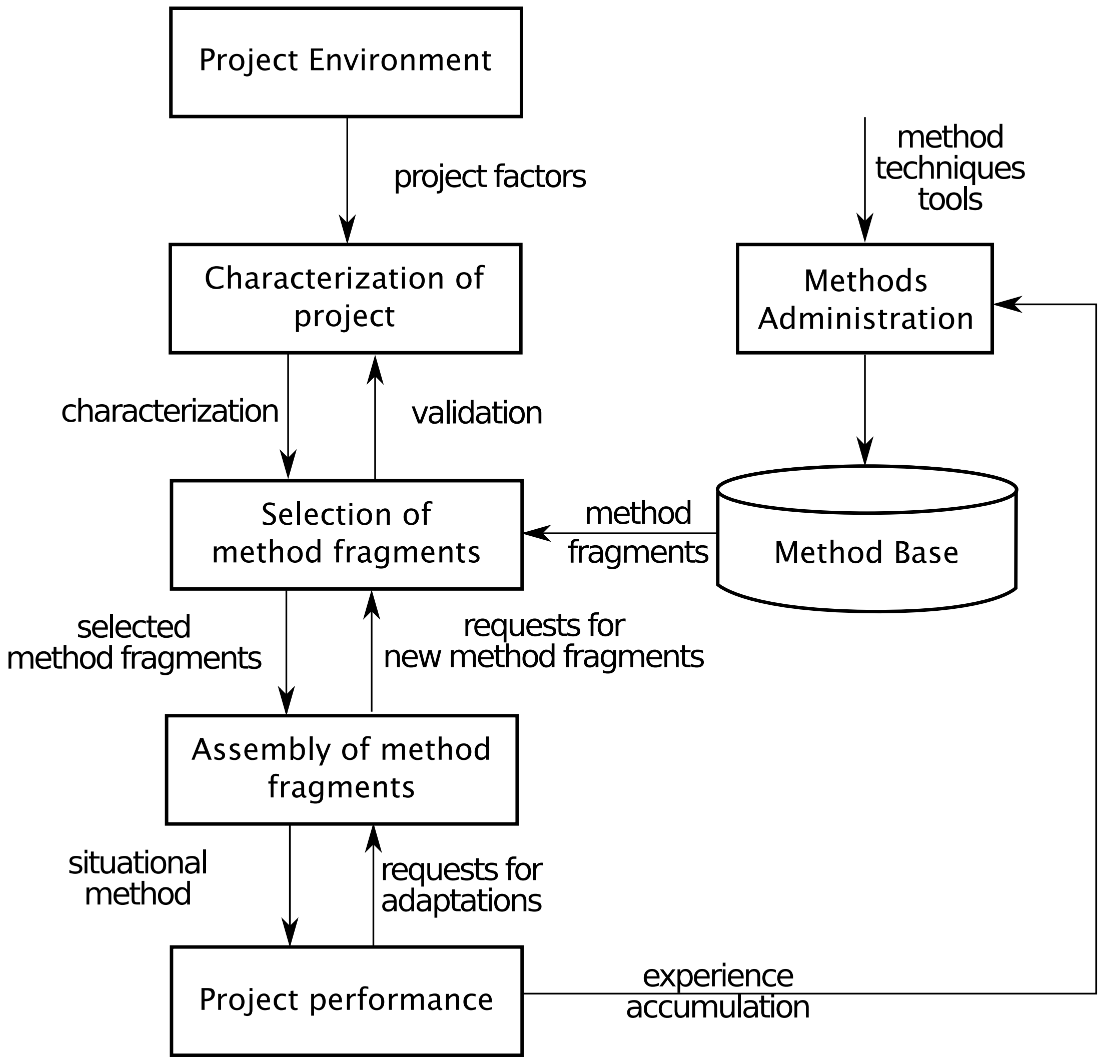
2.3 Method Engineering

Method engineering was defined by Brinkkemper (1996), based on the previously published works by (KUMAR; WELKE, 1992). In this thesis, method engineering is employed to construct a new method for MDSE software projects which are developed by using model-driven techniques since its inception.

> “Method engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems” (BRINKKEMPER, 1996, p. 2).

In this thesis, with the objective of increasing abstraction levels, meta-modeling is applied both in the context of method design and in the software model meta-level specification. A method definition is composed by method fragments, which are selected to form the final method. These fragments represent activities to be performed within the defined method. In order to properly define fragments and their sequences, this chapter also includes meta-level formats and formalization for specifying the fragments of the method proposed within this thesis.

Figure 8 – Method Fragment Management



Source: Olle and Verrijn-Stuart (1994)

The method engineering guidelines shown in Figure 8 are composed by six activities. The first activity (“Project Environment”) involves establishing which project is the target for the method to be defined. The second activity (“Characterization of Project”) involves the characterization of the chosen project in order to gather further details.

The third activity is represented in Figure 8 as the “Selection of Method Fragments” (BRINKKEMPER, 1996; OLLE; VERRIJN-STUART, 1994). The fragments are stored into a repository of fragments (“Method Base”). The fragments are selected to compose a new method adequate for a certain project.

The forth activity (“Assembly of Method Fragments”) involves assembling the selected method fragments into a proper method. This activity is important to define the order in which these fragments should be performed within the actual project. After the project is performed, its outcome should be measured to provide feedback to the used fragments. This measurement is part of the “Project Performance”. Methods and the results of their executions are inserted into the method base to foster reuse on other method engineering activities.

This flexibility of method definition is also considered a meta-level approach. Brinkkemper (1996) has discussed the need to provide meta-modeling techniques in order to specify methods adapted for software development projects. Therefore, they have identified the need for adequate meta-modeling in the context of method design.

2.4 Software Process Discovery

Development processes may become successful without a previously specified software process. This does not mean that there is no software process, however, this indicates that the executed process was not documented. Process mining techniques were created to support (and automate) the discovery of existing processes, fostering the analysis and improvement of the executed process (AALST, 2012).

With the intent of discovering the executed process, Jensen and Scacchi (2004) have executed semi-automated analyses on artifact repository logs from software projects. These logs allow to identify information on the impact of each change, including date and authors. After identifying the artifact records and their log records, it is possible to identify the roles of each author and the sequences of their work, which are mapped towards a software process.

Porter and Selby (1991) have devised an approach for empirical software process discovery based on the data analysis of the process execution by using software metrics. This was implemented on a tool developed to capture data employing software and software process metrics, which allow to capture quantitative data on the process execution, allowing to measure the work performed by developers during the phases, iterations and tasks. According to how the metrics fluctuate throughout the process execution, it is possible to identify positive and negative steps of the process during its execution, reporting to the project manager when to take action whenever risks to the development tasks arise.

Collecting data on processes can also support documentation on the execution, allowing to reproduce the success cases in similar projects. These metrics are further documented in another work (SELBY *et al.*, 1991a).

In more recent works, another set of metrics was defined with similar goals (TAKAHASHI; MURAOKA; NAKAMURA, 1997; COOK; WOLF, 1994). Punitha and Chitra (2013) have presented a survey on metric types with the intent of comparing their precision for detecting software defects. In this thesis, the need of a software process comparison metric has been identified during the research to identify adequate software processes. This metric is further described in Chapter 4.

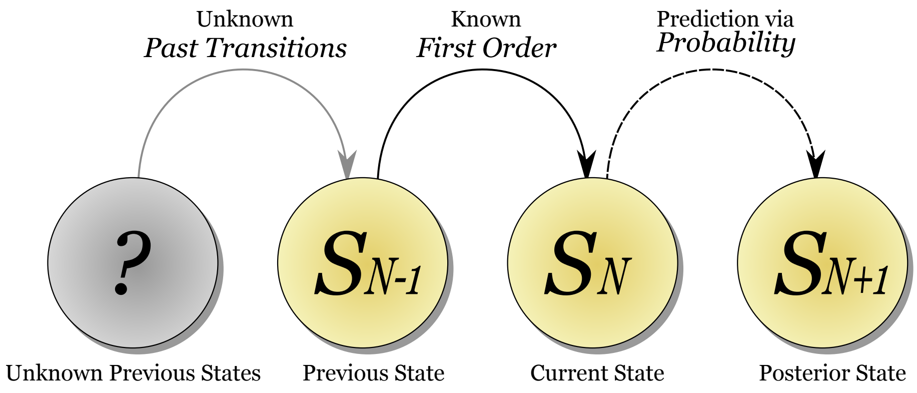
2.5 Markov Models

In this thesis, a software process comparison metric was proposed based on the software process model discovery algorithm described by Cook and Wolf (1998). Similarly to their algorithm, the proposed metric employs Markov models in its definition, more specifically, discrete space Markov process models, also called as Markov chains.

Markov process models are a specific kind of stochastic process models that follow the *Markovian* Property. Stochastic process models are used to represent probabilistic or random behavior while the Markov Property is described as a memory-less property (KOSKI; NOBLE, 2011).

Formally, stochastic processes are defined as a set of states and transitions. The transitions are taken depending on a parameter from the sequence of random variables (KOSKI; NOBLE, 2011; FINK, 2014). These random variables assume values that can be calculated as a probability.

Figure 9 – Finite Automaton Example



Source: Created by the author

For example, considering the finite automaton represented in Figure 9, in order to calculate the first order probability of transition between states S_N and its posterior S_{N+1}, it is only required to know the previous state S_{N-1}. However, the states that were active before S_{N-1}

may be unknown, hence the memory-less property. The states that must be known are referred here as “within the scope” of calculation.

The Markov property is important when calculating the probabilities matrices, which are defined by counting the actual execution of an event stream from a previous state to a current state, while the execution of states that were active before this scope are not necessary. This scope is also important to provide flexibility: the usage of a extremely broad scope would cause the discovered process model to be a single sequence of activities that copies the execution stream, hampering the definition of loops or branches.

2.6 Models at Run-Time

In this thesis, a development method is proposed and evaluated. The resulting software systems based on this method also include the handling of models at run-time. The key differences of the proposed method when compared to Models at Run-Time is due to how the models are used to represent the problem. Still, it is important to provide a brief introduction about Models at Run-time (MRT) approaches, as they represent relevant work related to our proposal.

Similarly to the introduction to Model-Driven Software Engineering (MDSE), the context of this section is related to issues caused by developing software at low abstraction levels. Source-code at low abstraction levels does not include information from the problem space for which the software was designed, regardless if the source-code was manually programmed or generated from automation (FRANCE; RUMPE, 2007).

In MDSE processes, model transformation is employed to generate source-code. In the same manner, the generated code is created to cope with the solution for a problem. As the models may contain concepts from the problem, information may be lost after the generation. If the execution of the resulting system depends on information regarding the problem, this issue leads to undecidability, i.e., the algorithm in execution could be indefinitely executing in an attempt to find an information that will not be available (ASSMANN *et al.*, 2014).

Models at Run-Time (also written as *Models@Run.Time* and abbreviated as *M@RT*) were defined as a plausible solution to cope with limitations related to undecidability and uncertainty (GIESE *et al.*, 2014).

The key property of a model at run-time system is its capability of reflection empowered by the usage of the software models during run-time, instead of only the resulting source-code (ASSMANN *et al.*, 2014). Therefore, these systems must have special mechanisms to connect the problem concepts to their solutions, causing the system to be aware of the problem space while executing the solution. This allows the system to capture information from the models (e.g., problem space) whenever necessary (BENNACEUR *et al.*, 2014). Models at Run-time systems are being tested as a solution for self-adaptive and self-aware systems that are capable of self-healing and automatic integration (TRAPP; SCHNEIDER, 2014; AUTILI; INVERARDI; TIVOLI, 2015; AUTILI *et al.*, 2017)

2.7 Model-Oriented Programming

Model-Oriented Programming is a programming paradigm created with the intent of further tightening the gap between code and modeling (BADREDDIN; LETHBRIDGE, 2013). Concepts of this paradigm were used to establish the software category proposal that is described in this thesis.

The need for a programming paradigm was motivated by several problems identified by the authors when developers try to evolve the code from diagrams in a MBSE approach. While MDSE adds the ability to automate this effort, the need to synchronize the models is still required. The round-trip engineering approaches are attempts to cope with this issue, but they still require the hurdle of adding annotations to code to maintain the required semantics (FORWARD; BADREDDIN; LETHBRIDGE, 2010).

In order to realize Model-Oriented Programming, a programming language named Umple was devised (FORWARD; BADREDDIN; LETHBRIDGE, 2010). This language was based on both the Java Programming Language and UML, with the intent of allowing programmers to write code without losing semantics from the design models as well as rapid prototyping thanks to code generation.

Umple allows developers to compile programs and diagrams using a single language representation to avoid information loss that may be caused when transitioning from design to coding activities. Umple also includes a run-time system to execute code while keeping objects in an easy to export format, which allows developers to save the data of the prototype for easy debugging and testing, which is also explored in test-driven development techniques (BADREDDIN; FORWARD; LETHBRIDGE, 2014).

2.8 Web Services

In this thesis, Web Services are used as a technological domain for the proposed techniques. Web Services (WS) are software systems designed for machine-to-machine interaction over a network, according to the definition by the W3C Working Group (W3C Working Group, 2004). All the WS examples provided within this thesis employ Hyper Text Transfer Protocol (HTTP) over the Internet in order to allow this interaction.

In the same manner, the examples provided also employ Extensible Mark-up Language (XML) for data interchange, according to W3C recommendations (W3C Working Group, 2008). XML is a structured language definition to represent data in text format suited for machine interpretation.

More specific languages can be represented and structured as XML. In order to define the specific language rules, a meta language format named XML Schema Definition (XSD) is employed. XSD is referred in this thesis as a meta-meta-language, because XSD is itself represented as XML, while being also defined by a XSD (W3C XML Working Group, 2012).

There are different standards or recommendations for WS systems. For flexibility, the examples provided within this thesis are based on two types of web services: Representational State Transfer (REST) and Simple Object Access Protocol (SOAP). REST is an architectural style for data interchange which employs HTTP features, namely universal resource identifier and action verbs (method operations). This style allows a uniform specification for manipulating web resources (FIELDING; TAYLOR, 2000).

SOAP is a WS protocol recommendation devised by a working group published by the W3C (W3C Working Group, 2007). SOAP can be used over HTTP for machine interaction. The XML format is recommended for structuring both requests and responses of SOAP messages. Following the recommendation, data objects of the software systems we have devised are serialized into XML streams, which are then encapsulated by another XML structure named as SOAP Envelope. This envelope is used to transmit the object data with headers and the actual object data, although it is not limited as a protocol to be used over HTTP and the Internet.

Web Services Description Language version 1.1 or Web Services Definition Language version 1.0 (WSDL) are language specifications for interfaces and data types that are provided by a web services server. The data types of the WDSL are based upon XSD definitions and provide specific constructions to define the operations served by the server along with the input and/or output parameters. Therefore, artifacts in WSDL format serve as interface models. A XSD for WSDL is also available, which specifies how WSDL artifacts can be represented in XML format (CHRISTENSEN *et al.*, 2001).

The OMG has published a recommendation for structuring models and meta-models as XML files according to the XML Meta-data Interchange (XMI) specification. XMI also has its own definition in XSD format (Object Management Group, 2014).

2.9 Final Remarks

This chapter contains theoretical topics related to the work developed which is described within this thesis. Among these topics, MDSE principles, processes and process discovery techniques and models were covered. A brief introduction on Web Services was also provided. The goal is to provide an initial reference based on the literature review, as well as introducing terms used throughout the development of the research presented within this thesis. Following the study of problems developers face when applying MDSE in their projects, in Chapter 3 a systematic mapping is presented to further contextualize this problem.

CHAPTER

3

SYSTEMATIC MAPPING ON MODEL-DRIVEN DEVELOPMENT SUCCESS CASES FOR DOMAIN-SPECIFIC AND GENERAL PURPOSE APPROACHES

3.1 Initial Remarks

This thesis was written with the intent of proposing advances on the software development by using modeling facilities and, to achieve that, it is important to identify the application domains in which MDSE is successful or not. With this knowledge, it would be possible to better understand MDSE potentials and pitfalls within different contexts, allowing to propose a generalization towards other similar domains.

However, since no secondary studies have been found addressing this topic, it was necessary to plan and conduct a systematic literature mapping with three objectives: identify the software domains where model-driven techniques applications are successful; identify the software domains where model-driven techniques applications are unsuccessful; and identify challenges when applying these techniques to general purpose software development.

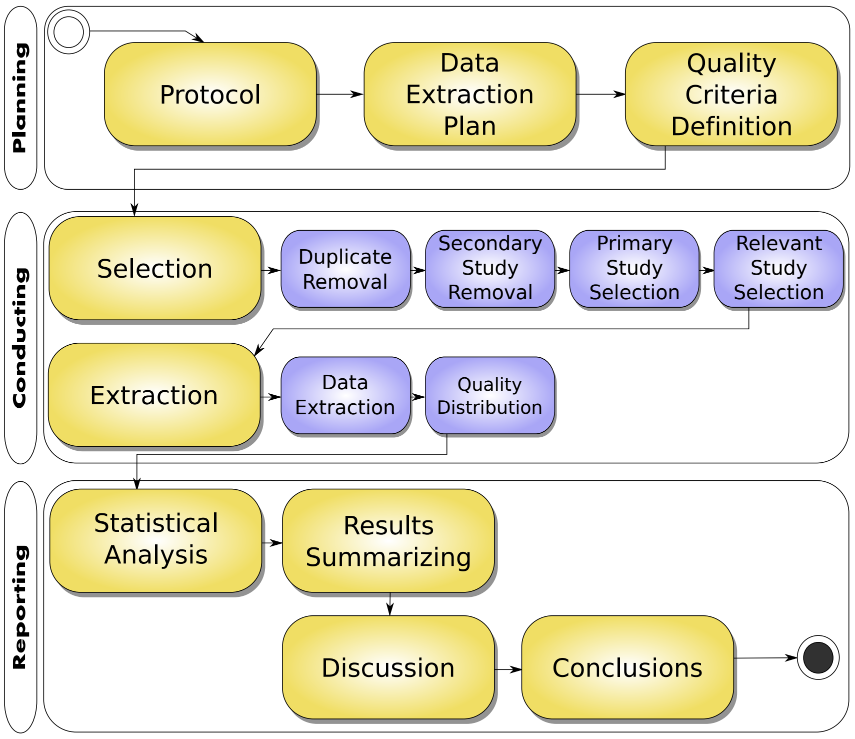
This chapter is organized as follows: In Section 3.2, systematic mapping (SM) concepts are presented, as well as the activities done during the conduction of the SM. The quantitative results gathered from the SM process are listed in Section 3.3. These results are discussed in Section 3.4, which is done by also including qualitative data. Limitations and threats to validity of this study are described in Section 3.5. Works related specifically to this chapter are cited in Section 3.6. Finally, the conclusions for the study are in Section 3.7.

3.2 Systematic Mapping Execution

A Systematic Mapping (SM) is a specific method of literature review. It allows identifying and quantifying primary studies relevant to research questions in a specific knowledge area (PETERSEN *et al.*, 2008).

This SM was executed according to the guidelines by Kitchenham and Charters (2007), which were defined in order to establish a systematic and repeatable literature review process. They recommend three phases for the executed process: planning; conducting and reporting, as illustrated in Figure 10. Rounded rectangles represent activities and sub-activities, while the directed lines present their sequence.

Figure 10 – Systematic Method Process Model Diagram



Source: Created by the author

3.2.1 *Planning*

The Planning Phase is the first phase visible in Figure 10. It contains activities in which the researchers develop a document named as “Protocol”, which is shown in Subsection 3.2.4. During the Planning Phase, “Data Extraction Plan” and “Quality Criteria Definition” are also developed. By defining the execution procedure in a review protocol, this process instance becomes controlled and repeatable, which is one of the primary objectives of following a systematic approach.

3.2.2 Conduction

The Conduction Phase is composed by the “Selection” and “Extraction” activities. These activities must be performed by following the established protocol.

The extraction activity deserves a further explanation. Its objective is to extract data and fill the extraction form (“Data Extraction”), which is composed by categories planned as described in the protocol. The extracted data is employed to distribute the studies into the established quality criteria (“Quality Distribution”).

3.2.3 Reporting

The Reporting Phase has activities related to the data summarizing. The first activity involves performing statistical analysis on the quantitative data (“Statistical Analysis”). Then, these results are summarized into text and plots. It also involves discussing the results in the effort of identifying new insights related to the study objects. This is the last phase visible in Figure 10 and then, the process is finished.

3.2.4 Protocol

The complete protocol definition is visible on Table 1. This table contains two columns. These columns are arranged into field name and value pairs that include the objective, questions, intervention, control, population, results, application, keywords, source selection criteria, study language, search engines and study selection criteria. Among these fields, the questions and study selection criteria are frequently referenced during the results of the conduction phase presented in Section 3.3.

The most important item of the protocol is its objective. By intending to identify MDSE success/failure cases and challenges, two questions were devised. The first question aims to identify the success cases and the failures in specific domains. The results of this question are important because the search for challenges in MDSE outside these domains are also part of the objectives. If the success cases became too ubiquitous, then, one could argue that the secondary question is irrelevant. This is because the secondary question is related to challenges encountered when applying MDSE into domains that do not have visible success cases. Without visible success cases, there would be a lack of tools or methodology, thus making the challenges more apparent.

Another important item of the protocol is the set of inclusion and exclusion criteria. The inclusion criteria are marked with a leading “I” whereas the exclusion criteria are marked with an “E”. The “I1” and “I2” were created when intending to answer the primary question, while the “I3” is related to the secondary question. The exclusion criteria are employed to remove the unrelated studies and other results that are not primary studies.

Table 1 – Protocol Definition

Protocol Item	Item Description
Objective	The objective of this literature review is to identify success cases and challenges in domain specific and general purpose MDSE approaches.
Primary Question	What are the specific domains in which developers have achieved success by employing MDSE?
Secondary Question	What are the the general purpose MDSE approaches and what are the challenges to create such approaches?
Intervention	Studies related to MDSE approaches and their challenges must be identified and categorized.
Control	The search results must involve a list of studies related to the questions that are known by the researchers. This list includes articles and books by Pastor, Whittle and Czarnecki.
Population	MDSE application cases.
Results	Quantitative data on approach frequency distribution within domain categories. Qualitative data on reported challenges.
Application	This secondary study is provided as support to research regarding MDSE.
Keywords:	Software Development; Approach; Support; MDSE and MDD.
Source selection criteria:	Source must index studies on Computer Science, Mathematics or Engineering. Source must allow Boolean operators. Source must be accessible by the researchers.
Study Language:	at least title and abstract must be in English.
Search Engines:	ACM; IEEE; Scopus; Engineering Village/Compendex
Selection Criteria:	**Inclusion:** • I1 - Primary studies that present a success case of MDD, MDSE, DSL or MDA in a specific domain; • I2 - Primary studies that present a non success case of MDD, MDSE, DSL or MDA in a specific domain; • I3 - Primary studies that present challenges of applying MDD, MDSE, DSL or MDA in general purpose projects; **Exclusion:** • E1 - Unrelated to MDD, MDSE, DSL or MDA. • E2 - Not a primary study.

3.2.5 Search Strategy

The searches were divided into different search sessions, which were conducted to collect three different categories of studies.

The first category is focused on the software development approaches using modelling, i.e. not limited to MDSE.

The second category is related to any study that involves model orientation, whereas the third category is related to any study involving models at run-time.

Therefore, three search strings were created, one for each category. It is worth mentioning that these search strings have been constructed by joining the basic keywords defined on Table 2, complemented with synonyms and related terms.

The keywords shown on Table 2 were used as basis to create the first category search string. The final search string is obtained after a conjunction operation (represented by “$\wedge$”) applied to the table rows and a disjunction operation applied among the synonyms of each keyword. Therefore, the final search string is (A) $\wedge$ (B) $\wedge$ (C) $\wedge$ (D).

For the second category (Model Orientation), the keywords shown on Table 3 were used. Since there is only one row, the disjunction operation is applied among the synonyms of the keyword, with the intent of capturing any study related to this keyword.

The third category (Models at Run-time) was also defined to capture any study related to the specified keyword. Therefore the keyword shown on Table 4 was used. Since there is only one row, the disjunction operation is applied among the synonyms of the keyword.

The searches were planned to be carried out through the following search engines: ACM Digital Library[1], IEEE Xplore[2], Engineering Village Compendex[3], Wiley Digital Library[4], Web of Science[5], Science Direct[6], Elsevier Scopus[7], Springer Link[8] and Google Scholar[9],

However, some search engines were canceled after search sessions, since it was not possible to calibrate the results by using the same search string construction rules.

The search was then concluded through the following search engines: ACM Digital Library (DL), Engineering Village Compendex (EV), IEEE Xplore (IEEE) and Elsevier Scopus (Scopus).

Their update dates may vary, as specified in Table 5. The initial search sessions were executed on May 30th, 2014 by collecting studies from all reported search engines (1651 studies). This review was then updated during the project by repeating the searches on the specified search engines, effectively reaching 4859 studies. It is important to mention that due to the broad nature of this search, this is a continuous work of literature review that requires periodic updates.

1 <http://dl.acm.org>
2 <http://ieeexplore.ieee.org/>
3 <http://engineeringvillage.com>
4 <http://onlinelibrary.wiley.com>
5 <http://wokinfo.com/>
6 <http://sciencedirect.com>
7 <http://scopus.com>
8 <http://link.springer.com>
9 <http://scholar.google.com>

Table 2 – MDSE Software Development Approach Search String Definition

Identifier	Keyword	Synonyms and Related terms
A	Software Development	• software development; • software engineering;
B	Approach	• approach; • process;
C	Support	• tool; • support;
D	MDSE and MDD	• mdd; • development; • mde; • engineering; • software; • mda; • model-driven architecture; • model driven architecture; • Model-Driven; • model; • driven; • model-driven. • model-oriented • model oriented.

Despite not updating all engines completely, these searches returned more studies from past years that were not collected during 2014 searches, effectively exceeding the number of studies found during initial searches. Nevertheless, all studies returned by engines were thoroughly evaluated according to the processes established previously in this section.

Table 3 – Model Orientation Search String Definition

Identifier	Keyword	Synonyms and Related
E	Model Orientation	• model-orientation; • model orientation; • model-oriented • model oriented.

3.2.6 Study Selection

After ensuring that there were no duplicated studies, the review was conducted by analyzing studies returned by each search engine. The search engine priority was set by reviewing first the engine that returned the higher number of previously known studies.

The data extraction form contains fields that must be filled during the extraction phase. These fields where planned since the inception of this secondary study. The planned form contains three fields:

1. Identified Success Case Domain;
2. Identified Failure Case Domain;
3. Identified MDSE problem or challenge.

The valid values for each of the enumerated fields are presented on Table 6. These valid values are defined as nominal sets, i.e., groupings of enumerated and named items that represent categories of studies that include information that is relevant for this secondary study. All nominals, except the domain nominals, were completely planned prior to the execution.

The first nominal set is named as Domain Set, which contains 45 nominals, including domains related to Web, Embedded Systems, Business Information Systems, Telecommunications and Networking, Industrial Control Systems, Military, Parallelism, Simulation, Computer Aided Design, Education and Computer Games. A complete list of these nominals is shown on Table 7.

It is important to establish that these nominals have been evolved throughout extraction as new domains were categorized. They were also defined to allow hierarchical analysis. For example, every nominal which starts with "Embedded System" is considered as child of the first "Embedded System" nominal, then, upon counting the numbers of studies that are related to this domain, the studies marked with any child nominal are also counted along with their parents. The domains were distributed into application and technological domains during results analysis.

Table 4 – Models at Run-time Search String Definition

Identifier	Keyword	Synonyms and Related
F	Models At Run-time	• models at runtime; • models at run-time; • models at run.time; • models at run time; • model at runtime; • model at run-time; • model at run.time; • model at run time; • models @ runtime; • models @ run-time; • models @ run.time; • models @ run time; • model @ runtime; • model @ run-time; • model @ run.time; • model @ run time; • models@runtime; • models@run-time; • models@run.time; • models@run time; • model@runtime; • model@run-time; • model@run.time; • model@run time;

The second nominal set is the Problem Set, which contains the nominals “Methodology Problem”, “Maintenance Problem”, “Testing or Validation Problem” and “Tools Problem”. It is important to mention that throughout the study the focus on “Tools Problem” was diminished since it is unclear to define whether the study exposes MDSE problems or the authors were simply encouraged to create new tools. Moreover, this issue was already covered in the literature (WHITTLE *et al.*, 2013).

The third nominal set is the Validation Set, which contains the nominals “Case Study”,

Table 5 – Search Sessions and Updates

Search Category	Search Engine	Last Update	Returned Study Count
MDSE	ACM DL	May 30th, 2014	26
	IEEExplore	September 6th, 2017	405
	EV	May 30th, 2014	655
	Scopus	September 8th, 2017	2805
MO	IEEExplore	January 1st, 2018	18
	EV	January 1st, 2018	37
	Scopus	January 1st, 2018	42
MRT	IEEExplore	December 9th, 2017	65
	EV	December 9th, 2017	389
	Scopus	December 9th, 2017	417
All	ACM DL		26
	IEEExplore		488
	EV		1081
	Scopus		3264
	Total		4859

Table 6 – Valid Values for Data Extraction Fields

Field Number	Field Name	Field Type	Cardinality	Nominal Set
1	Identified Success Case Domain	Subset of Nominals	zero to many	Domain Set
2	Identified Failure Case Domain	Subset of Nominals	zero to many	Domain Set
3	Identified MDSE Problem or Challenge	Subset of Nominals	zero to many	Problem Set
4	Identified Validation Type	Subset of Nominals	zero to many	Validation Set
5	Presents Solution for an MDSE challenge	One Nominal item	one	Boolean Set

"Experimental or Empirical Study", "Experience Report", "Feasibility Study", and "Proof or Demonstration". The Boolean set contains two nominals, true and false. This set is used to indicate whether the study presents a solution to the challenges or problems identified in MDSE.

3.2.7 Study Quality Criteria

The quality criteria were divided into two groups. The first group is used to select the studies related to domain successes and failures. In this case, we have defined that these studies must not present the domain simply as a case study, that is, to avoid papers that use a domain as a validation without providing a practical success or failure report.

Considering the second group, it is related to the studies that present challenges or problems related to MDSE. We have planned strategies for defining if these studies are relevant

Table 7 – Complete Domain Set Nominals

Academic	Math/Scientific
Access control	Middleware
Answer Set	Military/Defense/Aerospace
Autonomous Mobile Robotics	Mobile
BioInformatics	Multimedia/Audio/Video
Business	Network
CAD Tool	Parallelism
Cloud Computing/DataWarehouse	Questionarie
Communication/Chat	Real Time System
Control System	Robotics
E-commerce	Simulation
Education	SOA/WEB Service
Embedded System	Software architecture
Farming	SoS
Fault Tolerance / Adaptative	Space - Satellite
Game	System Virtualization
Government	Telecom
Hardware	Tourism
Health	TV
Human Interaction	Ubiqua Pervasive
Industrial	User-Interface
Information System	WEB
Large Scale	

for our secondary study. Therefore, we have defined that only the studies that either have a solution for the challenges or contain a validation should be carried into the discussion activity.

3.2.8 Employed Tools

A custom set of tools was developed by the author to be employed during the conduction phase. Their requirements involved supporting the hierarchical nominals and allowing the researchers to cooperate on the same review and to provide real-time reports about the review evolution and preliminary summarizing via a web page that features graphics and descriptive statistics. More details of these tools are also available as part of the packing documents.

3.3 Secondary Study Results

This section contains the summarizing of results, which was carried out after conducting the extraction phase. Qualitative and quantitative data that were used to answer the research questions are provided.

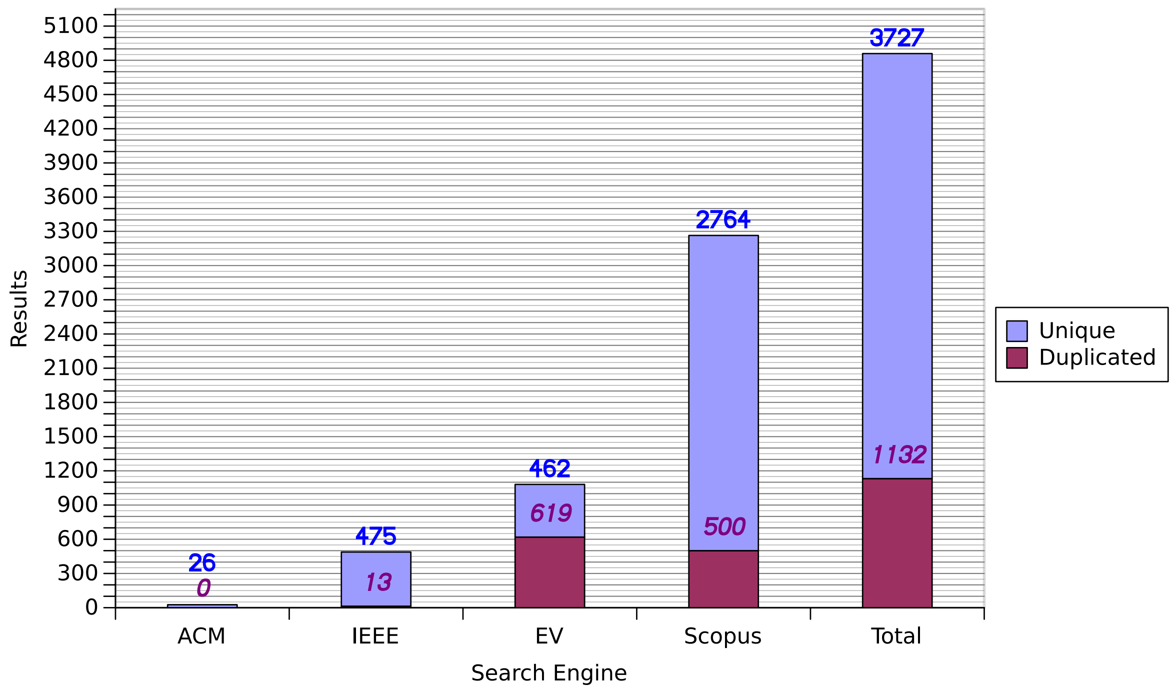
3.3.1 Search Results

The aim of this subsection is to provide information regarding the results returned by the search engines. The searches returned 4859 studies, in which, 3727 were not duplicated.

Figure 11 contains a plot that was created to allow a general visualization of the distribution of collected studies. The graph was designed as a stacked bar plot, which allows the viewer to compare the portion of duplicated and unique results returned from each search engine.

Since the search sessions were distributed into three categories, they are also specified as separate plots. MDSE approaches are represented in Figure 12. Model-Oriented studies are shown in Figure 13. Similarly, Models at Run-time studies are represented in Figure 14.

Figure 11 – Source Distribution



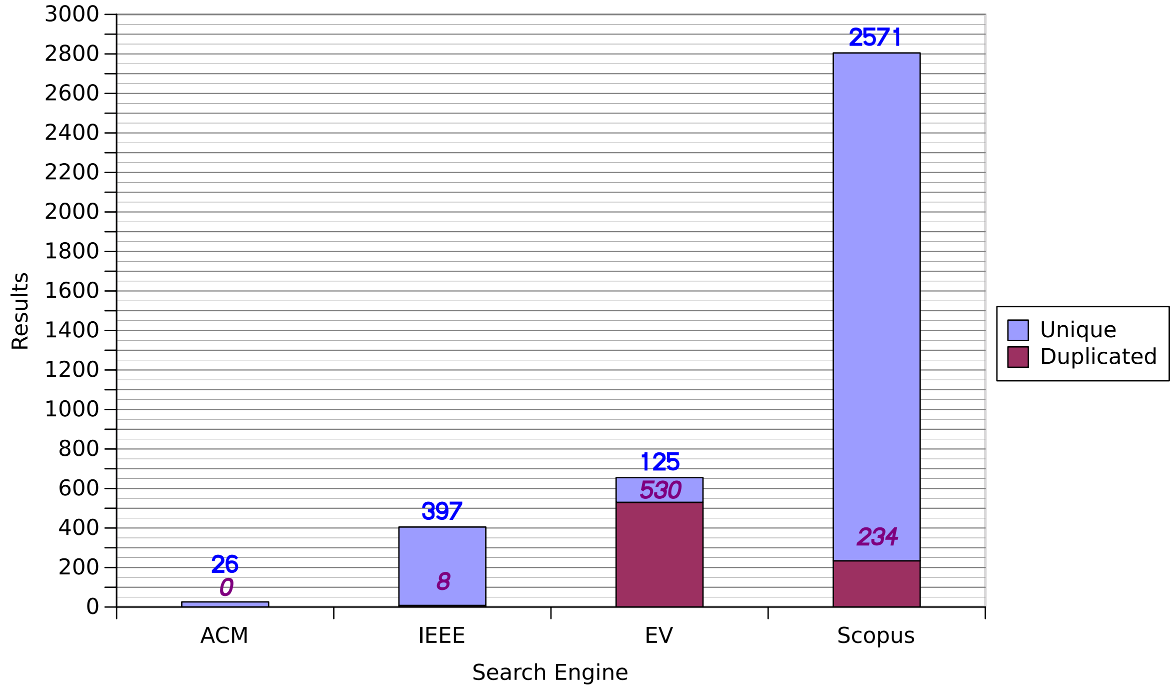
Source: Created by the author

The columns of Figures 11 to 14 are named as “ACM”, “IEEE”, “Scopus” and “EV” since they represent, respectively, the search engines ACM Digital Library, IEEE Xplore, Elsevier Scopus and Elsevier/Engineering Village Compendex.

In this review, studies were not filtered by their publication year. The oldest study that passed the selection phase was published in 1985 and written by Hoffnagle and Beregi (1985). Their study is related to automated software generation, however, since there is no explicit model as input, it was not categorized as MDSE.

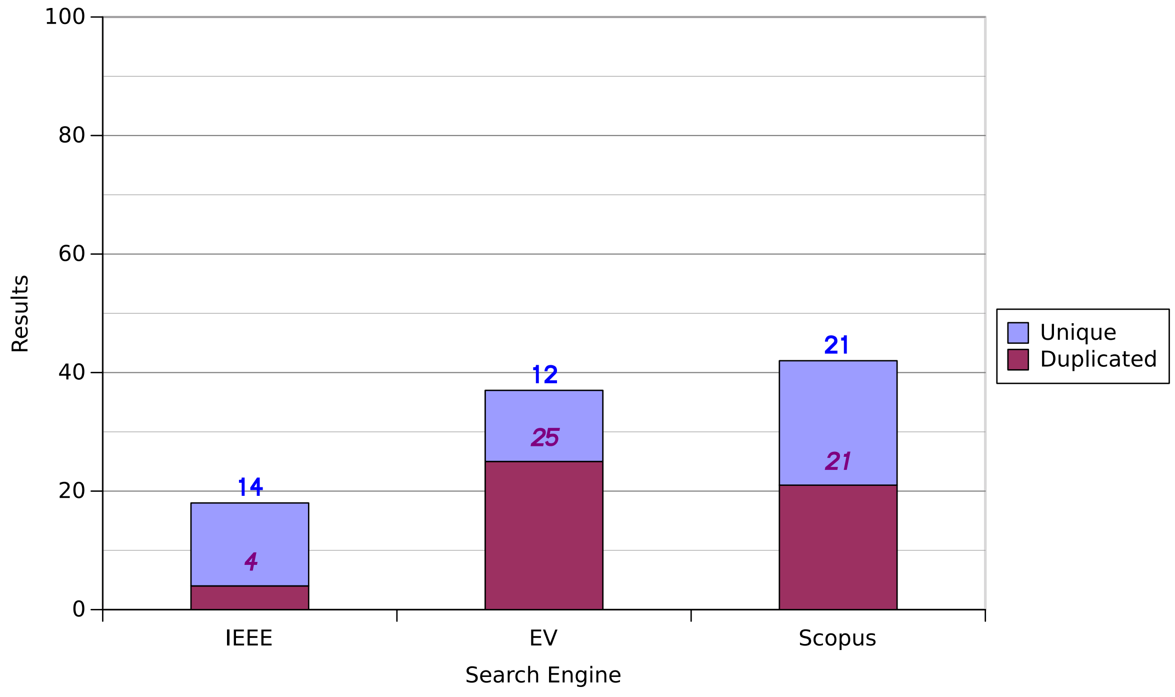
In order to improve the visualization of the plots here shown, it was established that they would start from one year before the year that got the oldest results. In every bar plot, the

Figure 12 – MDSE Approaches Search Session Results



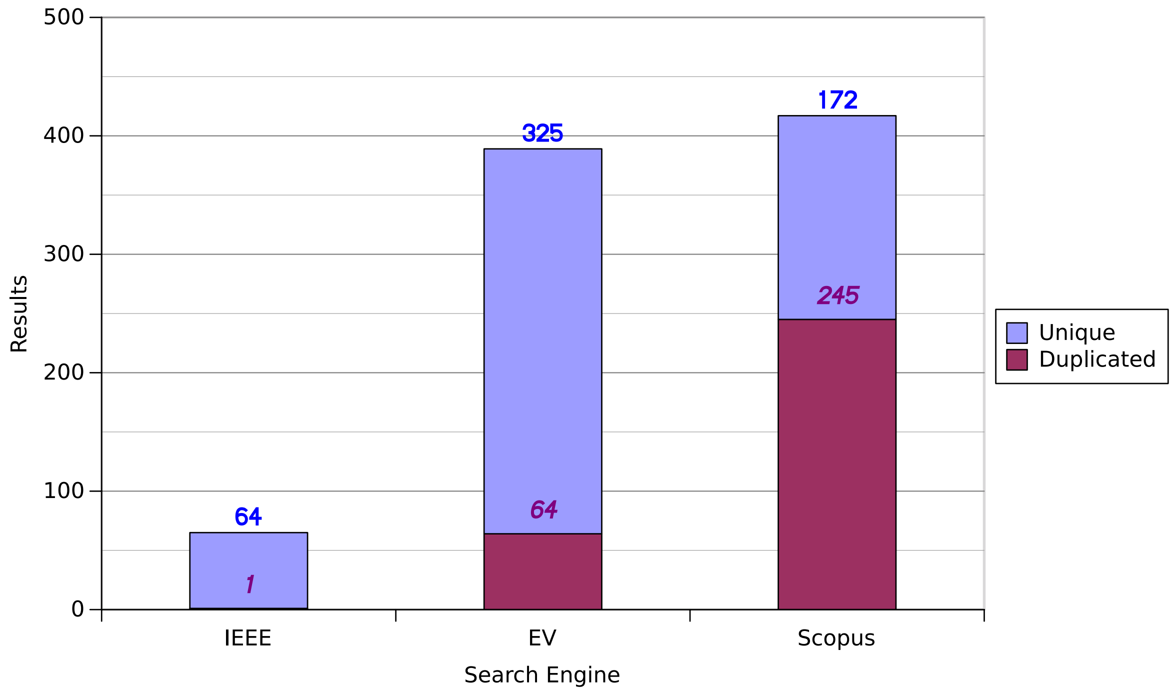
Source: Created by the author

Figure 13 – Model Orientation Search Session Results



Source: Created by the author

Figure 14 – Models at Run-time Search Session Results



Source: Created by the author

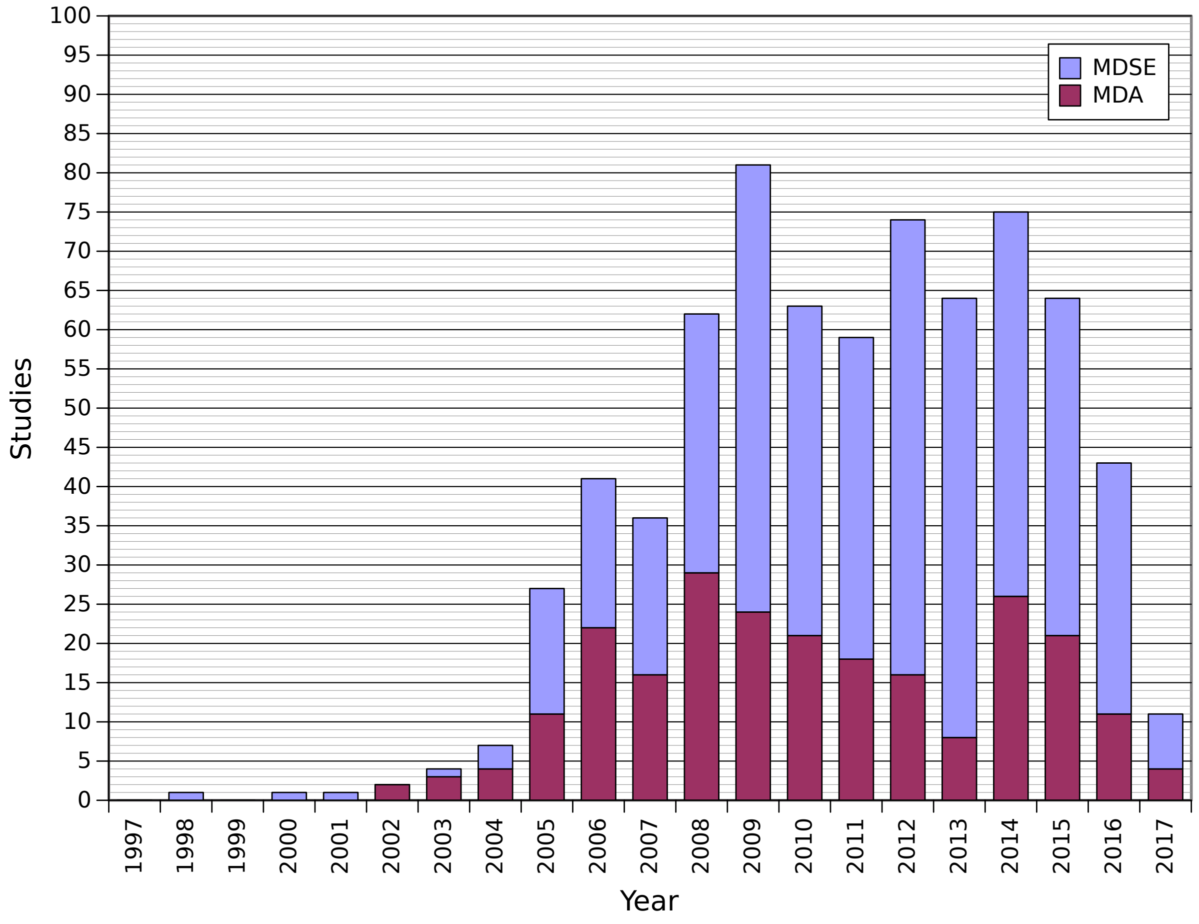
vertical axis contains the number of studies, whereas the horizontal axis may represent categories, process phases or years. In Figure 15, there is a plot that allows the reader to visualize the evolution of MDSE-related studies, which also represents the selected references.

Alkadi and Carver (1998) wrote the oldest study that was considered as related to MDSE by this review process. It was published in 1998. They created an approach which employs models for test case generation.

During the review process, we have also categorized the studies which have abstracts that present or discuss MDA-based approaches. Then, these numbers were stacked into the plot of Figure 15. This figure allows further discussion of the secondary question, which is related to MDSE challenges. One could argue that the MDA use seems to be diminishing while the MDSE have been published in a more constant pace.

After suggesting that MDA use may be diminishing, we have also categorized the use of MDA model levels, as shown in Figure 16. The most common model level that was actively declared in articles was PIM, followed by PSM. This analysis allows to identify that the highest abstraction level (CIM) is hardly declared as used in most MDA approaches. This data leads to discussions about how the highest abstraction level of MDA models, the CIM, is important in development.

Figure 15 – Evolution of Studies Related to MDSE and MDA



Source: Created by the author

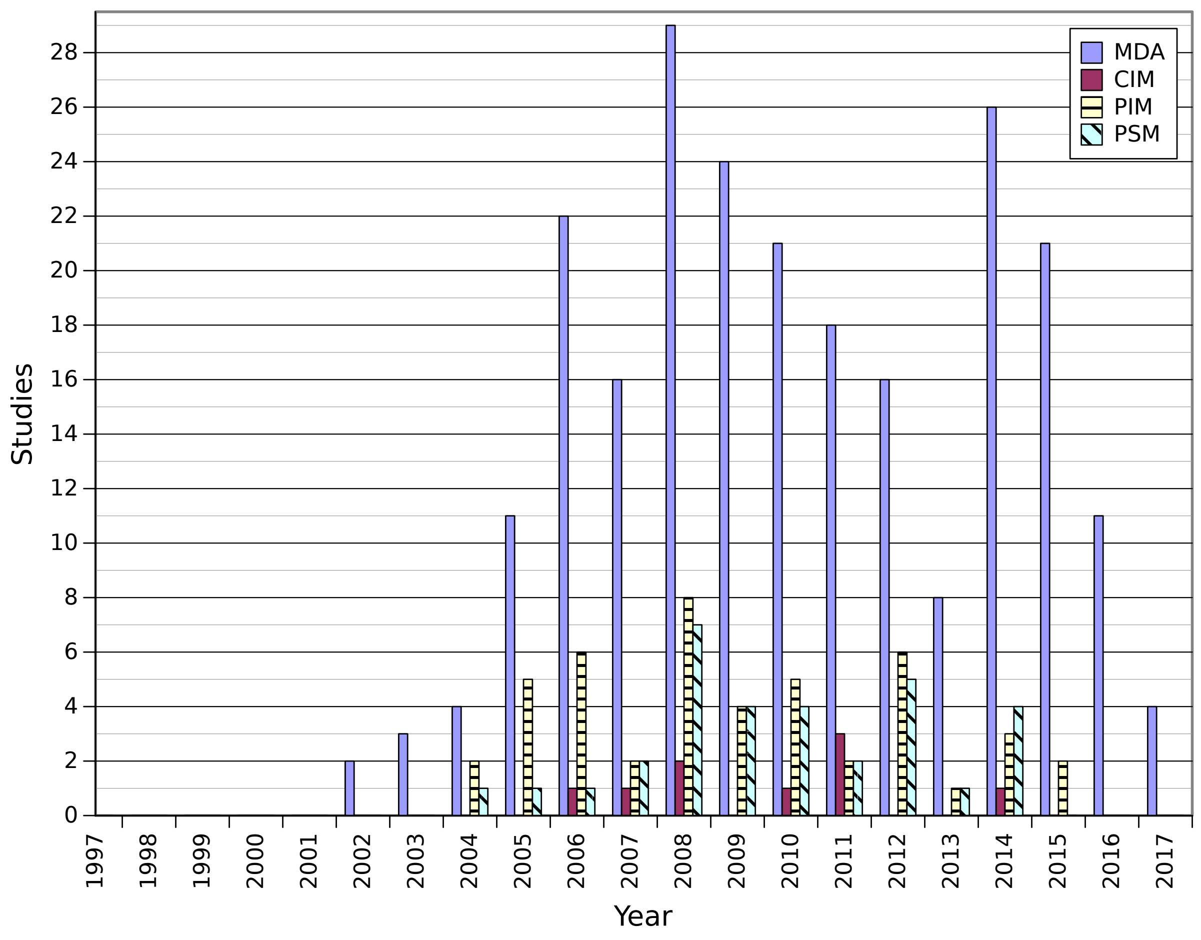
3.3.2 Domain Categorization: Results

This section is related to the primary question of the study, which involves finding successful MDSE application and technological domains. Among the selected studies, there was no study reporting unsuccessful MDSE development cases. Therefore, this section is focused only on the success cases.

The aim of categorizing the studies was to identify application and technological domains in which MDSE has been successful. In Figure 17 there is a plot that allows easy visualization of a large set of data gathered during this categorization process. It is possible to identify the evolution of the most common domains. As seen on the legend, there are four domains, namely “Embedded”, “Web”, “Network” and “Business”, which represent the domains of Embedded Systems, Web Systems, Network Systems and Business Information Systems, respectively.

Although not planned prior to execution, it has been identified that the use of MDSE in Embedded Systems Domain is very scattered among different subdomains. To avoid concerns related to how broad is the categorization of this domain, we have identified Embedded Systems

Figure 16 – MDA Model Levels Evolution



Source: Created by the author

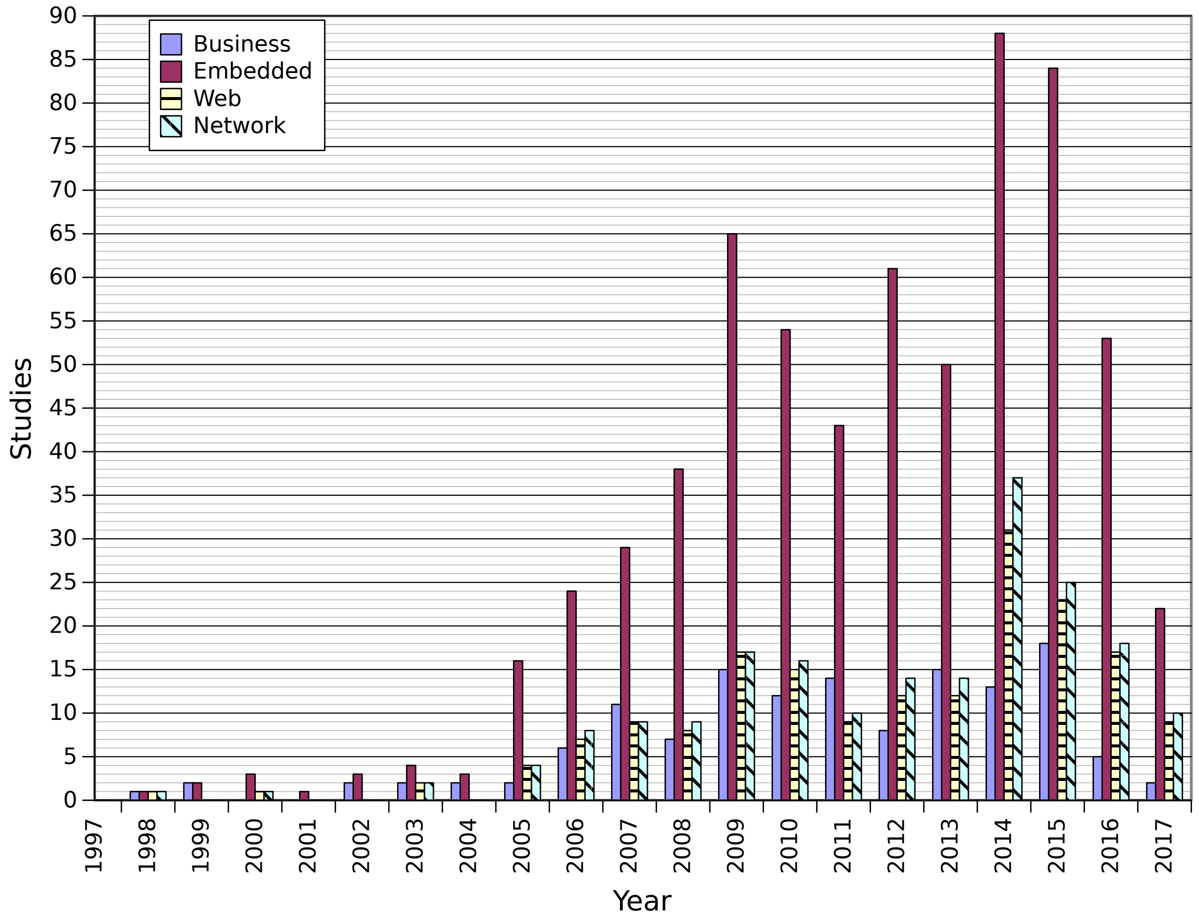
subdomains which employ MDSE. These subdomains are presented in Figure 18, which contains a plot showing the evolution of number of studies. With the intend to ease visualization, all subdomains are categorized in order for each year. The subdomains "Avionics and Aviation", "Robotics", "Agriculture", "Cruise Control", "Home Automation", "Sensors and Actuators" and "Road Vehicle" are distributed per year. It is important to mention that the "Total" bar is also the total count of all studies related to Embedded Systems with or without a specific subdomain.

3.3.3 MDSE Challenges: Results

The goal of this subsection is to provide quantitative results regarding the number of studies that present challenges on applying MDSE into general purpose projects.

After the process, eight studies that contain references to maintenance challenges were identified. In the same manner, nine studies that discuss methodology challenges were identified. Considering these studies, there is the total number of seventeen unique studies, since there is a single study which is common in both listed categories. Further discussion regarding this data is

Figure 17 – Evolution of Most Common Domains



Source: Created by the author

presented in Section 3.4.

3.4 Results Discussion

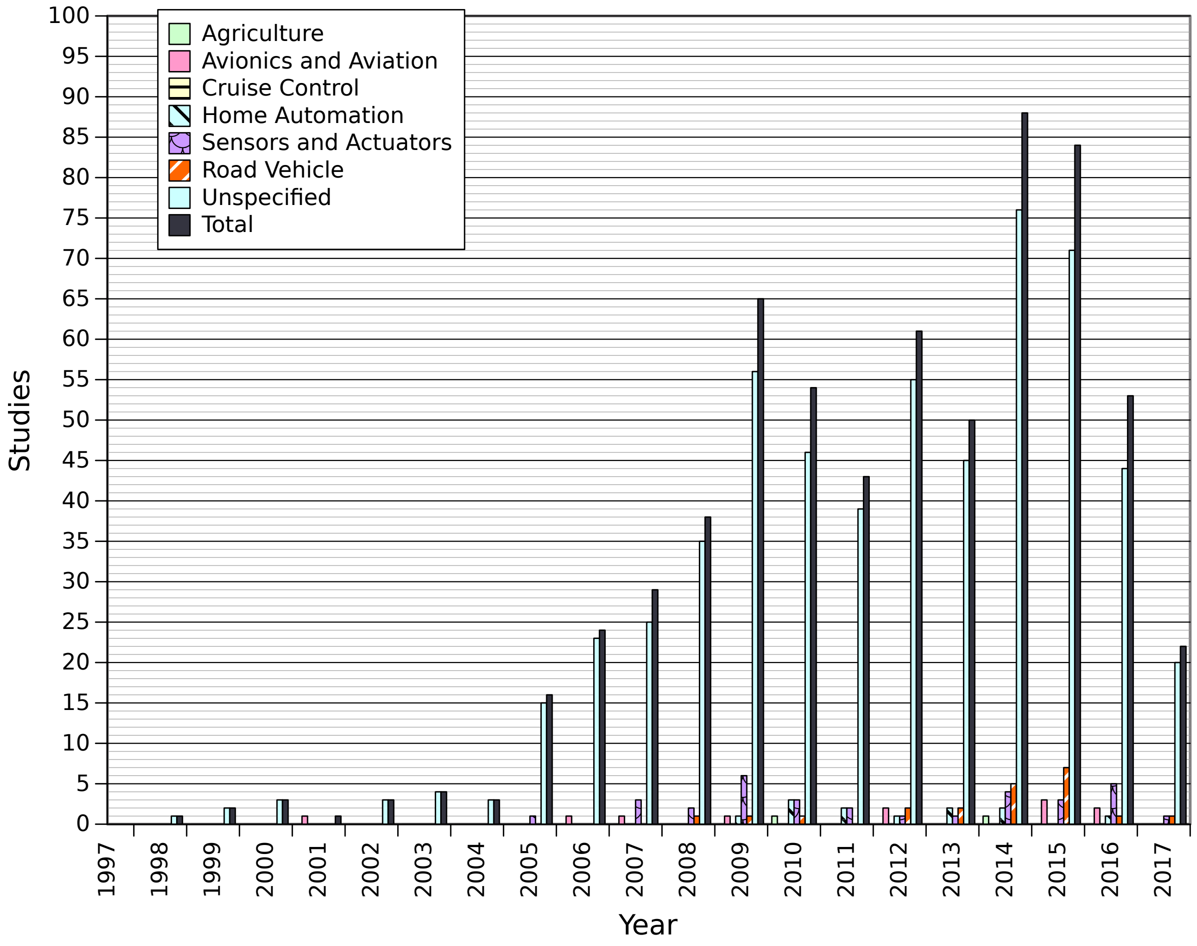
This section presents a discussion about the SM results, along with further qualitative data. It includes discussion on the identified software domains, MDSE remaining challenges and limitations of the study.

3.4.1 Domain Distribution

The results helped to confirm our previous expectations about MDSE tools intended for Business Information Systems, Web and Embedded Systems. By reading the related studies, we believe that a constant concern among these approaches is to accelerate the implementation of data entities and their manipulation.

However, we did not expect that the network domain would achieve high frequency in

Figure 18 – Evolution of Embedded Systems Subdomains



Source: Created by the author

the identified distribution. Most of the MDSE studies in the network domain are related to the development of parallel distribution of software execution.

3.4.2 MDSE Remaining Challenges

The General Purpose Challenge is related to the secondary question of this SM. This question was planned because our research team is investigating the application of MDSE in non-conventional domains and in general purpose projects.

Another result of this SM is that the number of primary and secondary studies related to challenges in unconventional domains and in general purpose methodologies of MDSE application is far below the expected, i.e., 1 in the total number of reviewed studies. The search results were carried out exhaustively, which means that all studies were checked for challenge discussion.

During the conduction we have identified nine works that discuss methodology problems in MDSE besides the proposal of MDA (CERNICKINS *et al.*, 2010; ASADI; RAVAKHAH; RAMSIN, 2008; CHITFOROUSH; YAZDANDOOST; RAMSIN, 2007; NIKULSINS; NIKIFOROVA, 2008; NIKIFOROVA; NIKULSINS; SUKOVSKIS, 2009; SANCHEZ; BARREDA; OCON, 2008; NOYER *et al.*, 2015; NAZARI; RUMPE, 2015; TEPPOLA; PARVIAINEN; TAKALO, 2009). We have also found eight studies that contain information on maintenance problems that occur in projects that employ MDSE (SEIFERT; BENEKEN; BAEHR, 2004; TEPPOLA; PARVIAINEN; TAKALO, 2009; ER; TEKINERDOGAN, 2012; BENDRAOU *et al.*, 2008; WESTFECHTEL, 2014; YU *et al.*, 2014; MANTZ *et al.*, 2015; HOVSEPYAN *et al.*, 2010). It is important to note that there is a single study which is common to both categories. Consequently, there are seventeen studies presented in this subsection.

The summarizing of these studies is presented on Table 8. This table is ordered chronologically. It also contains seven columns. In these columns, it is possible to visualize the reference number of each study, authors, title, problem type and study type. The problem types are split into two columns: methodology and maintenance, which represent, respectively, methodology problems and maintenance problems. Therefore, the study lines that deal with specific problem types have their respective cells shaded in order to indicate that they relate to the problem type.

Chitforoush, Yazdandoost and Ramsin (2007) are among the researchers who identified that MDSE lacks methodology, processes and guidelines to instruct when developers should use each model in a MDA-based project. In their attempt to provide a solution to this problem, they have defined a general methodology framework based on MDA. These authors claim that their framework is flexible enough to be adapted to various processes and needs. They also compared their framework to similar approaches. However, they do not provide validation on the efficiency of their approach. They have suggested this validation as future work but we could not find it published.

Asadi, Ravakhah and Ramsin (2008) are from the same research department and created another solution to the problem identified by Chitforoush, Yazdandoost and Ramsin (2007). They have defined a MDSE development life-cycle , which has as main advantage to propose more specific process definitions to guide developers using MDA. However, the stricter definition could also affect flexibility. The authors compared their approach to related approaches and we could not find a validation study.

Both of these studies have identified problems in the methodology of previous works, but there is no validation whether the problem was completely solved. It is also worth citing the analytical survey by the same authors (ASADI; RAMSIN, 2008), which provides the theoretical foundation used to create their new approach.

Nikulsins and Nikiforova (2008) have described the need for customized processes. They have studied how to adapt Rational Unified Process and Microsoft solutions framework to support MDA. This need for customized process is further described in a more recent work (NIKIFOROVA; NIKULSINS; SUKOVSKIS, 2009), in which the authors report that MDA

Table 8 – MDSE Challenges Summarizing

Reference	Title	Methodology Problem	Maintenance Problem	Study Type
Seifert, Beneken and Baehr (2004)	Engineering long-lived applications using MDA		▨	Experience Report
Chitforoush, Yazdandoost and Ramsin (2007)	Methodology support for the model driven architecture	▨		Method Proposal
Asadi, Ravakhah and Ramsin (2008)	An MDA-based system development lifecycle	▨		Method Proposal
Bendraou *et al.* (2008)	MDA Tool Components: A proposal for packaging know-how in model driven development		▨	Model Manage-ment Model
Nikulsins and Nikiforova (2008)	Adapting Software Development Process towards the Model Driven Architecture	▨		Method for Process
Sanchez, Barreda and Ocon (2008)	Integration of domain-specific models into a MDA framework for time-critical embedded systems	▨		Method for Process
Nikiforova, Nikulsins and Sukovskis (2009)	Integration of MDA framework into the model of traditional software development	▨		Method for Process
Teppola, Parviainen and Takalo (2009)	Challenges in Deployment of Model Driven Development	▨	▨	Experience Report
Cernickins *et al.* (2010)	An outline of conceptual framework for certification of MDA tools	▨		Method Proposal
Hovsepyan *et al.* (2010)	From aspect-oriented models to aspect-oriented code? The maintenance perspective		▨	Experimen-tal Assess-ment
Er and Tekinerdogan (2012)	MoDSEL: Model-driven software evolution language		▨	Language Proposal
Westfechtel (2014)	Merging of EMF models: Formal foundations		▨	Algorithm for Mainte-nance
Yu *et al.* (2014)	From model-driven software development processes to problem diagnoses at runtime	▨		Generator Dev. Method
Nazari and Rumpe (2015)	Using software categories for the development of generative software	▨		Generator Dev. Method
Mantz *et al.* (2015)	Co-evolving meta-models and their instance models: A formal approach based on graph transformation		▨	Algorithm for Mainte-nance
Noyer *et al.* (2015)	A model-based workflow from specification until validation of timing requirements in embedded software systems	▨		Method Proposal
Gottardi and Braga (2016)	Model-Oriented Web Services	▨	▨	Part of this project

provides no guidelines for activities, roles, phases and responsibilities.

While facing a similar issue, Sanchez, Barreda and Ocon (2008) and Noyer *et al.* (2015) have noticed the lack of a model-based process for embedded systems development.

Cernickins *et al.* (2010) also have described a methodological problem, however, their focus is on tool certification. Thus, their certification framework contains guidelines that can be employed to identify if a tool set or a project is lacking an important activity or feature which they claim to be necessary.

Besides these studies, Nazari and Rumpe (2015) have devised instructions specifically for developing software generators, which is a specific concerning activity related to MDSE projects.

Seifert, Beneken and Baehr (2004) have written an experience report where they explain that the use of MDSE increases dependency of the tool chain. They argue that this problem is not only limited to custom made tool chains, because tools may be updated and become incompatible to the older model instances. This is caused by changes on the language definition. For instance, new UML definition versions are made available and tool developers may follow the new definitions and break backward compatibility (SEIFERT; BENEKEN; BAEHR, 2004).

Bendraou *et al.* (2008) have discussed the need for packaging metadata about the artifacts used within projects that employ MDA principles. This packaging would then support the maintenance activities.

Hovsepyan *et al.* (2010) have studied the impact of code-generation on software maintenance. The most important contribution of this study, when compared to others presented herein, is the in depth statistical analysis of the maintenance impact using metrics to compare the results in a quantitative manner. However, it focuses on specific models for a programming paradigm (HOVSEPYAN *et al.*, 2010).

Er and Tekinerdogan (2012) describe a language named "MoDSEL" (Model-Driven Software Evolution Language). They claim that this language can be used to compare models, track their changes and identify maintenance impacts. Therefore, this is a solution that deals with the maintenance problems that may be found in software projects employing MDSE (ER; TEKINERDOGAN, 2012).

Westfechtel (2014) and Mantz *et al.* (2015) have discussed the need for formal foundations to merge models. These foundations are employed to implement tools to handle maintenance issues faced by developers when dealing with version conflicts that may arise during development and maintenance.

Yu *et al.* (2014) have discussed the problem that arises when it is required to provide maintenance to code generators. In their study, they have devised a tool to help debugging software produced by model-driven development.

The studies shown on Table 8 span eleven years of publications. Roughly in the middle of these years would be 2009. Prior to 2009, 5 out of 7 studies were focused on the methodology problem. After 2009, 6 out of 8 studies were focused on the maintenance problem, which could indicate a trend on the research efforts.

Incidentally, the first study that covers both categories was published in 2009. It was written by Teppola, Parviainen and Takalo (2009) and contains an experience report created by applying surveys on software development companies. These authors have described that software developers using MDSE perceive both maintenance and lack of methodology issues. They claim that despite the advantages that have been experienced by the developers while using MDSE, there are still several issues to be treated. The authors conclude that the developers are optimistic hoping that these issues are solved because they approve the use of MDSE regardless of its current limitations (TEPPOLA; PARVIAINEN; TAKALO, 2009).

After learning about most of these works, a proposal for evolving MDSE into a new development method, including a development method with maintenance flexibility was defined within this project (GOTTARDI; BRAGA, 2016) and returned by the search engines after the SM update. Further details of the proposal are described within Chapter 5.

It is not part of this study to discuss other secondary studies, although it is important to list related studies. The previous version of this systematic mapping has been published (GOTTARDI; BRAGA, 2015) and collected during its update . This previous version was conducted in 2014, which included all studies up to the one authored by (ER; TEKINERDOGAN, 2012).

An analytic survey, which is part of the work described by Asadi and Ramsin (2008), should be cited as an initial discussion of limitations of MDA and MDSE. It is worth mentioning that there are other two works that were collected as control for this secondary study, and were not added to Table 8: Whittle *et al.* (2013) and Whittle (2013). These studies also indicate that the evolution of MDSE tools should focus more on the human aspects of developers, who still face difficulties when trying to use them. Further details on related works are presented within Section 3.6.

3.5 Study Limitations and Threats to Validity

The aim of this subsection is to provide details on the identified limitations and what we have done to mitigate them. The limitations were categorized by their origin, which include “Search Strategy”, “Study Selection”, “Data Extraction”, and “Researcher Bias”.

Search Limitations

There are a few limitations related to our search strategy. The first of them is regarding to the search string. After calibrating the string in order to achieve most relevant results, it is possible that the string lost part of the original intended semantics and may fail to return some of the intended studies.

To mitigate this problem, we have identified which search engines indexed the preliminary known primary study defined as “control” in the protocol. Then we confirmed that the search string was enough to cause the search engine to return the indexed known studies.

The number of search engines is less than the initially planned. It was originally intended to include the databases by ISI Web of Knowledge, Science Direct, Wiley InterScience and Google Scholar. The decision to cancel conduction of results carried by Springer Link was late in our process. This was decided because this search engine provided a small relative number of relevant studies and the high number of results was affecting the review time.

Study Selection

The guidelines by Kitchenham and Charters (2007) have been defined considering that the selection activity should only be used to define which studies should proceed to the extraction activity. However, as a slight variation of the method, the authors preferred to categorize every study since the selection phase with the intent of providing a broader view on software domains and modeling approaches.

The impact of this approach is unknown. In our strategy, the selection phase became longer, however, we believe that it was positive to avoid another limitation of this phase: it was not possible to reject studies during the selection phase without providing general categories to the study, which is an evidence that no studies could have been rejected without proper reading of their abstract.

Data Extraction

The main item analyzed during data extraction was identifying the set of categories in which each study should be linked to. This also includes the application domains.

A constant concern during the execution was to provide an exhaustive categorization of every returned study. Several categories were planned before the conduction. After discovering more specific studies, the authors then decided to create an hierarchical category definition interactively in order to provide an in depth report.

Since it was not possible to define if the interactively defined categories should be applied to studies reviewed prior to their definition, only a few categories were selected, and the review was restarted considering the new category set.

Considering that only the studies that provided information related to problems to apply MDSE were selected for discussion, the number of discussed studies has became much inferior than expected. However, as the primary question dealt in this study was to provide a systematic mapping related to the most common domains, this issue should not affect its credibility.

Researcher Bias

Since this work was carried out by two researchers, the risk of researcher bias affecting the results is considerably high.

In order to mitigate this threat, we have defined keywords for each category and established systematic approaches to carry the review as impartially as possible.

All the studies that present information regarding MDSE challenges were reviewed extensively. However, the studies unrelated to the challenges topic were not fully read during the process.

3.6 Works Related to This Systematic Mapping

Model-Driven Development is not a new topic in software engineering. We have found 19 secondary studies among the search results. It is worth mentioning that the work by Asadi and Ramsin (2008) is a secondary study closely related to this one. However, their work is not a systematic mapping but a literature survey. Also, their work is specific to MDA while we intended to identify every MDSE related approach, including non MDA-based.

In summary, there were no systematic mappings among the secondary studies. The identified systematic reviews were not related to the application of MDSE in general purpose use cases.

Whittle (2013) have put forward a survey applied on professional software developers in order to identify their problems when using MDSE in practice. For example, some authors have suggested that MDSE tools would evolve significantly and solve most of challenges (PASTOR; MOLINA, 2007). However, in another work, Whittle *et al.* (2013) have gathered evidences that indicate that this assumption is not accurate, as the evolution of these tools should be more focused on the developers. In their study, they have identified that despite improvement of tools, there are still difficulties faced by developers, related to lack of methods, lack of training and misconceptions about the usage of models. The main similarity of this work is that it also investigates challenges related to MDSE application and we also argument that these difficulties could be all related to the lack of adequate methodology.

3.7 Conclusions and Final Remarks

A secondary study has been presented in this chapter. After reviewing a total of 4859 studies, in which 3727 are unique, the most common domains have been identified as well as discussions related to challenges developers face while attempting to apply MDSE to projects dealing with uncommon or too specific domains.

As part of results summarizing, we have identified that the MDSE success domains are clustered into application and technological domains. This data was presented quantitatively considering the success cases that were not only used as case studies. The success cases indicate that MDSE has reached production levels for specific domains. In this manner, it is suggested that MDSE is recommended for specific domains, involving both academia and software industry.

During our searches, we could not find a report on a failure case. It is possible that there is a publication bias that discourages authors to write reports on failure cases. Still, we identified challenges and presented these qualitatively and a discussion section.

There are 17 identified studies which are related to MDSE challenges. The discussion on these challenges involved issues related to software maintenance and methodology. These studies have also been categorized and summarized as part of the discussion. The studies with challenge discussion encourage new approaches to cope with the existing issues related to MDSE. In the context of methods and maintenance, it is possible that new processes, techniques, tools and developer training could mitigate both of these issues. As the recent development of new tools has taken place, in this thesis, we discuss how other approaches could be employed besides creating new tools.

This secondary study presented the review of studies from 1985 to 2018, however, studies related to MDSE challenges are scarce. Therefore, it is difficult to identify if a challenge has been dealt with in the recent years.

Moreover, we could find no evidences that the proposed solutions were in fact used. Considering the maintenance problems, we argue that these issues could rise in any project and they should be mitigated since its beginning.

After discussing the successes and challenges, we have also identified how paradigms derived from MDSE suggest that the legacy from MDSE research efforts could lead to new methods and techniques to improve software quality and adaptability in ways beyond the original proposal of code generation, as presented in Chapters 4 and 5. Further packing material and related works by the same author on this context are also reported in other documents besides this thesis[10].

[10] <http://tiny.cc/gottardi-doc>

CHAPTER 4

METHOD AND RESEARCH HISTORY

4.1 Initial Remarks

This thesis contains a set of different studies which were conducted during different moments of the research project. The purpose of this chapter is to document the research history of these studies. The motivation behind this text is to justify the presented studies. This chapter includes the general overview for both the completed and the canceled studies, therefore, it is intended that this apportioned chapter would encourage other researchers to access both the reasoning behind both successful and unsuccessful studies related to the context of this thesis.

In this manner, in Section 4.2 the research history of this project is presented. Empirical studies that were carried out prior to the main proposal and posterior studies are presented in Section 4.3. The process discovery algorithm, which was adapted from an existing algorithm, is formally defined in Section 4.4. Afterwards, a metric for process comparison which was entirely created within this project is presented in Section 4.5. This metric is analytically proven in Section 4.6 and experimentally evaluated in Section 4.7. Process analyses inspired a set of method engineering executions, as presented in Section 4.8. Finally, the conclusions for this chapter are pointed out in Section 4.9.

4.2 Study History

The first study conducted within this project was the Systematic Mapping (SM) presented in Chapter 3. This study is the basis behind all the other studies that are presented in the following chapters. Its success also encouraged several new searches to maintain it updated. The positive outcome of conducting this mapping was to confirm if the novelty value of this research topic was sufficient as expected for a thesis. Indeed, this SM affected the planning to search for new proposals for the studied problem. The preliminary conclusion of this SM suggested that there are existing challenges related to method research.

After the literature review on methods for MDSE it has been suggested that the greatest majority of existing methods and processes are inspired by MDA (ASADI; RAVAKHAH; RAMSIN, 2008; CHITFOROUSH; YAZDANDOOST; RAMSIN, 2007; ASADI; RAMSIN, 2008; NIKULSINS; NIKIFOROVA, 2008). According to the results of the SM, most MDSE tools are focused on PIM and PSM levels. High abstraction models are still recommended for software processes, including agile life-cycles which avoid the usage of models (BECK, 2000; AMBLER, 2002; WHITTLE, 2013). The lack of MDA projects that employ CIM during development could suggest two hypotheses: 1) developers do not have enough training or encouragement to fully employ MDA with CIM; 2) MDA lacks proper method for CIM.

The suggestion that the developers do not have enough training was evaluated as part of the first empirical study. In this study, it has been evaluated how difficult is to recognize each level of MDA model abstraction, i.e., CIM, PIM and PSM.

Further details on this study are presented within this chapter (Section 4.3.1). Still, it is important to report that according to its results, the participants had more trouble dealing with CIM level models. The importance of this study for the research project resulted on discussing the applicability of the CIM level.

The second empirical study was the first study planned to assess processes or methods for MDSE. In this short study, it was intended to ask developers to prioritize activities for a development process which uses MDSE since inception. This study was cancelled as related work has been identified (WHITTLE *et al.*, 2013).

There were further studies to assess hypothetical and empirical MDSE processes. The hypothetical processes were inspired by the second empirical study and processes presented as part of the theoretical foundation as presented in Chapter 2.

These empirical MDSE processes are discovered from data captured from logs collected during software development. The techniques required for process discovery were searched as part of a systematic review, which is presented in Appendix A.

One of the techniques for process discovery has been selected and adapted for executing process comparison analysis. The intent behind comparing processes was to identify an actual process that is the closest to an ideal hypothetical process. It was possible to carry on this analysis, as well as to provide formal proof for the process discovery algorithm and the comparison metric, which is a novel contribution to the state of art by itself. This algorithm, metric and proof are discussed as part of this chapter (Sections 4.4 to 4.6).

Despite the successful validation for the algorithm and metric, it was not possible to apply it to large scale development projects. In fact, even considering small projects, which were used as examples for employing the metric, it was required the author interference, because the participants could not finish all tasks by themselves. Although the algorithm and metric have been thoroughly validated, it still poses a major threat of validity for the evaluated MDSE projects for the main objective of this thesis.

Following the lack of data issue, it has been suggested to evaluate MDSE tools. The objective of this study was to identify whether tools were affecting the productivity of developers, which led them to quit using them, thus affecting the applicability of MDSE. This study was cancelled before any publication or conclusion.

Efforts on creating ideal hypothetical software processes were not cancelled besides the lack of real project data. The activities required to complete the ideal processes were further studied as part of a method engineering effort.

This method engineering effort shows that there are two fundamental types of MDSE processes: top-down and bottom-up. In top-down processes, models and code generators are defined before the coding activity starts. In the case of bottom-up, the coding activity is started prior to the code generators and possibly even before modeling. According to basic literature on code generation (HERRINGTON, 2003), the recommended technique is bottom-up since it allows the code-generator developers to learn how to generate new code based on existing code. By using method engineering principles it was planned to create an hypothetical software category which would be easier to be carried by employing the top-down principle, which is then proposed as a possible evolution for the MDSE.

4.3 Empirical Studies

This section contains two empirical studies that were conducted with the intent of identifying the difficulties that developers face when trying to employ MDSE. The first study is quantitative and involved 30 undergrad students verifying and categorizing models according to MDA levels. The second study is qualitative and involved 14 graduate students who had to develop a software using MDSE.

4.3.1 Empirical Study on MDA training difficulty

The first study was carried with 30 Information Systems undergraduate students. The training session was based on MDA concepts including CIM, PIM and PSM levels with examples. The objective was to evaluate how the participants would identify these models.

After the training, the participants received a set of models on paper and were asked to categorize these models according to the MDA levels they were trained on. These model instances were completely unknown to the participants and were in different languages than the ones shown as examples during the training session. This was planned to encourage the participants to identify their categories based only on their abstraction level or platform dependency.

There were three different models, one for each MDA level, therefore the maximum correct answers for each level would be 30 (one per participant).

All 30 participants handed in their answers, i.e. all participants completed and there were no null answers. This allowed the author to collect which answers were correct. As results, the correct answer count was 14, 20, 30 for CIM, PIM and PSM, respectively.

Despite the limitations of being a small study, it was helpful to suggest that the highest abstraction levels require more training for the participants (and developers). Besides training, there could be a lack of better methods to employ CIM models in MDA or other MDSE processes. Therefore, it has been considered how to create a set of guidelines or methods for top-down MDSE processes.

4.3.2 Empirical Study on MDSE Development Difficulty

The second empirical study is a qualitative study that was conducted with 14 graduate students. The objective of this study was to identify difficulties that arise when developers learn and attempt to build MDSE tools.

In this study, it was observed how the participants performed when developing a Web application by using two PIM metamodels as taught in the book by Brambilla, Cabot and Wimmer (2012).

The first PIM model is used to represent a web application with the content pages to be generated dinamically. The second model represents the MVC (Model-View-Controller) architectural pattern for implementing the system without platform dependency (BRAMBILLA; CABOT; WIMMER, 2012).

The participants had to adapt both metamodels to their tool environment, while also extending them with one metaclass each. It also required them to create two model transformation tools from scratch. There was a model to model transformation, which involved both model types, and a model to text transformation to create the web application code from the second model.

Since this study is qualitative, it is focused on the feedback provided by the participants on their efforts required to complete the task. All participants have cited that it was very demanding to understand model-to-model transformation written in a transformation language named as ATL (Eclipse Foundation, 2009). This language is based on functional transformation rules, which could be the cause of difficulty. Eight participants have mentioned that the first metamodel was the most difficult to understand, when compared to the second. Four participants suggested that it would be much easier to start from the final application source code and then raise the abstraction levels in a bottom-up strategy. Therefore, most participants did not agree to this suggestion, stating that starting from the highest abstraction level has advantages on design, which is beneficial for the development cycle.

This study was used to motivate the possibility of creating a more concrete technique for MDSE development based on the top-down strategy. These results encouraged further surveys on development difficulty. It is important to announce that upon identifying the work by Whittle (2013) as related, further surveys specifically on this topic were canceled. Indeed, Whittle (2013) has mentioned that it was deemed possible to apply both MDSE strategies to agile methods, however, it was not verified for other processes. After this study, more efforts on process analysis were carried out, as described within this chapter.

4.4 Process Analysis and Discovery Algorithm

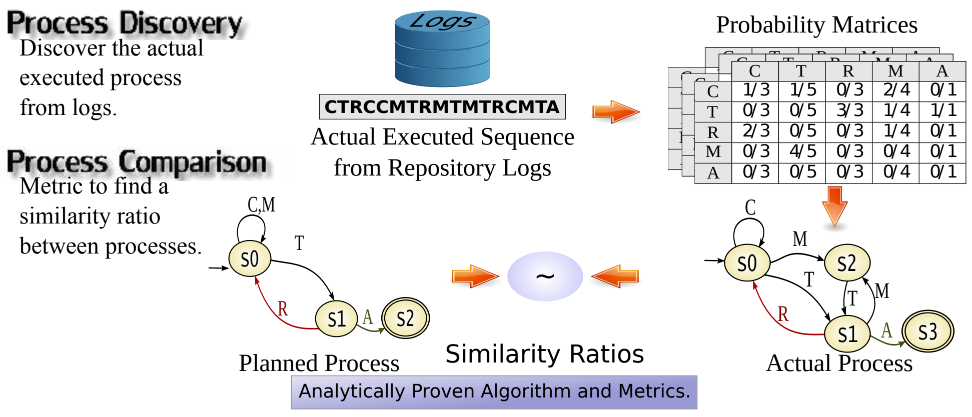
Software development projects may be successful without proper process documentation, because even if developers are not aware of the process being followed, the present development context (team, project complexity, design choices, etc) can lead to success. Therefore, in case of project success, one would not be certain whether it was a random success or thanks to a good process. Worse than that, if developers wish to repeat the process in other projects, the effort necessary to recover the past experience could prevent them from trying. To solve that, it is possible to collect quantitative data from the project execution to discover the employed process, as well as calculating metrics that are useful for its analysis (AALST, 2012; SELBY, 2009).

The lack of process awareness is particularly challenging in model-driven software engineering (MDSE). After conducting a systematic mapping in the context of MDSE processes, we have identified a few software processes defined specifically for this software development method (Chapter 3). While there are a few processes, we have not identified practical application outside the works of the authors who provided the processes. We have discussed that the developers are more focused on producing artifacts. Therefore, an approach to collect the process based on the log of produced artifacts was suggested as part of the conclusions of our previous study.

This motivated the establishment of a method for identifying and comparing processes according to their execution. Therefore, in this section, we present an algorithm based on the algorithm by Cook and Wolf (COOK; WOLF, 1998; COOK; WOLF, 1995; AALST, 2012; LAKSHMANAN; KHALAF, 2013). Their algorithm was selected after we conducted a systematic review on process discovery that indicated its adequacy for our purposes, except for two minor issues. The first is that in their method there is a threshold that would not allow precise process comparison. Also, their papers lacked enough formalism of their algorithm. Therefore, it was required to create a more precise algorithm evolved from the proposal by Cook and Wolf (1998) including the formalism enough to allow precise implementations, as well to create the comparison metric.

The algorithm presented in this thesis is used to discover process models and includes a process similarity metric that allows to identify which known process is the most similar to the one used during a certain process execution. After evaluating our algorithm, it was evidenced that it could also be applied onto several other process development comparisons.

Figure 19 – Process Discovery and Metric Graphic


Process Discovery
Discover the actual executed process from logs.
Process Comparison
Metric to find a similarity ratio between processes.
Logs
CTRCCMTRMTMTRCMTA
Actual Executed Sequence from Repository Logs
Probability Matrices
C,M
T
R
A
S0
S1
S2
S3
M
~
Planned Process
Similarity Ratios
Actual Process
Analytically Proven Algorithm and Metrics.

	C	T	R	M	A
C	1/3	1/5	0/3	2/4	0/1
T	0/3	0/5	3/3	1/4	1/1
R	2/3	0/5	0/3	1/4	0/1
M	0/3	4/5	0/3	0/4	0/1
A	0/3	0/5	0/3	0/4	0/1

Source: Created by the author

Figure 19 contains a graphic that illustrates the overall execution of the process discovery algorithm: Logs from artifact repositories are collected; Changes to artifacts must be manually assigned to expected activities of the executed process; These activities are processed into probability matrices; These matrices are used to discover the actual process (described in Subsection 4.5.3). In the case of the availability of more than one log, it is possible to capture different sets of matrices. A comparison metric was created (described in Subsection 4.5.4) that results into a similarity ratio when pairs of sets (of matrices) are used as input.

4.5 Algorithm Definition

In this section, we describe the algorithm we have devised to discover processes, which also includes a process comparison metric. The algorithm is based on logs of activities performed during the software process, and it stands on probabilities of occurrence of activities in a certain sequence. This log can be collected manually or automatically by development tools, however the phase of collection is out of the scope of this thesis, and the algorithm is independent of how this is done. Therefore, despite not being the main goal of this section, this algorithm can be implemented to allow automated process discovery.

The algorithm is composed by two steps, which are described within this section: (1) Gather First Order Probabilities; (2) Gather Second Order Probabilities. For both steps, the input is a sequence log of activities, and the outputs are two matrices that can be used to generate a process model automata and other metrics.

The log sequence may be captured from software repositories. The possible activity types must be predicted (manually) prior to the discovery execution. In the case of validation studies presented in this chapter, we have captured the creation and edition of different artifact types, i.e., source code edition for coding, execution logs for testing and approval, etc.

For a simple hypothetical example, we consider the following log sequence: L =

$\boxed{\text{CTRCCMTRMTMTRCMTA}}$. The set of the activity types is referred within this section as D. There are $N = 5$ different activity types and each letter represents one activity execution (highlighted in bold letter): **C**ode; **T**est; **R**eject; **M**odify; **A**pprove.

4.5.1 First Order Probabilities

The first step of the algorithm is to gather first order probabilities. The input of this step is the activity stream and the output is a two-dimensional matrix named F. This matrix is used to represent the probability of the previous state of a given current state, i.e. each cell indexed by $[i,j]$ contains the probability of event i to be followed by event j. Then, we count the instance of digrams (event pairs) and divide them by the total count of digrams that end on the current state. This way we can define how likely was a transition to come from specific previous states. The scope for this calculation involves $C = 2$ consecutive states. For the first order, C is always 2.

In order to count the number of digrams, we define a sum of ones and zeros. Therefore, it is necessary to define the instance count function $I_F(z,x)$ (Equation 4.1), where z and x are activities within D, i.e. $z \in D \wedge x \in D$; the length of the log stream is M; and the result of the function is a natural number.

$$I_F(z,x) = \sum_{i=1}^{M-1} \left(\begin{array}{ll} 1 & \text{if } (L_i = z) \wedge (L_{i+1} = x), \\ 0 & \text{otherwise.} \end{array} \right). \tag{4.1}$$

The total number may be computed to be zero. Since it is the denominator of any matrix cell, it is necessary to avoid it from being zero by using the coalesce function R (Equation 4.2) that replaces the zero by any other non zero number. For simplicity, we replace the zeros by ones, therefore whenever there is no instance count for a given current state, the complete total probability for each column would be always either zero or one.

$$R(z,x) = \left(\begin{array}{ll} z & \text{if } (z \neq 0), \\ x & \text{if } (z = 0). \end{array} \right). \tag{4.2}$$

It is then possible to calculate the first order probability matrix F (Equation 4.4) that has C dimensions of N elements, i.e. NxN. Each cell at coordinates "i, j" is calculated by the function $F_F(i, j)$ (Equation 4.3) The N different activities may be represented by using a one-dimensional array D that has N elements.

$$F_F(i,j) = \frac{I_F(D_i,D_j)}{R(\sum_{i=1}^{N} I_F(D_i,D_j),1)} \tag{4.3}$$

$$F = \begin{array}{c} \\ D_1 \\ D_2 \\ \vdots \\ D_N \end{array} \begin{array}{c} \begin{array}{cccc} D_1 & D_2 & \cdots & D_N \end{array} \\ \begin{pmatrix} F_F(1,1) & F_F(1,2) & \cdots & F_F(1,N) \\ F_F(2,1) & F_F(2,2) & \cdots & F_F(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ F_F(N,1) & F_F(N,2) & \cdots & F_F(N,N) \end{pmatrix} \end{array} \tag{4.4}$$

Considering the provided log example (referred as L), vector D would be [C T R M A]. The sum within function F takes $M-1$ values. So, if we consider the first three letters of the provided example, $\boxed{\text{CTR}}$, we would have 2 instances: from C to T and then from T to R. The resulting matrix for this length-three excerpt would allow to represent 1/1 probability to reach T from C, and 1/1 probability to reach R from T. In this section we also present the resulting F matrix for L = $\boxed{\text{CTRCCMTRMTMTRCMTA}}$ as a whole. The F Matrix is composed by the first order probabilities of the process transitions (Equation 4.5).

$$F = \begin{array}{c|ccccc} & C & T & R & M & A \\ C & 1/3 & 1/5 & 0/3 & 2/4 & 0/1 \\ T & 0/3 & 0/5 & 3/3 & 1/4 & 1/1 \\ R & 2/3 & 0/5 & 0/3 & 1/4 & 0/1 \\ M & 0/3 & 4/5 & 0/3 & 0/4 & 0/1 \\ A & 0/3 & 0/5 & 0/3 & 0/4 & 0/1 \end{array} \tag{4.5}$$

According to the data represented in this matrix, for example, it is possible to identify the previous letters that have transitions to C. Then, to reach C, there is a $^1/_3$ probability that the previous letter was another C, and a $^2/_3$ probability that the previous letter was R.

4.5.2 Second Order Probabilities

The second step of the algorithm is to gather second order probabilities. The scope for this calculation involves three consecutive states. This step takes as input the same activity stream of the previous step and outputs a three-dimensional matrix named S. Similarly to the previous step, this matrix is also used to represent the probability of previous states that reach the current one, however, it takes three states, the current and two previous ones. Therefore, we count the instance of trigrams and divide them by the total count of trigrams that come from a precursor state, then transition to a previous state, and then end on the current state. Following this rule, the scope for this calculation involves $C = 3$ consecutive states and for the second order, C is always 3. The instance count function for the second order probabilities is also defined as a sum of zeros and ones. As represented by $I_S(z,x,y)$ (Equation 4.6), there are three activity letters.

This function results to a natural number.

$$I_S(z,x,y) = \sum_{i=1}^{M-2} \left(\begin{array}{ll} 1 & \text{if } (L_i = z) \land (L_{i+1} = x) \land (L_{i+2} = y), \\ 0 & \text{otherwise.} \end{array} \right). \tag{4.6}$$

Since the total instances per column can be also computed to zero, the function R is also used to replace zero denominator values to one. This is the same R function described in Subsection 4.5.1.

For calculating the second order probability matrix S that has C dimensions of N elements, i.e. $NxNxN$, each cell at coordinates "i, j, k" is calculated by function $F_S(i, j, k)$ (Equation 4.7). The same vector of the N different activities is represented by using a one-dimensional array D that has N elements. Since Matrix S is three-dimensional, it could be also represented as an array of N NxN matrices. Then, there would be matrices for S_1 to S_N (Equation 4.8).

$$F_S(i, j, k) = \frac{I_S(D_i, D_j, D_k)}{R(\sum_{i=1}^{N} I_S(D_i, D_j, D_k), 1)}. \tag{4.7}$$

$$S_N = \begin{array}{c} \\ D_1 \\ D_2 \\ \vdots \\ D_N \end{array} \begin{array}{c} \begin{array}{cccc} D_1 & D_2 & \cdots & D_N \end{array} \\ \begin{pmatrix} F_S(N,1,1) & F_S(N,1,2) & \cdots & F_S(N,1,N) \\ F_S(N,2,1) & F_S(N,2,2) & \cdots & F_S(N,2,N) \\ \vdots & \vdots & \ddots & \vdots \\ F_S(N,N,1) & F_S(N,N,2) & \cdots & F_S(N,N,N) \end{pmatrix} \end{array}. \tag{4.8}$$

For the first four letters of the provided example, $\boxed{\text{CTRC}}$, we would have 2 instances, since the sum within function S takes $M - 2$ values. These instances would be from C through T to R and from T through R to C. The resulting matrix for this length- four excerpt would allow to represent 1/1 probability to reach R from C through T and 1/1 probability to reach C from T through R.

We also provide the resulting second order Matrix S. However, since it is not simple to represent this matrix on paper due to its three dimensions, it has been transformed into Table 9. The first column contains the pair of initial state transitions. Each sequence of five rows represents one dimension of the original Matrix S, hence the separator lines.

4.5.3 Automata Generation

The resulting process model is represented by an automata, which is illustrated in Figure 20. Firstly, a non-deterministic finite automata is generated. In order to generate this automata, the values from the second order probabilities matrix are taken according to the following rules:

- For each non zero value from matrix S:

Table 9 – Calculated Second Order Example

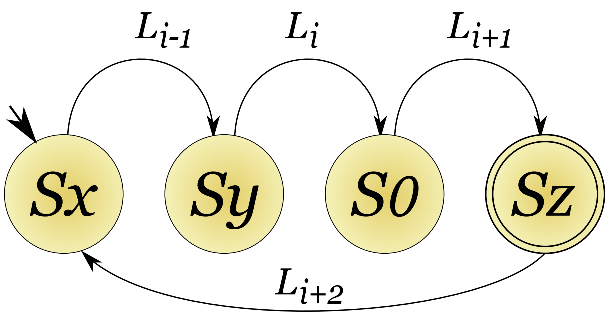
	C	T	R	M	A
CC	0/3	0/4	0/3	1/4	0/1
CT	0/3	0/4	1/3	0/4	0/1
CR	0/3	0/4	0/3	0/4	0/1
CM	0/3	2/4	0/3	0/4	0/1
CA	0/3	0/4	0/3	0/4	0/1
TC	0/3	0/4	0/3	0/4	0/1
TT	0/3	0/4	0/3	0/4	0/1
TR	2/3	0/4	0/3	1/4	0/1
TM	0/3	1/4	0/3	0/4	0/1
TA	0/3	0/4	0/3	0/4	0/1
RC	1/3	0/4	0/3	1/4	0/1
RT	0/3	0/4	0/3	0/4	0/1
RR	0/3	0/4	0/3	0/4	0/1
RM	0/3	1/4	0/3	0/4	0/1
RA	0/3	0/4	0/3	0/4	0/1
MC	0/3	0/4	0/3	0/4	0/1
MT	0/3	0/4	2/3	1/4	1/1
MR	0/3	0/4	0/3	0/4	0/1
MM	0/3	0/4	0/3	0/4	0/1
MA	0/3	0/4	0/3	0/4	0/1
AC	0/3	0/4	0/3	0/4	0/1
AT	0/3	0/4	0/3	0/4	0/1
AR	0/3	0/4	0/3	0/4	0/1
AM	0/3	0/4	0/3	0/4	0/1
AA	0/3	0/4	0/3	0/4	0/1

- There are four states: Sx, Sy, S0 and Sz.
- There are four transitions: L_{i-1}, L_i, L_{i+1}, L_{i+2};
- T1 links Sx towards Sy;
- T2 links Sy towards S0;
- T3 links S0 towards Sz;
- T4 links Sz towards Sx.

The start state is any Sx that is linked to a transition taking the first letter of vector L. The final state is any Sz that was generated by the last trigram of L. The union of this automata represents the complete automata, which may be transformed into a simpler deterministic automata. This simplification algorithm is not within the scope of this thesis.

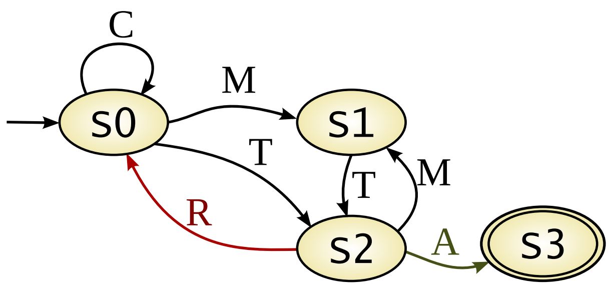
Still, this section also includes the resulting simplified automaton constructed according to the algorithm for the process based on the provided log in Figure 21.

Figure 20 – Non-deterministic Automaton Instance



Source: Created by the author

Figure 21 – Generated Automaton for the Example



Source: Created by the author

4.5.4 Comparison Metric

In addition to the automata generation, the main goal of this section is to present a metric defined for process execution similarity detection. The idea is to count how many matrix cell values that are greater than zero occur in both matrices at the same coordinates, plus how many zero values occur in both matrices at the same coordinates. By dividing this count by the total number of cells, the quotient would be the ratio of matrix similarity. Therefore, we employ the first and second order matrices, namely F (Equation 4.4) and S (Equation 4.8).

Since it is necessary to compare two first order matrices and two second order matrices, the matrices are also defined as parameters for these functions, thus, avoiding ambiguity. The complete pair of functions are defined as Equations 9 and 10. These ratio can be used separately or as a final product, which takes of both first order and second order matrix similarities, i.e. $C_2(F,N) \cdot C_3(S,N)$. These functions were defined by using the exclusive-or operator ($\oplus$), which results in true when the pair of Boolean values are different. If the values are different, zero is added to the sum, otherwise one is added to increase the similarity count. Since the sums take zeros and ones as parameters, the maximum case would be summing up N^2 or N^3 ones. Therefore, their maximum value for the sums would be exactly count of cells taken as comparison. Since these sums are multiplied by $\frac{1}{N^2}$ and $\frac{1}{N^3}$, respectively, this result would be a rational between zero and one (inclusive). This metric follows the generalized metric property of always resulting in zero for the most similar pair (ARKHANGEL'SKIĬ; FEDORCHUK, 1990). Therefore, in this case, one would represent the most different pair.

$$C_2(F,G,N) = \frac{1}{N^2} \cdot \sum_{i=1}^{M} (\sum_{j=1}^{M} (\begin{cases} 1 & \text{if } (S_{i,j} > 0) \oplus (G_{i,j} > 0), \\ 0 & \text{otherwise.} \end{cases})). \tag{4.9}$$

$$C_3(S,R,N) = \frac{1}{N^3} \cdot \sum_{i=1}^{M} (\sum_{j=1}^{M} (\sum_{k=1}^{M} (\begin{cases} 1 & \text{if } (S_{i,j,k} > 0) \oplus (G_{i,j,k} > 0), \\ 0 & \text{otherwise.} \end{cases}))). \tag{4.10}$$

4.5.5 Related Algorithms

This section includes the definition of an algorithm for process discovery and process comparison metric. As mentioned before, the work of Cook & Wolf provided guidelines for the definition of the discovery algorithm provided herein (COOK; WOLF, 1998; COOK; WOLF, 1995). Therefore, our algorithm is closely related to theirs, however, their algorithm includes a threshold steps but lacks a formal proof and definition. In the case of their algorithm, a threshold step was added to ignore low probability transitions to further increase flexibility. Since we planned to create a formal algorithm that also includes metric for comparison, being excessively flexible would decrease the precision of the metric.

Our work is also focused on empirical software engineering in the context of process models, which is similar to the works by Selby *et al.* (SELBY *et al.*, 1991b; SELBY, 2009), who have worked on a set of metrics for empirical analysis of software development. It involves a tree categorization metric (SELBY *et al.*, 1991a) and a scripting configurable capture system that calculates metrics on demand (SELBY *et al.*, 1991b). They have also worked on a dynamic metric capture system that allows a software development manager to watch several metrics collected from software development projects (SELBY, 2009).

Even though this work has followed the results of a Systematic review on software process discovery, presented within Appendix A, it is still worth mentioning related surveys and other secondary studies on this subject.

The work by Akman and Demirörs describes the application of four process discovery algorithms (AKMAN; DEMIRöRS, 2009), namely Markov Method, Heuristic Mining, Fuzzy Mining and Genetic Process Mining.

Other works related to process discovery algorithms have been presented in the literature. For instance, Lakshmanan and Khalaf, and Der Aalst *et al.*, have written secondary studies where they present a set of different algorithms and their advantages (AALST *et al.*, 2012; LAKSHMANAN; KHALAF, 2013). We recommend them as a solid literature for readers willing to understand the basics of process discovery.

4.6 Analytic Metric Proof

In this section, we present an analytic proof that the proposed metric is a generalized metric, i.e., it complies to the properties expected for a generalized metric according to mathematical theory. This proof is also presented as validation that the proposed metric can be used to identify the process differences and similarities. As discussed in Section 4.5, there are constraints for input variables. This does not restrict to compare different processes, it is simply required to compute the union of activities prior to the metric calculation. These restrictions are presented as part of propositions.

The propositions are based on the definition presented on Section 4.5 and they are always valid for the proposed metric: 1) The metric takes matrices as input, which are generated by the algorithm defined within Section 4.5; 2) N is a natural number which is equal to the size of set D (activity type count); 3) The matrices must be NxN or NxNxN according to their order; 4) The compared matrices must have the same dimensions.

As another proposition, it is also important to establish the truth table for exclusive-or operation as presented on Table 10. This operation is used in the metric function to sum zeros and ones for equal or different attributes, respectively.

Table 10 – Truth Table for Exclusive-Or

p	q	$\oplus$	expression	summed value
false	false	false	$(p \vee q)$	0
false	true	true	$(\neg p \wedge q)$	1
true	false	true	$(p \wedge \neg q)$	1
true	true	false	$(\neg p \vee \neg q)$	0

The logic (Boolean) expression for Exclusive-Or can be expressed in both ways, either by considering the false or the true results (MENDELSON, 1987), therefore, we have decided to take the expression from the true results, as expressed in (Equation 4.11), which is employed for the proof presented within the following sections.

$$p \oplus q = (\neg p \land q) \lor (p \land \neg q) \tag{4.11}$$

4.6.1 Properties and Generalized Metric Categories

This subsection includes the properties expected for the function to be categorized as a metric or generalized metric. The properties are enumerated starting on 1 up to 4. A metric must comply to all presented rules. Generalized metrics are categorized according to the subset of properties of their compliance (ARKHANGEL'SKIĬ; FEDORCHUK, 1990).

1. Non-negativity:
   $m(x,y) \geq 0$, for any x and y;
2. Identity of Indiscernible:
   a) $m(x,y) = 0$, if and only if $x = y$;
   b) $x = y$, if $m(x,y) = 0$;
   c) $m(x,y) = 0$, if $x = y$;
3. Symmetry:
   $m(x,y) = m(y,x)$;
4. Triangle Inequality:
   $m(x,z) \leq m(x,y) + m(y,z)$.

According to the non-negativity property (Property 1), the results of the function must be always positive for any valid pair of input parameters. This property is justified for all categories of generalized metrics because metrics must represent a distance between the parameters, which cannot be negative (ARKHANGEL'SKIĬ; FEDORCHUK, 1990).

The Identity of Indiscernible (Property 2) refers to the ability to identify equal input parameters by using the metric. There are three levels of this property. The first level (Property 2.a) is the most complete and restrictive, and also implies both other levels (i.e., 2.a implies 2.b $\land$ 2.c). According to the first level, whenever the metric results in zero, then, the parameters are equal.

In addition, having identical parameters is the only possible way for the metric to result in zero. The other levels relax one of these statements. Therefore, according to the second level (2.b), by having identical parameters, the metric must result in zero, however, there might be other zero results for different parameters. Finally, according to the third and last level (2.c), if the metric results in zero, then the parameters are identical. However, it might be possible that some identical parameters result in non-zero values.

The Symmetry (Property 3) establishes that the result of the metric is not affected by the order of the input parameters. This means that comparing x against y yields the same result as comparing y against x.

The Triangle Inequality (Property 4) statement is a property which establishes the difference when comparing parameters by taking a third element as an intermediate. Given the input parameters x, y, z, the comparison of x against z would never be greater than comparing x against y plus comparing y against z.

As mentioned, a metric must comply to every property as a rule, however, the generalized metric categories have relaxed rules. For all categories, the first property (Property 1) is always a rule. Therefore, they only vary on the applicability of the other properties. As presented on Table 11, we discuss the applicability of the following generalized metric types: A quasimetric must comply to Properties 1, 2.a, 2.b, 2.c and 4. A pseudometric must comply to Properties 1, 2.c, 3 and 4. A prametric must comply to Properties 1, 2.c and 3. A semimetric must comply to Properties 1, 2.a, 2.b, 2.c and 3. A metametric must comply to Properties 1, 2.b, 3 and 4.

Table 11 – Metric and Generalized Metrics Categorization

Category	Property					
Name	1	2.a	2.b	2.c	3	4
metric	true	true	true	true	true	true
quasimetric	true	true	true	true	false	true
pseudometric	true	false	false	true	true	true
prametric	true	false	false	true	true	false
semimetric	true	true	true	true	true	false
metametric	true	false	true	false	true	true

4.6.2 Proof for Properties

This subsection contains proofs for the proposed functions to establish which generalized metric categories are applicable. It is important to advise that some of these proofs are only applicable to the matrix inputs of the functions defined in Equations 9 and 10.

4.6.2.1 Non-negativity

Non-negativity (Property 1) is required for all the presented metric categories. In order to prove that the proposed metric function complies to this rule, we must analyze the formula. It includes multiplication of positive real numbers and a sum of Exclusive-Or expressions.

For all input parameters of the metric functions (Expressions 8 and 9), the resulting exclusive-or operations would result in a sequence of zeros and ones to be summed and multiplied by a positive rational number $\frac{1}{N^2}$ or $\frac{1}{N^3}$. Therefore, the function is never negative. Zero values are possible when all exclusive-or operations result in zero, any other value is positive.

4.6.2.2 Identity of Indiscernible

The Identity of Indiscernible (Property 2) is divided into three levels, where the first level (2.a) is the most complete and restrictive, and also implies both other levels (i.e. 2.b $\wedge$ 2.c). Therefore, proof must be written for Levels 2.b and/or 2.c.

Property 2.b establishes that if the result is zero, then both parameters must be equal, while property 2.c establishes that whenever there are identical input parameters, the result must be zero.

This can be proven if we consider each Expression (9 and 10) independently and also assume that the input parameters are simply the matrices, i.e. we are not establishing that the process executions are equal, but their generated matrices.

Proof for this property can be established as follows: Let A_p be the first process that generates matrix A. Let B_p be the second process that generates matrix B. It is possible to prove that the functions identify when A equals B if and only if the result is zero.

The important step of this analysis is to identify the only possible case when the sum equals to zero. Therefore, according to the definition of the functions (Expressions 9 and 10), there is a positive rational number $\frac{1}{N^2}$ or $\frac{1}{N^3}$ which multiply the sum of exclusive-or expressions. The exclusive-or expressions always result in either zero or one, as defined by the "sum column" of Table 10. The only possible way to attain a zero result is to calculate the sum of zeros. The presence a single one, which indicates a different cell in the matrices, would lead to a non zero result.

4.6.2.3 Symmetry

The Symmetry is Property 3 and indicates that the input parameters always result in the same value regardless of the order in which they are provided to the function. This is similar to the closure property present in "addition" (arithmetic operation) (MENDELSON, 1987).

In order to prove that the proposed functions comply to the symmetry property, it is required to analyze the usage of the input variables within the functions (Expressions 9 and 10).

The only operation that processes the input parameters is the Exclusive-Or inside the sums. Therefore, the symmetry would be observed if this operation complies the closure property. However, Exclusive-Or closure property is not provided as an axiom.

In order to prove closure property for Exclusive-or, it is simply required to assume that the input variables can be swapped. Then, check if the equality is preserved. Following axiom that operations disjunction ($\vee$) and conjunction ($\wedge$) comply to closure property (MENDELSON, 1987) and according to Equation 4.11, which is the expression for the Exclusive-Or, it is possible to prove the closure with the deduction presented in Equation 12.

$$\begin{gathered}
(p \oplus q) = (q \oplus p); \\
(\neg p \wedge q) \vee (p \wedge \neg q) = (\neg q \wedge p) \vee (q \wedge \neg p); \\
disjunction\ closure: \\
(\neg p \wedge q) \vee (p \wedge \neg q) = (q \wedge \neg p) \vee (\neg q \wedge p); \\
conjunction\ closure: \\
(\neg p \wedge q) \vee (p \wedge \neg q) = (\neg p \wedge q) \vee (p \wedge \neg q);
\end{gathered} \tag{4.12}$$

Therefore, by swapping the order of operators inside the functions presented in Equations 9 and 10, they would still be bound to the closure of Exclusive-Or operation, thus proving that the function is symmetric.

4.6.2.4 Triangle Inequality

The Triangle Inequality is Property 4, which states that a direct comparison between two parameters cannot be greater than the sum of comparing each one of these two parameters against a third parameter.

Starting from the basic corollary that the $\leq$ (less-equal) comparison is the same as proving they can be true for $=$ (equal) $\vee$ $<$ (less than), it is possible to prove intermediary statements as lemmas.

Lemma 1: It is important to analyze whether the sum of two results from the functions are always greater or equal to each of the operands. i.e. $(x \leq (x+y)) \wedge (y \leq (x+y)) | (x \in \mathbb{R}) \wedge (y \in \mathbb{R})$

Taking the proof for Property (1), it is already established that the functions results are never negative. The functions are composed by a sum of zeros and ones from an Exclusive-Or expression, property kept after multiplying by positive rational numbers.

Following this property: this lemma is valid since the sum of non-negative real numbers (or more precisely, rationals) is always equal or greater than each of the original numbers, i.e. non-negativity also stands for the sum of results. This can be shown according to arithmetic axioms (MENDELSON, 1987).

Lemma 2: The goal is to analyze when the sum results to an equal number. Following the first lemma, the sum of results can be further discussed if we consider the proof of Property 2. The only possible way to result in zeros require a pair of equal matrices. Sum of zeros is a neutral element which does not impact on the result. Therefore, if the sum of a pair of function results is equal to another result, there must be a zero result among the function results. This covers the case where the function is equal, therefore, it is still required to prove the existence of non-equal function results.

Final Proof: Following the previous lemmas, the goal is to prove that the result is either equal or greater than the sum of other results on a triangular comparison. Let the triangular inequality comparison be expressed as $t(p,q,r) = (m(p,r) \leq m(p,q) + m(q,r))$. This is a Boolean expression that must result in true at all times if the Triangular Inequality (Property 4) is valid for the function.

Table 12 – Truth Table for Sums of Three Exclusive-Or Operators

p	q	r	$p \oplus r$	$p \oplus q$	$q \oplus r$	$p \oplus r \leq p \oplus q + q \oplus r$
0	0	0	0	0	0	true (equal)
0	0	1	0	0	1	true (less)
0	1	0	0	1	1	true (less)
0	1	1	1	1	0	true (equal)
1	0	0	1	1	0	true (equal)
1	0	1	0	1	1	true (less)
1	1	0	1	0	1	true (equal)
1	1	1	0	0	0	true (equal)

In order to prove that $t(p,q,r) = (m(p,r) \leq m(p,q) + m(q,r))$ is always true, we must analyze the outcomes for the Exclusive-or operator when comparing the constant matrix B to both A and C. A truth table for these operations is presented on Table 4, which contains all possible values for the p,q and r variables to be summed and compared according to the Boolean function $t(p,q,r)$, i.e. if the statement is true for the functions, then, every possible value for the comparison of $p \oplus r \leq p \oplus q + q \oplus r$ must be true. This can be seen on Table 4, hence proving the statement for the proposed functions.

4.6.2.5 Proof Discussion

Considering the input matrices to be compared by the proposed function and by proving the four expected properties, one could claim that the proposed function complies to all properties required for metrics. Also, it does not need to be considered as one of the generalized metrics, which are relaxed versions.

However, we can only claim that the function is a metric from the scope of the input parameters as matrices, i.e., after the initial algorithm of matrix creation is executed.

If we consider the input parameters as the original process executions which generate the matrices, it would not be enough to prove the Property 2 (Identity of Indiscernible) completely. Still, it could be possible that the property is still valid for the scope of processes as inputs, which should be treated as future works.

Considering this issue, the function would still fit into generalized metric categories that do not require Properties 2.a and 2.b, namely, prametric and pseudometric (ARKHANGEL'SKIĬ; FEDORCHUK, 1990).

In the scope of our work, however, this is not a problem, since this matrix creation allows more flexibility when comparing processes. Therefore, whenever the number of the interactions inside the matrix changes but the valid transitions are the same, we consider these process executions as variations of the same type of process.

4.7 Metric Assessment and Practical Experiment

The proposed method was used in a quasi-experiment to verify that it is applicable to actual project logs. This quasi-experiment, herein referred as a study, was planned according to the recommendations by Wohlin *et al.* (2012). This study is referred as quasi-experimental since it was not possible to gather a large set of randomly assigned participants, i.e., they were manually assigned with the help of the researcher. This study does not affect the formal validity of the metric itself since it has been already validated analytically.

4.7.1 Study Planning and Definition

The objective of this study was to demonstrate the accuracy of the proposed metric system in actual logs to evidence similarities between process executions.

Since our research group is focused on identifying processes for Model-Driven Software Engineering (MDSE) (FRANCE; RUMPE, 2007), the study object was a software project that employs that methodology.

Research Question: The study was planned considering the following research question: "Is the proposed metric appropriate to identify the most similar executed process to an expected process?".

Context selection: We asked three graduate students to create software systems employing MDSE, while data was collected from the produced software.

Formulation of hypotheses: The hypotheses considered for the planned study are shown on Table 13 There are two types of Hypotheses: the null hypothesis represents the inconclusive outcome of the study, while the last two hypotheses are the alternate, which represent the conclusive outcome of the study.

Variable selection: The independent variable is a high level expected process definition. The participants were recommended to follow the provided process sequence which was not strictly enforced. The dependent variable to be considered in this study is the coincidence count between second order probabilities matrices, which is, in turn, calculated from the executed

Table 13 – Hypotheses for the study

Symbol	Name	Description
h_0	Null	There is no evidence that the metric is accurate to identify the similar processes (metric has no effect);
h_n	Negative	There is an evidence that the metric results into a greater than average value for similar processes (metric is incorrect).
h_p	Positive	There is an evidence that the metric results into a less than average value for similar processes (metric is correct).

processes. The idea is to identify whether this metric can be employed to determine the projects that execute similar processes.

Participant selection criteria: We have invited graduate students that have completed a course in software reuse, which included model-driven engineering topics. Despite the fact that it is not possible to assure that the group of students represent the population of developers it is important to remind that the object of study is not dependent on their performance. It was only conducted following this design to provide a more realistic view than using data from simulations. Therefore, the author collaborated with the students to create software systems to be used in the analysis. In this manner, the study subjects in this quasi-experiment are the produced software products.

Design of the study: This study has been designed to be composed by five different software systems written by three students according to expected processes: The first process is top-down, starting from the highest level artifacts to the lowest level i.e, design models. The second process is bottom-up, thus, developers initiate the cycle by writing the application source code before creating design models. The third process is a traditional code-only process without modeling artifacts.

It was meant to verify the metric system by comparing the produced software systems. Then, if the metric is accurate, the projects that employ the same process should result into a smaller metric result.

Instrumentation for the study: The participants were provided a requirements document for a small contact list, with the following requirements: (R1) ability to store a contact list; (R2) generated entity type, e.g. Person; (R3) entities have generated attribute types, e.g. telephone numbers and addresses; (R4) Every entity and attribute has a name; (R5) Three use cases: search entities by name, insert entities to contact list, and list all the stored entities. It is important to notice that requirements R3 and R4 contain variation points, i.e. despite the fact that they are exemplified, it is also expected that a generative programming approach allows them to be easily modified.

The processes were also provided in a high level description, i.e., they were not strictly enforced, to allow more freedom to the participants since we intended to avoid exact replications of process executions.

The provided processes, namely, code-only, bottom-up, top-down were also described. Code-only is a traditional process solely focused on writing source-code and then testing. The bottom-up process must start from coding based on the provided examples of the requirements document (in this case, the examples include the “Person” and “telephone number” suggestions). Once the application was functional, a meta-model should be defined for the variation points (in this case, the lists for entities and attribute types). By following the planned MDSE project requirements, the participant should define a model based on the provided examples and write a code generator, which should be capable of generating the same application from this model, and variant applications from a modified model.

The top-down process must start from meta-model definition and then the creation of a model instance based on the suggested examples. The participant should then code the application based on the model instance alongside with the code generator, which should generate the intended application as well as variant applications, exactly as the outcome of the bottom-up process.

4.7.2 Study Operation

During the study, the participants had to develop the software application according to the provided requirements and process. The requirements were fixed throughout the study, while the process is an independent variable that was under treatment.

Execution: The first execution was performed by using the top-down process and the next executions were carried out by using the bottom-up process.

Data Validation: Every resulting software application had to be tested and verified against the fixed requirements. This way, we can assure that the resulting applications are tested and performed as intended. Since the data is collected from a version control system containing all changes since the inception of the applications until their final test and verification, we could consider this data as valid.

Data Collection: The recorded logs were converted into strings of letters (symbols). Each letter represents one activity of the executed process. These processes have different activity orders, however, they still share the same activity types, listed on Table 14.

The expected logs for each process and the collected data are listed on Table 15. The expected logs (process templates used as means of comparison) are L_t, L_b and L_c (for the top-down, bottom-up, and code-only methods, respectively). These logs were defined according to the processes presented as instrumentation. The participants collaborated with the author during the executions to create the software instances to be analyzed in the study. The logs L_t1, L_b1 and L_b4 were collected from software instances developed by the students collaborating with the author, while the other logs were collected from software instances provided by the

Table 14 – Collected Activity Types

Symbol	Name	Description
M	Metamodel Definition	Metamodel definition and modification;
V	Metamodel Validation	Metamodel validation and verification;
K	Metamodel Approval	Metamodel approval for model creation;
I	Model Instance	Model definition and modification;
C	Coding	Application source-code definition and modification;
T	Test	Application functional testing;
A	Application Approval	Application approval after verification of requirements;
G	Generator	Code generator definition and modification;
E	Generator Execution	Code generation and generator script testing;
F	Finish	End of life cycle after confirming that the generated code with variant models comply to the requirements.

Table 15 – Expected and Collected Activity Logs

Symbol	Name	Description
L_t	Expected Top-Down	MVKIGECTAF
L_b	Expected Bottom-Up	CTAMVKIGEF
L_c	Expected Code-Only	CTAF
L_t1	First Top-Down	MVMVKIGECTCTCTAGEGF
L_t2	Second Top-Down	MVMVKIGECTGEGEF
L_t3	Third Top-Down	MVKGIGIGECTCTGEF
L_t4	Fourth Top-Down	MVMVMVKIGIGECTGEF
L_t5	Fifth Top-Down	MVKIGEGECTAF
L_b1	First Bottom-Up	CTCTCTAMVKIGIGEGF
L_b2	Second Bottom-Up	CTCTAMVMKIGEGEF
L_b3	Third Bottom-Up	CTCTAMVKIGIGEGEF
L_b4	Fourth Bottom-Up	CTCTCTAMVKIGEF
L_b5	Fifth Bottom-Up	CTAMVMVKGIGEF
L_c1	First Code-Only	CTCTCTAF
L_c2	Second Code-Only	CTCTCTCTCTAF
L_c3	Third Code-Only	CTCTCTAF
L_c4	Fourth Code-Only	CTCTCTCTAF
L_c5	Fifth Code-Only	CTCTAF

author.

These logs are then used to calculate the metric of similarity ratios, which are presented on Table 16. This table contains columns for the similarity ratios calculated from the first order probabilities (First), the second order probabilities (Second) and the result of multiplying these (Product).

Table 16 – Calculated Similarity Ratios

Sequences		First	Second	Product
	L_t5	1/100.0	2/1000.0	2/100000
L_t	L_b5	5/100.0	8/1000.0	40/100000
	L_t1	6/100.0	9/1000.0	54/100000
	L_t2	6/100.0	9/1000.0	54/100000
	L_t4	6/100.0	9/1000.0	54/100000
	L_b5	1/100.0	2/1000.0	2/100000
L_b	L_b4	3/100.0	6/1000.0	18/100000
	L_b3	4/100.0	7/1000.0	28/100000
	L_b1	5/100.0	7/1000.0	35/100000
	L_t5	5/100.0	8/1000.0	40/100000
	L_c1	1/100.0	2/1000.0	2/100000
L_c	L_c2	1/100.0	2/1000.0	2/100000
	L_c3	1/100.0	2/1000.0	2/100000
	L_c4	1/100.0	2/1000.0	2/100000
	L_c5	1/100.0	2/1000.0	2/100000

4.7.3 Data Analysis and Interpretation

The results presented on Table 16 are grouped for each expected process. Each group is also ordered (ascendant), to show the top five (smallest) most similar executions according to the metric.

It is possible to identify that the most similar executions were based on the expected software process. Considering each each group from Table 8, every first result has been confirmed to follow the expected process. The only exceptions (also referred as outliers) to the results are L_b5 and L_t5, which appear to be similar to L_t and L_b respectively, which is contrary to the original expectation. Therefore, it is required to execute a statistical test to establish if these outliers do not invalidate the intended confidence ratio for the metric.

4.7.4 Results Discussion

In this subsection, the goal is evaluate the applicability of the metric to measure the captured project logs. This applicability was measured with a confidence level by using a statistical method. The result is also discussed. The returned results were compared against the expected values by using a t-test method, as represented on Table 17.

The values to compare in this test are the resulting products for the executions which are expected to follow L_t and L_b. Therefore, we intend to compare the difference for the metric result when comparing the execution to the intended expected process minus the incorrect process. This result would be positive if the metric indicates the correct value, negative for wrong values and zero if the metric has no effect on comparison.

This table is composed by six columns: a pair of *mean* columns represent the calculated mean for each side of the paired sample (namely, correct and wrong values), $d.f.$ stands for degrees of freedom, t is the variable used to identify the alternate hypothesis while p and $confidence$ are the variables used to verify the null hypothesis. This way, it was possible to reject the null and the alternate negative hypotheses, thus accepting the alternate positive hypothesis as the most likely to be true.

Table 17 – T-Test results

mean $\bar{x}_c$	mean $\bar{x}_w$	$d.f$	t	p	$confidence$
0.00052	0.000598	9	0.5517	0.2972	99%

Concluding this section, the idea is to present a practical example of data captured from process executions, as shown on Table 15. We have collected data from Model-Driven Software Engineering projects, as mentioned before, because our research group is studying model-driven projects and adequate processes. It is important to mention that the planned process sequences are similar to each other, in that manner, the metric product would result into "16/100000", hence the small difference on the results. We claim that the result would be clearer for more different processes, as indicated in the analytic proof described in Section 4.6.

4.7.5 Threats to Validity

The validity of the results achieved in an experiment might depend on factors involving the experiment settings (WOHLIN *et al.*, 2012). Different types of validity can be prioritized depending on the experiment goal, and we can take actions during the experiment planning in order to minimize those threats. In our case, threats to four types of validity are analyzed: internal, construction, external and conclusion.

4.7.5.1 Internal validity

Experience Level of Participants:. The different levels of knowledge of the participants could have compromised the data. To mitigate this threat, the researchers have trained the participants.

Facilities used during the study: Different computers and configurations could have affected the recorded logs. However, participants worked by using the same computer at the same university, however, in different moments.

4.7.5.2 Validity by construction

Hypothesis expectations: the participants' expectations could have affected the results. To mitigate this threat, we have collected as much data as possible and encouraged the participants to perform as natural as possible. Also, we have concealed the objective of the experiment and their impact on it to avoid them to actively affect their data towards a specific result.

Researcher Interactions: the researchers have helped the participants to complete the planned software. While this affects the collected logs, the goal of the experiment was not to measure the ability of the participants to create software, but to create actual logs of software production.

4.7.5.3 External validity

Interaction between configuration and treatment:. It is possible that the exercises were not accurate for real world applications. The experimental application had simple requirements. To mitigate this threat, we designed the exercises based on reality.

4.7.5.4 Conclusion validity

Measure reliability. It refers to quantitative data used to calculate the process similarities. To mitigate this threat, all data was captured automatically as soon as the participants concluded each activity in order to allow better precision;

Low statistic power. Since we have a small population, we applied T-Tests to analyze the experiment data statistically to avoid the issues with low statistic power. Moreover, we are working on larger scale experiments and applications for the proposed methods.

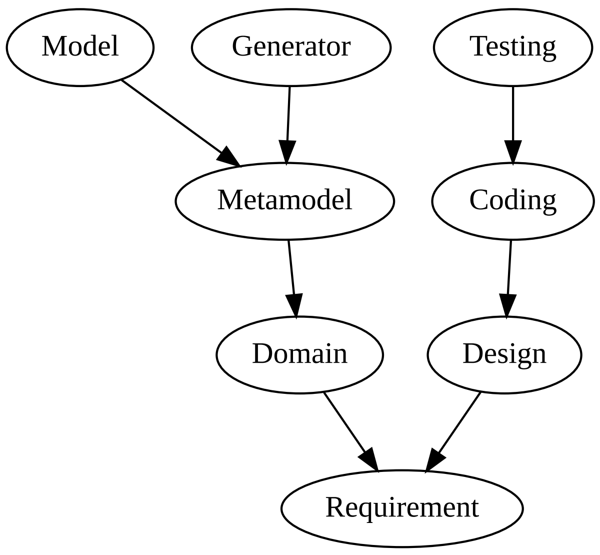
4.8 Method Engineering

Following the validation of the process discovery algorithm and comparison metric, it was investigated how the activities are ordered for possible MDSE processes that include generator development during the software life-cycle.

These activities are considered herein as method fragments in order to employ method engineering (OLLE; VERRIJN-STUART, 1994). In the study presented in Section 4.4, there were activities for: coding; testing; metamodel specification; (code) generator development; and modeling (generator input). Besides these activities, there are three activities that were carried out prior to the study: requirement analysis, application design, domain analysis. All these activities are considered as method fragments in this method engineering task.

Further description for these fragments are also provided in this section. Each one of these fragments are marked in bold to reference the graph in Figure 22, which shows the overview of the fragments and their dependencies, as explained in this section. The nodes represent activities that are required in a MDSE software development life-cycle that includes generator and metamodel construction. The edges represent dependencies between nodes, i.e., each node may point to another node that must be **previously** defined.

Figure 22 – Basic Method Dependency Graph



Source: Created by the author

Requirement analysis is the fragment which is executed to identify the requirements of the application to be developed. According to this study, requirement analysis is an activity that does not depend on any previous software development activity.

Application **Design** involves the activity to establish the architecture of the software application. Design depends on the application requirements to be carried out correctly. For this case, we are considering that design is always present and may be implied even though the design artifacts are missing.

Coding represents the actual programming effort to write source code manually, i.e., excludes code generation. According to arguments collected during the studies, it has been suggested that coding depends on requirements and/or design. Since we are considering the design to be always present, then, it is simpler to consider coding to be dependent on design (which, in turn, depends on the requirements).

Testing involves activities related to verification and validation of the application, by checking if it produces correct outputs for expected test-cases. While software testing activities can be executed during different moments during the software life-cycle, in this specific study, the intent was just to identify the functional testing of the resulting application. In this manner, it has been considered that testing depends on the code to be tested, therefore, it depends on coding.

Domain analysis is the basic activity where the application domain is studied to identify its concepts and glossary. It has been established that this analysis depends on the requirements, which includes a document to specify the application domain.

Metamodel creation refers to the actual activity of creating a metamodel by using a meta-metamodel. This metamodel is based on the requirements or the domain analysis. It is suggested in Figure 22 that it depends on the domain analysis to avoid multiple dependencies.

Model instance definition is the activity to define models that comply with the specified metamodel. Therefore, these model instances depend on the metamodel.

Generator development involves the implementation of the code generator, which depends directly on the metamodel that is used to create the instance models.

4.8.1 Ideal Sequence Synthesis

The dependency analysis of the method fragments can be employed to create new ideal method sequences. These methods are hypothetical and would only exist if the developers had no parallelism and never caused any deffects.

To create these methods, we can simply consider the graph in Figure 22 as a tree, and select the desired leaves that are expected for the software.

For instance, the simplest method that could end into a software that passes the testing is considered by taking the “Testing” leaf and then insert the dependencies before the current fragment recursively, i.e., coding before testing, design before coding and so forth. This basic sequence would contain “Requirements”, “Design”, “Coding” and “Testing”.

This basic sequence does not include code generators or MDSE. In the same sense, we could select all the leaves of the graph. It is important to mention that since there are three leaves, there are at least six different simplest paths to define possible methods.

The first possible order for these leaves would be: testing, (all testing dependencies recursively); generator, (all generator dependencies recursively); model, (all model instance dependencies recursively). There could be also generator before testing, and then model instance and so forth.

These permutations are important for this thesis since they confirm the possibility of top-down and bottom-up methods. For instance, when we consider that testing must come before the code generator, that would imply that a bottom-up method is created. In the same reasoning, top-down methods would require that the model comes before the testing (and coding).

As the previously conducted studies referred in Section 4.3 have encouraged the author of this thesis to create a top-down method, it is possible to suggest a possible example of method: “Requirements”, “Domain”, “Metamodel”, “Generator” with “Model”, “Design”, “Coding” and “Testing”.

The “Generator” with “Model” was added because both fragments are at the same level, therefore, it is not relevant at this point which one should be initiated first.

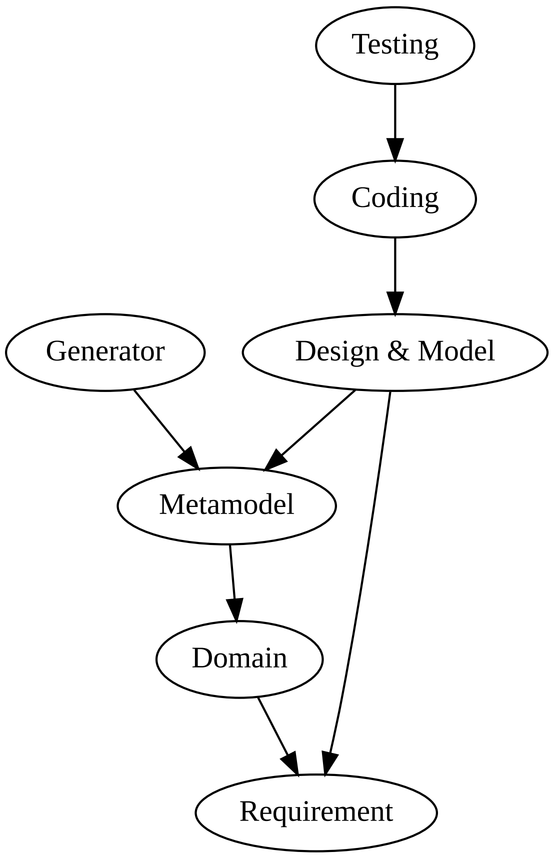
The greatest advantage of an idealistic top-down method is that the first coding effort would be built with the help of code generators, which would theoretically reduce the effort to create software using MDSE.

The major issue is to find a software category that could be easier to be built with MDSE than by creating sample code before the generator. The search for such a system is explained by trying to simplify the provided graph. This is made by merging two artifacts altogether as further described within this section.

4.8.1.1 Design is Model

One suggestion for simplifying the dependency graph is to merge the design to model, as represented in Figure 23. This would allow to create models that represent the software design and use this design to generate code. The suggested sequence for this method would become: "Requirements", "Domain", "Metamodel", "Generator" with "Design & Model", "Coding" and "Testing".

Figure 23 – Dependency Graph for Generative Design Tools



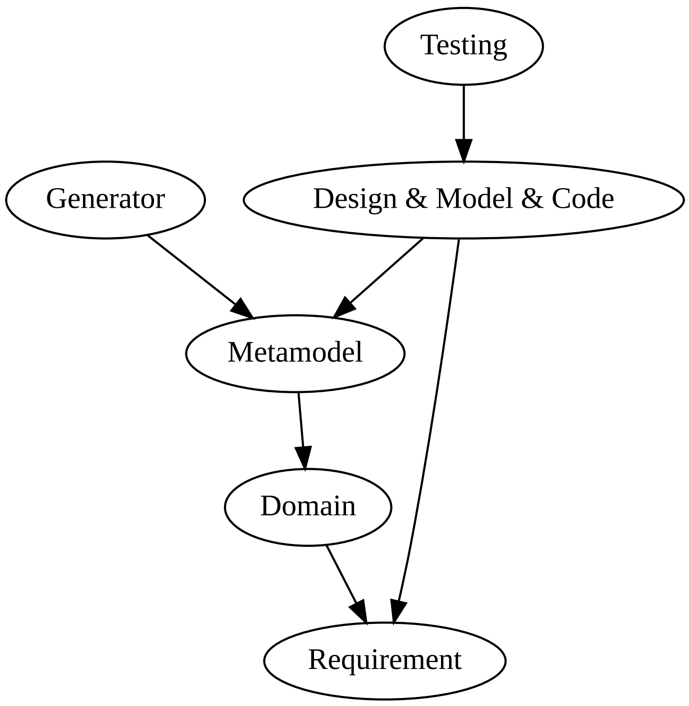
Source: Created by the author

This is the basic idea behind design tools that allow the developers to input the design and then generate code.

4.8.1.2 Design is Model and Code

Another suggestion for simplifying the dependency graph is to merge both the design to model and code, as represented in Figure 24. This would allow to create models that represent the software design and use these models to generate code. In the same sense, upon writing code, the design would be automatically implied. The suggested sequence for this method would become: “Requirements”, “Domain”, “Metamodel”, “Generator” with “Design & Model & Code” and “Testing”.

Figure 24 – Dependency Graph for Model-Oriented Programming

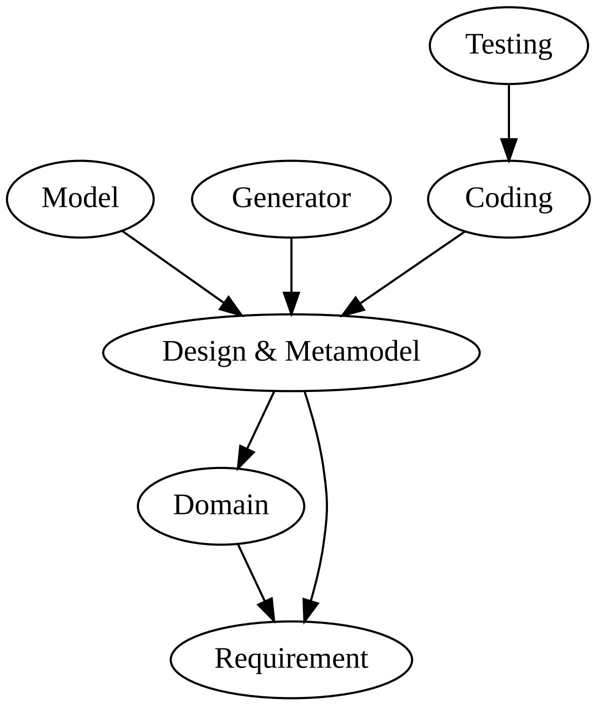


This is the basic premise of MOP (BADREDDIN; LETHBRIDGE, 2013). In this case, the programming language is close enough to the design that it allows round-trip engineering at any moment, allowing developers to return to design models at any time without losing the code semantics.

4.8.1.3 Design is Metamodel

Another possible step that is considered in this thesis is to merge the metamodel to design as represented in Figure 25. This would cause a shift on the abstraction levels, allowing code to execute beside model instances as in Models at Run-time (ASSMANN *et al.*, 2014). This allows to create flexible systems that change their design at run-time. The suggested sequence for this method would become: “Requirements”, “Domain”, “Metamodel”, “Generator” with “Design & Metamodel”, “Model”, “Coding” and “Testing”.

Figure 25 – Dependency Graph for Models at Run-time



Source: Created by the author

Optionally, this could also allow code generators to be either based on meta-metamodels, which could increase their reuse, since they would not depend on specific metamodels, i.e., not required to build new generators. This strategy also allows to generate or interpret code at run-time, which is related to the next suggestion.

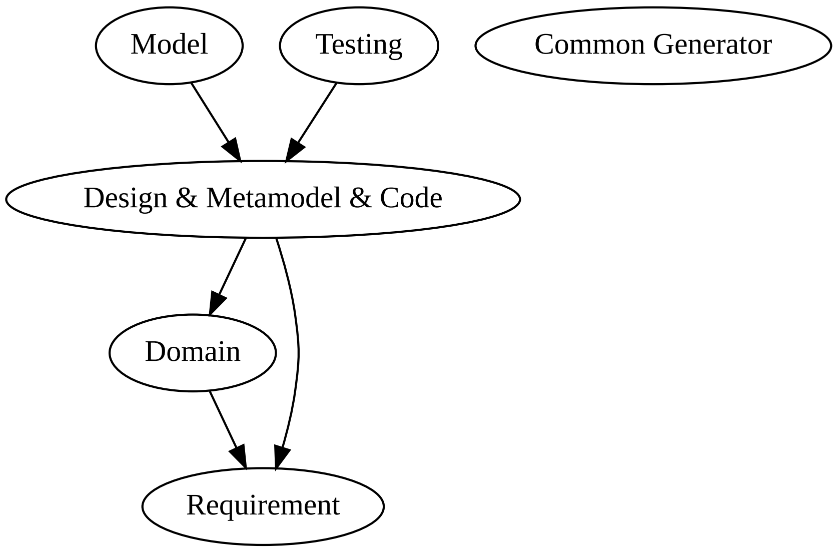
4.8.1.4 Design is Metamodel and Code

A further step on simplifying the graph is to consider joining both merges presented for MOP and Models at Run-time, as shown in Figure 26. This was created as an attempt to go beyond the programming promised by Model-Oriented Programming and make the final software Model-Oriented. Therefore, it would be a specific case of Models at Run-time with benefits present in MOP. While this is an extreme decision, this strategy is the only one among the presented sequences that guides developers to create metamodels before the code. Therefore, evaluating its feasibility would present a software category that benefits from MDSE since inception.

The suggested sequence for this method would become: “Requirements”, “Domain”, “Metamodel”, “Generator” with “Design & Metamodel & Coding” and “Testing”.

This method fixes the generators on meta-metamodels, making them generic for different metamodel types, which means that generators can be common to any software in the category for the same platform. Features available to both MOP and Models at Run-time could be perceived,

Figure 26 – Dependency Graph for Model-Oriented Software



Source: Created by the author

however it is necessary to remind that it could result into a simpler and more restrictive software category. Despite this restriction, it could be actually beneficial to create software systems that can be easily built. Therefore, this suggestion is further discussed and evaluated in Chapters 5, 6 and 7.

4.9 Final Remarks

In this chapter, we have presented an algorithm for process discovery based on the work by Cook and Wolf (COOK; WOLF, 1998), including a complete formalization. We have also proposed a metric that is suitable for process comparison, including a simple and accurate example. As for the validation, the metrics were proven analytically according to all the expected property for metrics and generalized metrics. We have also conducted an experimental study in order to compare process executions and discover a process model compatible to the recorded activity stream. The experiment packing was made available[1].

After validating the algorithm and metric, the same basis used by the algorithm for capturing the activity of the executed process was considered by employing method engineering techniques according to Brinkkemper (1996). This method engineering effort allowed the author to suggest possible methods that could be considered by evolving MDSE. The synthesis of possible methods was possible after establishing how to generalize artifacts. Among these methods, existing instances were discussed. However, one of the methods has led to create a new proposal for MDSE evolution, which is further presented in Chapter 5.

[1] <http://tiny.cc/gottardi-doc>

CHAPTER 5

MODEL-ORIENTED SOFTWARE SYSTEMS

5.1 Initial Remarks

This chapter contains a proposal for a software category referred as Model-Oriented Software System (MOSS). It was named as such since it applies concepts from Model-Oriented Programming to a broader level. In this manner, it blends concepts from MDSE, MOP and Models at Run-time. By blending, these concepts, this new software category could be developed by using a new method that is evolved from MDSE. However, this new method does not completely replace MDSE, MOP or MRT, i.e., there are applications suited for MDSE (e.g. time critical embedded systems) that may not be adequate for MOSS.

MOSS could be defined as more than a development method, but a software paradigm that could be perceived by its end-users, i.e., not only by the developers.

In this paradigm, the resulting software is planned to have a different quality when compared to software created without MDSE. The goal is to create model-aware software that would be different for exposing its underlying models. This exposure allows the software to be operated by employing modeling tools and approaches initially designed for development.

This chapter is organized as follows: The rationale for MOSS and the implicating requirements and features are presented in Section 5.2. Section 5.3 cites related proposals that must be considered prior to establishing MOSS. A concrete version of MOSS applied to Web Services named as Model-Oriented Web Service (MOWS) is presented within Section 5.4. Advantages and disadvantages of this software category are presented and discussed in Section 5.5. Technical documentation on how to implement MOWS systems is present in Section 5.6. A top-down development method suitable for MOWS is shown in Section 5.7. A design and definition language to assist MOWS development is introduced within Section 5.8. Further tools created to assist developers who wish to develop MOWS systems are shown in Section 5.9. Finally, the conclusions for this chapter are written in Section 5.10.

5.2 Model-Oriented Software Systems Definitions

A Model-Oriented Software System (MOSS) is a software system created according to a hypothetical software category that blends principles from both MOP (Model-Oriented Programming) and MRT (Models at Run-time) along with an MDSE method for building software. Despite being hypothetical, the purpose of this chapter also includes presenting concrete applications of this category.

The MOSS software category was defined as a proof of concept for a type of software that would require less effort to develop by using MDSE (since inception) and editing models before source code.

Table 18 – Origins of MOSS Characteristics

Characteristic	Origin
Model Interpretation	MRT
Run-time Flexibility	MRT
Code is Model	MOP
Design Models as Source Code	MOP
Model-Based State	MRT and MOP
Model-Based Data Persistence	MOSS
Model-Based Data Transfer	MOSS
Metamodel as Design	MOSS

A list of characteristics that have been considered from MRT and MOP are presented on Table 18. Therefore, all characteristics presented on the table are available in MOSS. Model interpretation allows the software being executed to process its own model representation. This interpretation also allows run-time flexibility, where the software is capable of altering its own definitions to adapt to changing requirements, for instance, data types and structures. MOP tightens the gap between source code and models to allow reversible/round-trip model transformations. This includes representing the design model inside the source code without semantic loss. Both MRT and MOP software executions may include a state representation as a model, in the case of MOSS, it was intended to make this a rule, which allows the systems to transfer and persist the data as models. This major abstraction to employ models in MOSS development is possible by creating metamodels to represent the design of the model-oriented software. This specific usage of metamodels is the basic construction of MOSS. This is also completely specific to its case, as discussed in Chapter 4.

From the perspective of developers, a MOSS can be developed by using modelling tools since inception. It includes optional support for round-trip software engineering, avoiding semantic loss from transformations, thanks to its MOP properties. From the perspective of users, a MOSS is a system that is configured by using models and runs by using models thanks to its Models at Run-time properties.

5.2.1 Rationale for Model-Oriented Software Systems

The purpose of this section is to present the intent behind proposing MOSS and how it was planned to differ from MOP and MRT. MOSS was planned using the method engineering execution presented as part of Chapter 4.

MOSS was planned as a platform independent definition for software. This was considered to avoid forcing developers to use a new language or platform. Therefore, it was intended to provide a set of requirements that would fit existing languages, libraries, frameworks and tools. This condition makes MOSS different from MOP, which depends on programming language constructions. Indeed, MOP was created by employing a new programming language (BADREDDIN; FORWARD; LETHBRIDGE, 2014).

Besides this difference, MOSS inherits features found in MOP, where the design of the system is published as a model. MOP allows reflection support, which empowers the software to query its own design model. This ability required the authors of MOP to create a specific execution engine for the running software.

In the specific case of MOSS, the published design model is actually a metamodel. This shift in meta level was intended to allow MOSS to handle the model instances that comply to the design metamodel. These instances are handled as data without requiring a specific execution engine. MOSS systems can be compiled to convert the metamodel into data structures (schema), releasing the need of interpretation. This schema allows the software to handle the models as data at run-time. Therefore, the metamodel is the most important artifact when designing MOSS.

MOSS also inherits concepts from MRT since it handles models at run-time. This handling of models by MOSS allows the system to be configured by these models, since these models can store data to serve as parametric configuration of the system. The state of the systems can also be stored as model. Furthermore, this generalization empowers the developers to utilize modeling tools to edit and visualize the configuration and the execution. It is arguable that this support for visualization could also be beneficial to end-users who wish to understand the data of the system.

As the data is stored as models, the transmission of data among modules of the system could also be carried by models. This suggests that the system could be built by using a Domain Specific Language (DSL) that has been optimized to properly represent the concepts of the domain of the MOSS application. Models can be alternatively represented in a different concrete syntax, bridging the gap between the data that is handled by the machines and humans who wish to read this as graphical or textual information.

All these arguments provided herein as the rationale for MOSS have been considered to create proper requirements for these systems. The requirements to build MOSS systems were compiled into Subsection 5.2.2.

5.2.2 MOSS Requirements

A MOSS differs from other proposals discussed in Chapter 4 because the models also have to comply to a set of seven requirements. These requirements were based on the MOSS rationale presented in Section 5.2.1. These requirements must be dealt with by the developers when building these systems, as follows:

Requirement 1: **Design is defined as models**; System design must be defined as models and should be documented as such for future reference.

Requirement 2: **Metamodel defines data structure, glossary and schema**; A metamodel must be defined specifically for the system to represent the data structure for its specific application domain.

Requirement 3: **The configuration of the system is defined as a model**; The run-time configurations of the system are also represented as a model instance, which can be edited by model or diagram editors.

Requirement 4: **If the system must record its state, this state is recorded as a model**; Systems may need to store data to record their current situation. We refer to this requirement as the necessity of storing the state. The state of the system is represented as a model instance.

Requirement 5: **The use cases of operations directed towards the system define the manipulation of the state, which is also a model manipulation**; The operations called by the users are to be designed considering how they manipulate the model.

Requirement 6: **The communication between modules is carried by interchanging models**; Since the data type schema is represented by the metamodel, the native format for files and intra-modular messages are also model instances.

Requirement 7: **The metamodel must follow a suitable structure**; There are design requirements for the metamodel definition in order to allow it to be handled as data and configuration, as well as to be persisted and transferred as a model by the MOSS system.

5.2.3 MOSS Features

While the requirements might seem restrictive, they are important to provide the expected characteristics for a Model-Oriented Software System. The characteristics of a Model-Oriented Software System also include features, which are end-user visible characteristics (KANG; LEE; DONOHOE, 2002).

Feature 1: The first feature is the **Published machine readable design**. This is possible thanks to Requirement 1. Since the design of the software is published as a model and available as a machine readable model for further reference. The system module interface can also be published as a model, which makes this similar to a component or a service with interface documentation requirement.

Feature 2: The second feature is the **Metamodel-Based Data Structure**. This is related to Requirement 2. According to this requirement, the metamodel must be defined to specify the data structure, glossary and schema. This means that the system uses a domain specific language for its data handling. This can also include a language glossary for reference and schema for its persistence.

Feature 3: The third feature is **Model-based configuration**. The configuration of MOSS modules are represented as model instances, taking advantage of model and diagram editors for easier edition.

Feature 4: The fourth feature is **Model-based state**. As the state of the module (in case of needing to store state) is a model, according to Requirement 4, administrators and developers can easily watch the state model as it is changed. This can be beneficial to understand, test, debug and edit the state using model-driven editors and transformers.

Feature 5: The fifth feature is the **Transformation tools for state update**. According to Requirement 5, the use cases of the system that modify the state also modify the model. This feature is similar to the causal reflection present in MRT implementations. This is possible as a direct effect of the previous requirement, where the system states are models. This allows to define the behavior of the running system by using model transformation techniques. This is provided as the fifth feature.

Feature 6: The sixth feature is the **Model-based data transmission**. Since the metamodel that defines the domain concepts and the data glossary is the only schema valid for the software system, every communication between modules is carried on a domain specific language. Every payload of interchanged data is, therefore, a model instance that can be edited and visualized by model and diagram editors, which can ease comprehension, data analysis and debugging.

Model-Oriented Software Systems are simply hypothetical. Model-Oriented Web Service have been defined as a concrete epitome for Model-Oriented Software, as described in Section 5.4, including concrete examples in Subsection 5.6. Other technological domains are discussed as part of Appendix C.

5.3 Proposals Related to MOSS

In this section, it is discussed how Model-Oriented Software System was defined and how it compares to related proposals, i.e. MOP, Models at Run-time and MOD.

5.3.1 Object Orientation Comparison

Model Orientation (MO) can be defined as a paradigm in which the model is the most important element which every other is oriented towards. When this envision is applied to software (i.e., MOSS), Model Orientation represents the usage of models throughout the complete software life-cycle. In this fashion, models are used to develop and to execute the software.

Table 19 – Basic Comparison Between Object Orientation and Model Orientation

Characteristic	Object Orientation	MOSS
Declaration	Classes	Meta-model
Data Instance	Object	Model

A comparison is shown on Table 19, which has three columns, including a name for the characteristic and how they are treated by both Object Orientation (OO) and MOSS. In OO, the code can be declared within classes, the data is structured and transmitted as object instances (JACOBSON, 1992); in MOSS these concerns are performed by employing models: code can be declared within meta-models, data is structured and transmitted as model instances.

In this section, MOSS is also compared to similar concepts. MOSS also has advantages and disadvantages for developing Web Services, which are discussed in the following subsections.

5.3.2 MDSE Compared to MOSS

Model-Driven Software Engineering (MDSE) involves methods for software development in which models can replace source-code completely or partially (PASTOR; MOLINA, 2007). In MDSE, the metamodel is typically used to define the modelling language and the model instance is the artifact employed to replace the code completely or partially.

In MOSS, the metamodel is used to define both the language, the handled data types as well as the and the data structure employed by the software. This metamodel supports the system to handle model instances at run-time.

Therefore, in MOSS, the metamodel plays the concrete role of defining the language and auxiliary code, including data structures and how to handle these structures. The models can both store data and code to be interpreted at run-time. These characteristics represent a different approach to how to deal with model meta-levels that is described within the MDSE literature (BRAMBILLA; CABOT; WIMMER, 2012). This allows us to provide a table for more detailed comparison on each modeling level that was described by Brambilla, Cabot and Wimmer (2012) in order to introduce MDSE. According to Table 20, there are four columns. The first column contains the modeling level. The second column presents the name of the artifact in this abstraction level. The third column names the purpose of this level for MDSE development and the fourth column names the purpose of this level for MOSS development.

Table 20 – Basic Comparison Between MDSE and MOSS

Level	Name	MDSE	MOSS
M3	Meta-metamodel	Language for Metamodeling	Language for Metamodeling
M2	Metamodel	Language Specification	Application Design
M1	Model	Development Artifact	Data File
M0	Code	Real World Objects	Program or Model Interpreter
M-1	Data Instance	–	Models in Memory

Table 20, also indicates that the modeling levels are shifted when comparing MDSE to MOSS. The usage of metamodels as design is the most specific change of MOSS when compared to any other related method. Despite this difference, it is still possible to combine MDSE and MOSS and by using MDSE to generate MOSS code. MOSS does not replace the generic generative capabilities of MDSE methods. MOSS is not dependant on MDSE either, since MOSS software can be written with or without use of code generation, e.g. manual coding.

The usage of generative programming in MDSE accelerates development of a code that is either repetitive or reusable within similar projects. However, the main goal is still to generate high quality source-code, that, even though being automated, could be achieved by manually coding as well (HERRINGTON, 2003).

However, while the final software developed with MDSE methods may be completely unrelated to modelling techniques, the resulting MOSS is still based on models because it uses models at run-time.

5.3.2.1 MOD Compared to MOSS

Model-Oriented Development (MOD) is a development method related to MDSE. Authors of MOD tools claim it makes it possible to apply code generation at any time during the development, i.e., less dependant on code generation when compared to a MDSE project (CodePlex, 2016).

It shares the Model-Oriented feature of employing models to represent the final software, however, only the development is defined as model-oriented to be considered Model-Oriented Development.

In this manner, MOD is not an alternative to Model Orientation in the context of the resulting software. A model-oriented software can be developed by using MOD techniques, while this is not required. What defines a software with Model-Orientation is how the models are employed at run-time, as well as its data structures.

5.3.2.2 Model-Oriented Programming Compared to MOSS

Model-Oriented Programming (MOP) is a paradigm where the software source code has additives to its semantics in order to allow round-trip engineering from design to code without loss of semantics. Therefore, MOP tightens the gap between design models and code (BADREDDIN; LETHBRIDGE, 2013).

This tightening requires the creation of new programming constructions which may become incompatible to existing software. In the case of MOSS, it was planned to avoid dependencies on new languages and platforms in order to mitigate this issue.

Table 21 – Basic Comparison Between MOP and MOSS

Characteristic	MOP	MOSS
Goal	Tighten design and code	Metamodeling as design and code generation
Language Support	Requires specific programming language	Not tied to programming languages

The differences between MOP and MOSS are summarized on Table 21. This table is composed by three columns, the first column indicates a characteristic that is different between MOP and MOSS; the second indicates the characteristic from the perspective of MOP while the last column indicates the perspective from MOSS.

It is important to elucidate that these concepts have slightly different goals, i.e., MOP is more focused on tightening the gap between design and code, while MOSS is more focused on a method for metamodel-based design for native model interpretation. MOP also requires a specific programming language for its development, while MOSS is based on existing metamodeling languages.

5.3.2.3 Models at Run-time Compared to MOSS

Models at Run-time (MRT) is described as a paradigm in which models can be used as code for interpretation and/or for data (ASSMANN *et al.*, 2014; GIESE *et al.*, 2014; BENNACEUR *et al.*, 2014). This paradigm is a broad definition in which software can employ models, which also includes model-oriented software.

In this case, a MOSS is a specific case of MRT, in which the data and code structures are based on the metamodel which specifies the model structure to be handled at run-time.

The differences between MRT and MOSS have been summarized as Table 22. This table is composed by three columns, the first column indicates a characteristic that is different between MRT and MOSS; the second indicates the characteristic from the perspective of MRT while the last column indicates the perspective from MOSS.

Table 22 – Basic Comparison Between MRT and MOSS

Characteristic	MRT	MOSS
Scope	Broader case of model interpretation	Specific usage of metamodeling and data models
Focus	Dynamic System Flexibility	Data Handling with MDSE tools

According to the presented table, it is important to clarify that MOSS also employs models at run-time, therefore, it is still a specific case of MRT. MOSS has a specific list of features, as described in Section 5.2.

5.4 Model-Oriented Web Services Definition

While MOSS is a hypothetical definition for a software category, Model-Oriented Web Service (MOWS) are concrete and feasible software systems that inherit all characteristics of a MOSS. MOWS are web service systems that comply to MOSS definitions presented in Section 5.2, i.e., they follow both the requirements and the features proposed for a MOSS. MOWS systems are concrete and functional, allowing to implement actual software systems as case-studies (further discussed in Appendix C).

MOWS systems are feasible and functional since they can be built by following both WS standards and modeling tool standards at once. Indeed, both W3C Working Group (2004) and Object Management Group (2014) recommend XML for data interchange. This allows these systems to operate solely by using models in XMI and XML at the same time. This makes these systems inextricably compatible to WS standards and modeling tools without requiring any conversion, since they use models as their native format.

From the perspective of developers, a MOWS is a Web services system developed by employing a MDSE process. They follow the same set of seven requirements of MOSS appended by a few new requirements. Therefore, the current list is numbered from eight to thirteen, which were specifically adapted for the case of MOWS systems:

Requirement 8: **MOWS must comply to MOSS requirements.** The first requirement of a MOWS system is to comply to all the requirements of MOSS.

Requirement 9: **MOWS must comply to WS protocol rules.** A MOWS must be a WS itself. Therefore, it has to comply to protocol rules defined for WS systems.

Requirement 10: **The client can invoke operations provided by the WS server while the server cannot invoke the client.** In order to comply with protocol rules (or limitations) only servers can be called by clients. Therefore, clients cannot be indiscriminately invoked by the server. If the client requires to be notified by the server, it must first invoke a blocking call and wait for the server to return a message.

Requirement 11: **WS interfaces and ports design are defined as models.** WS interfaces must be defined as models, which is commonly performed in many other approaches by employing WSDL. This is related to MOSS Requirement 1 which states that the system must be designed by using models. In this case, the service interface is included as a specific requirement.

Requirement 12: **The configuration of WS servers and clients is defined as a model;** The run-time configurations of WS servers and clients are also represented as a model instance, which can be edited by model or diagram editors. This is a MOWS specific requirement that is based Requirement 3 of MOSS.

Requirement 13: **The communication between WS servers and clients is carried by interchanging models;** Since the data type schema is represented by the metamodel, every message between clients and servers are also model instances. This is a MOWS specific requirement that is based on MOSS Requirement 6.

5.4.1 MOWS Features

Like MOSS, MOWS also include the same set of six end-user visible features, described next after adapting them accordingly.

Feature 7: **Published WS Interface**. Thanks to Requirement 11 and Requirement 13, the WS interface can be published as a model. WSDL instances can be considered models since their schema can also be mapped to a metamodel. This feature is similar to any WS with publishing requirement.

Feature 8: **Model-based state**. Since MOWS follow MOSS requirements (Requirement 8), the state of the server and the client (in case of needing to store state) are models. This allows administrators and developers to easily watch the state model as it is changed, which can be beneficial to understand, test, debug and edit the state using model-driven editors and transformers.

Feature 9: **Model-based configuration**. The configuration of both server and clients are represented as model instances, taking advantage of model and diagram editors for easier edition. This feature is related to Requirement 12.

Feature 10: **Model-based data transmission**. Since the metamodel that defines the domain concepts and the data glossary is the only schema valid for a web service server, every communication between servers and clients is carried on a domain specific language. Every payload of interchanged data is, therefore, a model instance that can be edited and visualized by model and diagram editors, which can ease comprehension, data analysis and debugging. This feature is possible thanks to Requirement 13.

5.4.2 Legacy WS Differences

A MOWS system is still technically a WS system that can completely adhere to standards. From a technical point of view, a MOWS system behaves as a typical WS.

A typical WS that employs XML for data interchange, the developers have the option to define the data format specification by employing XSD (W3C XML Working Group, 2012). For a MOWS system, the format specification is also defined by the meta-model instead of a separate specification.

On the other hand, MOWS systems depend on metamodels and the metamodel of these meta-models. The data payload that is transferred between clients and servers may include this dependency on their format headers. This dependency does not restrict modules, e.g. clients, from being developed without MO characteristics. This means that non MOWS-based clients could still invoke MOWS services by simply using their published interfaces along with an existing WS technology that adheres to the same underlying WS standard.

Since MOWS systems also adhere to requirements from MOSS, models can both represent data and/or be interpreted to allow flexible configuration. This means that we are not limiting the models within MDSE as "Models are Code", in this case, models are software, data structures, the data itself along with the state representation of the running software. Therefore, both code generation and code interpretation can be employed for the same final application.

This epitomization of MOSS as MOWS represent a software category where models are used throughout the complete development life-cycle and during the execution, being visible to the end-users, which we argument that is what could represent the evolution of MDSE into a paradigm. In this manner, models are not only the code, models are also the data and can represent the configuration, settings and state of the running application.

5.5 MOWS Advantages and Disadvantages

In this section there is a qualitative analysis about the advantages and disadvantages that can be identified when comparing traditional WS systems to MOWS systems.

The advantages are distributed into three categories: **Documentation**; **Readability** and **Configuration Flexibility**. The MOWS documentation advantages are related to the creation of a single metamodel that provides documentation for the system definition without requiring the definition of a completely separated artifact. This metamodel can also have a graphical representation including notes on the application of each meta-level element.

The metamodel serves as a multi-role artifact that avoids the drift of not being updated along with the data structure format it represents, since it is in fact the artifact used to define this structure (PASTOR; MOLINA, 2007). This characteristic is leveraged by the fact that the metamodel can be the input to several transformations, including the servers, clients, databases and auxiliary handling code, leaving only a small portion of server behavior to be added manually by the developers.

The MOWS readability advantages are consequential to the ability of employing models at run-time (GIESE *et al.*, 2014). This includes the option of using model interpretation to increase understanding of the software at run-time. Since the final software operates natively by using models, it is possible to manage the software by using modeling tools. Thus, the data transmitted and recorded by the MOWS system is a model that can be handled by model editor tools without any necessary conversion.

In the same context, the run-time models could be structured to allow configuration flexibility of the MOWS system. The models employed at run-time may contain WS-related configuration variables, meta-data targeted at adapting the data instance models or interpretative code.

On top of these differences, the MOWS system would still adhere to WS standards, i.e. the W3C recommendation (W3C Working Group, 2004), increasing compatibility to WS that are not model-oriented. Since MOWS systems are designed to communicate by employing model instances, it is possible to come up with a common modeling language that can be used by both humans and machines. This is leveraged by the fact that modeling languages can have more than one concrete syntax. This means that model instances can be represented in different formats, e.g. data trees, human readable text or graphical diagrams (BRAMBILLA; CABOT; WIMMER, 2012).

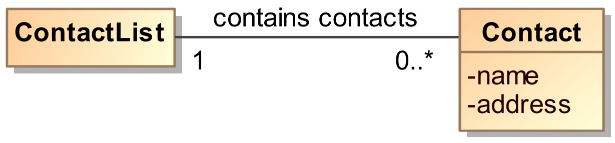
Despite the presented advantages, there are also disadvantages that could discourage the adoption of MOWS into real world projects. Therefore, we are not claiming that MOWS could replace every kind of WS system if these disadvantages become problematic. The disadvantages are grouped into two categories: dependency and overhead.

The dependency problem is related to the use of models which refer to metamodels, which, in turn, refer to *meta-metamodels*. This dependency makes the code dependent on the *meta-metamodeling* specification. These specifications may be revised by an external organization, breaking compatibility to new modeling tools or clients, which might force MOWS systems to be updated.

While this update might be beneficial, it would require specific maintenance efforts. However, this effort could be diminished by using code generation. Despite this, the code generation tools would also be affected by this specification update and the final software would be dependent on the code generators, as also reported in the literature (GOTTARDI; BRAGA, 2015).

Another dependency is imposed by the recommendation of XML standards (W3C XML Working Group, 2012; Object Management Group, 2014). This hinders the option of employing alternative data formats that could be preferred over XML for specific domains. Despite this

Figure 27 – Basic CWEB Conceptual Model



Source: Created by the author

recommendation, case studies have been carried out (further discussed in Appendix C) by using other formats. While it is technically possible to use non-XML formats for models transmission, this would break compatibility with modeling tools.

The overhead problem affects both data transmission and parsing. Since model-orientation would add dependency for metamodeling, the metamodel would also have a recursive dependency towards the metametamodel. These dependencies increase overhead for parsing the models when compared to typical XML parsing. Because of this overhead, a MOWS system can have a worse performance than a typical WS system. Also, this issue could cause the MOWS approach to be discouraged for time-critical systems.

Despite not completely solving this issue, it is important to point that since a large amount of code production can be automatized, better optimizers for this kind of systems could be implemented in the future. This is possible by using compilers or code generation techniques.

In order to assess the impact of this overhead, we present possible implementations of MOWS (Section 5.6) and a mathematical analysis of the overhead (Section 6.2) in Chapter 6.

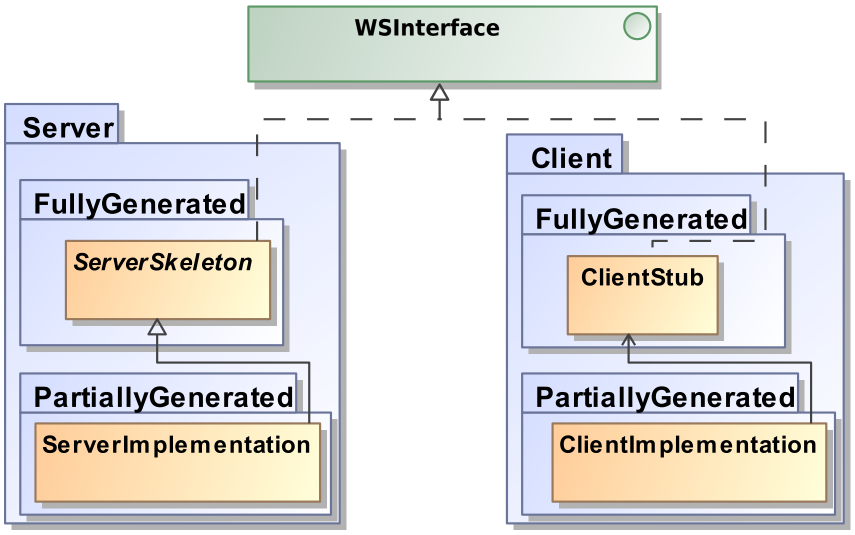
5.6 MOWS Development Use Case

In this section, an example is presented to illustrate the development of a MOWS system. A sample system referred herein as CWEB was created to handle a contact list over the web and is used within the section. The requirements for CWEB are: 1) store a single contact list; 2) contacts have a name and an address; 3) create a new contact; 4) retrieve an existing contact; 5) update an existing contact; 6) delete an existing contact. Besides these functional requirements, CWEB must adhere to all MOWS and MOSS requirements. CWEB is completely data centered and was added for simplicity. For a more complex system based on MOWS, please refer to the example in Subsection 5.8.1 (inside Section 5.8).

As explained in Section 5.7, the central artifact of a MOWS system is the metamodel. This implies that a new metamodel must be created or reused from an existing specification for the new MOWS system. In this example, the metamodel is created based on an initial conceptual model. In order to create the conceptual model, it is necessary to perform domain analysis. For simplicity, CWEB only handles data entities for contacts with fixed data types.

Figure 27 presents a conceptual model for the CWEB system built based on the concepts gathered from the requirements: contact list; contact and address.

Figure 28 – Interface and Component Diagram



Source: Created by the author

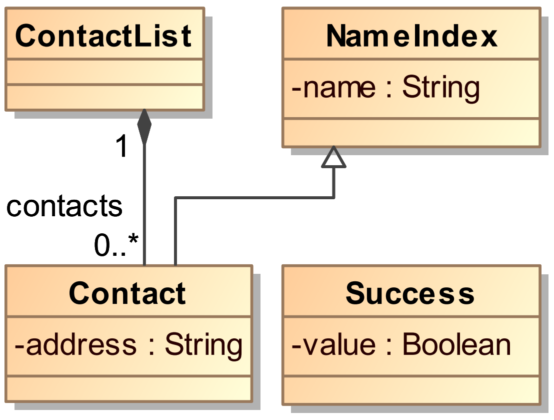
Figure 28 presents the basic design of a web service system that is used as a basis for all the subsequent examples. It is composed by a generic interface (“WSInterface”) that is shared by a server and client. The server is composed by a skeleton (“ServerSkeleton”). The skeleton can be automatically generated, but it is simply an abstract implementation of the server that has to be made concrete by the developers by creating an implementation (“ServerImplementation”). On the client side, there is a stub, which can be created by code generators. The developers would then call the interface implemented by this stub, which would redirect the invocation to the server.

Following the conceptual model, it is then possible to design the interfaces. At this point, the interface should include the method signatures according to the requirements. These signatures represent the methods that a server would implement. Any compatible client would be able to call these services via the web service interface.

The first method signature is defined to add the contact provided by the client to the list on the server; The second method signature allows the client to request a contact object based on a name definition. For updating the address of a contact, the server would also provide a method that allows the client to submit a new contact object. Finally, the last method allows the client to request a contact deletion. Similarly to the retrieve method, the client submits a name definition and expects a response on the success status of the request. Further illustrations for the interface design with methods are provided as part of practical cases (Subsections 5.6.3 and 5.6.4).

After the interface design, the CWEB basic design is complete. Then, it is required to develop the metamodel that is used to specify the language and structure of the MOWS system, as shown in Figure 29.

Figure 29 – Design for CWEB metamodel

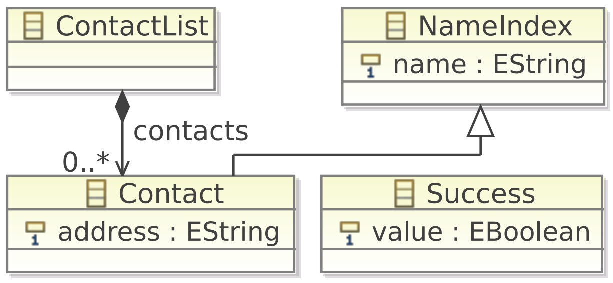


Source: Created by the author

In the case of this fixed data type system, the metamodel would simply contain meta-classes to represent the data types, which are the contact and the address. We also add an aggregator that represents the contact list.

In order to comply with the MOWS Requirement 13, which states that every server response must be a model, it is also important to add conceptual classes for the response of the contact deletion method.

Figure 30 – Actual CWEB metamodel



Source: Created by the author

In Figure 30, it is presented the ECore version of the metamodel that includes these concepts. The basic difference between the models is that the version presented in Figure 30 represents the actual metamodel to be used to create compliant models. These models would then be employed in data transmission. Please notice that in this specific case, the metamodel resembles a typical class diagram which contains meta-classes based on the conceptual model. However, in order to create a compatible metamodel, it is important to remember that only a single object may be transferred at a time. To cope with this issue, we must define composition relation types, which allows objects to be transferred along with its composite objects. Therefore, the contact list metaclass must have a composition towards the contact, which as the address as an attribute.

The basic requirements for creating a web service system adhering to W3C standards (W3C Working Group, 2004; W3C XML Working Group, 2012) include specifying XSD and WSDL files. These files are used to generate the client and server skeleton/stubs completely. This means that even the model transmission behavior can be generated. Yet, the server would lack code body for the methods, which needs to be implemented according to the chosen platform.

Therefore, the next activity is to create the XSD based on the metamodel. This can be automatically generated without requiring further efforts by the developers. From the MDSE perspective, the XSD represents the same role as the metamodel, so this is basically a model transformation. After the XSD is complete, it is possible to establish the WSDL for the designed interface. The generated XSD is used for the data types, which are based on the meta-classes. As these data types are generated from the metamodel, they also allow the MOWS system to handle models at run-time.

Since the following activities of this process are dependent on the platform, we provide subsections for practical cases. The practical cases are provided as proof of concept for the activities described within this chapter. They also pose as suggestions on platforms to be used, which were successful to provide all the requirements described for a MOWS system.

It is important to mention that since the cases are designed to adhere to standards, clients and servers are completely inter-operable among the supported platforms. This feature allows heterogeneous platforms of clients and servers to communicate transparently. In this thesis, we refrain from comparing the platforms. Our goal is not to restrain the freedom of choice by favoring a platform, but to provide evidence that MOWS systems are not platform-specific. Therefore, in case of adopting MOWS features, project leaders would be able to choose the platforms that best suits their needs.

5.6.1 Practical Case: MOWS with REST

REST is a recommendation for WS that takes advantage of the request headers of the HTTP protocol, with the objective of parameter passing. Based on the request style of HTTP, which provides verbs for action (namely “methods”) to be carried on the addressed resource identified by the request, REST provides a simple and common manner for dealing with resources stored on web servers (FIELDING; TAYLOR, 2000).

For instance, the HTTP methods “POST”, “GET”, “PATCH” and “DELETE” may be mapped to the CWEB methods “addContact”, “retrieveContact”, “updateContact” and “deleteContact”, while the resource address is fixed as “Contact”.

Since our MOWS recommendation includes the requirement of only using models for transmission (Requirement 13), in the case of REST, using variable parameters as part of the resource address is discouraged. This resource address is transmitted as a basic string (text) that may not be sufficient to store a model object according to XMI recommendations. Therefore, in this specific case, it is not possible to guarantee that the transmitted model represents the data being handled during the specific request.

One key advantage of using MOWS with REST is that it becomes easier to save the model by using typical internet tools, since it has the ability to transmit the model with less protocol overhead, as opposed by SOAP.

5.6.2 Practical Case: MOWS with SOAP

SOAP is the protocol recommended by W3C for web service systems (W3C Working Group, 2007). The main advantage of employing SOAP for developing a MOWS system is that it has extensive standardization for the interface format, named as WSDL, and the data structure format based on XSD. Also, it provides tool support to generate code based on WSDL artifacts.

This standardization encouraged us to develop MOWS systems using the SOAP protocol, allowing to define metamodels in XSD and interfaces in WSDL. After defining these artifacts according to the MOWS system requirements, the development can be carried on using typical processes for SOAP WS system development, because these artifacts already involve the differences related to the interfaces, data structures and transmission.

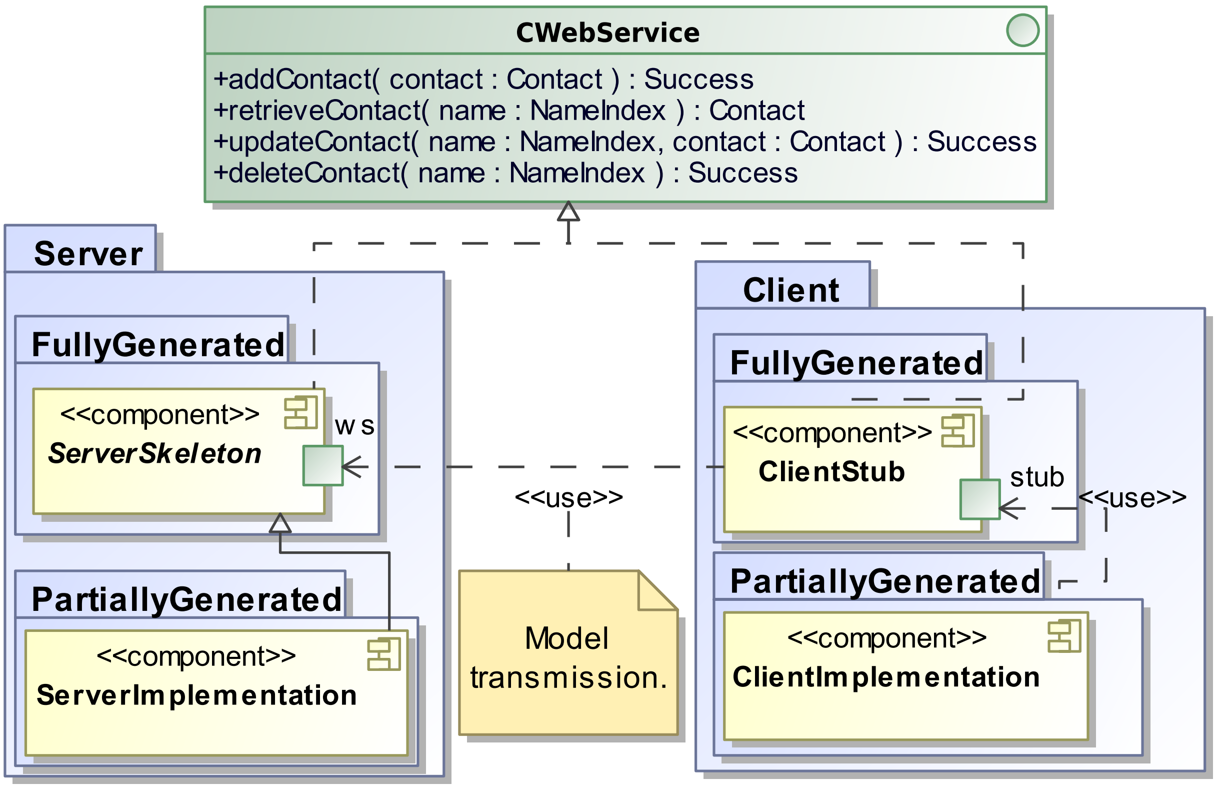
5.6.3 Practical Case: MOWS with SOAP using Genivia gSOAP 2

Genivia gSOAP 2 is a framework and a tool-kit for implementing web services. Its project was commenced with the objective of creating a tool-kit that allows text based communication over the network. After the W3C published the SOAP standard, the project was eventually named as gSOAP (W3C Working Group, 2007), including support to create servers and clients according to the standard, though it has added REST support afterwards (FIELDING; TAYLOR, 2000; ENGELEN; GALLIVAN *et al.*, 2002). The target platforms of gSOAP are the C and C++ programming languages. Servers based on the gSOAP framework can function in standalone mode, as modules for Apache HTTPD (APACHE, 2016) and as Common Gateway Interface executables, which would also be invoked by HTTP servers.

The code generator supplied with gSOAP is capable of generating C++ classes for both the client and server by reading the WSDL and XSD files generated during the previous activities.

The component diagram for this case is represented in Figure 31. In the case of creating the CWEB MOWS system using gSOAP, the tool-kit generates the server and skeleton class as a single abstract class named “CWEBService”. The concrete class must be implemented by the developers, who would add the behavior code of each specific method, namely: “addContact”, “retrieveContact”, “updateContact” and “deleteContact”.

Figure 31 – Interface and Component Diagram for the gSOAP 2 Case



Source: Created by the author

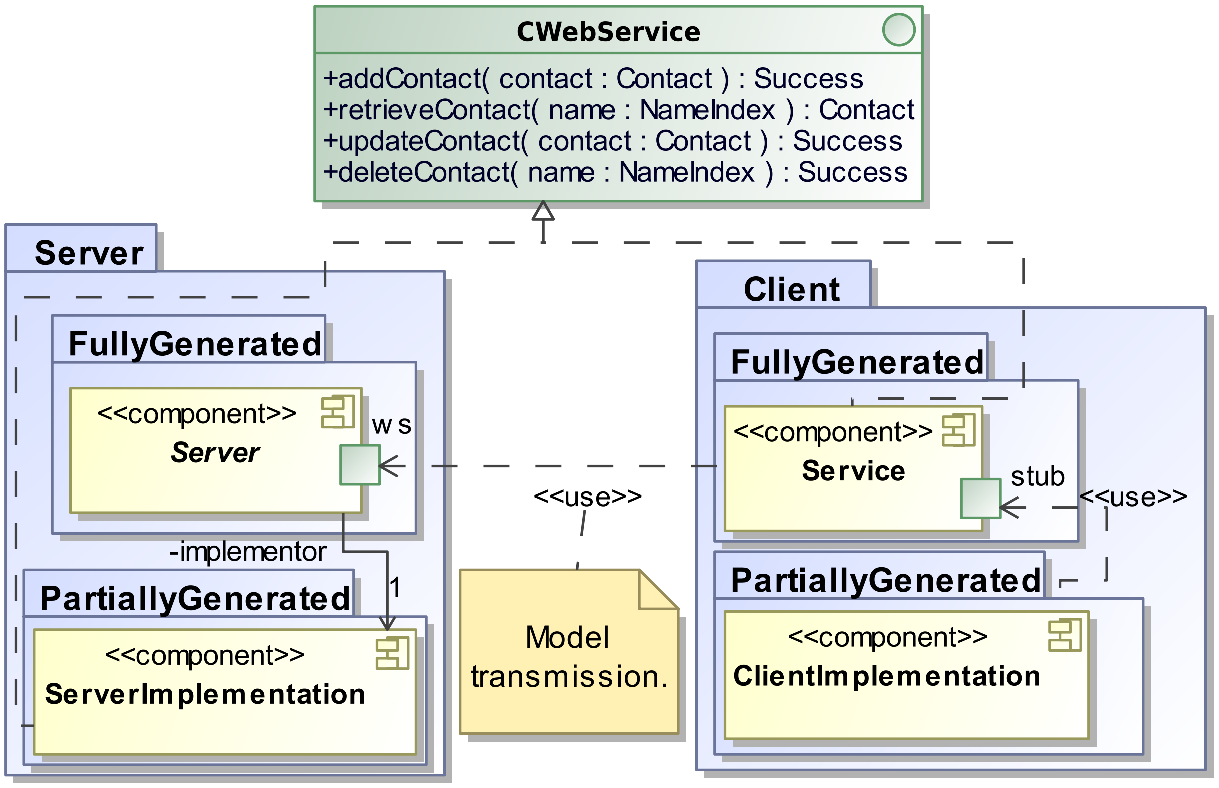
Since server is generated as an abstract class that inherits the service interface, the concrete server class would also be the main class to be invoked by the server. The behavior required by the WS must be inherited from the framework by using the generated "soap" class. Therefore, the developers would only need to focus on the method bodies and the MOWS system would still adhere to WS standards and provide MOWS features, since they were established during the definition of the metamodel and the service interface.

On the client side, the tool-kit generates the stub which is named as "CWEBProxy". This is a concrete class with implementation for the methods to delegate the call to the server. The client is then able to invoke the server via the proxy.

5.6.4 Practical Case: MOWS with SOAP using Apache CXF 3

Apache CXF 3 is a framework and a toolkit for implementing web services. Its project was started after merging WS framework projects (Apache Foundation, 2017). It also includes support for creating clients and servers based on the SOAP standard by W3C (W3C Working Group, 2007). Besides SOAP support, CXF also allows to develop REST Web Services (FIELDING; TAYLOR, 2000). The target platform for CXF is the Java Programming Language, and it is designed to run with Java Standard Edition or Java Enterprise Edition libraries. Web Service servers powered by CXF can function in standalone mode or as Java web server modules known as “Servlets” (Apache Foundation, 2017).

Figure 32 – Interface and Component Diagram for the CXF 3 Case



Source: Created by the author

The component diagram for this case is represented in Figure 32. A Java interface is generated to represent the interface defined by the WSDL file. In comparison to the gSOAP 2 case, where the server inherits the interface, CXF creates a generic and concrete service class that requires the developer to provide an object instantiated from a concrete class that actually implements the service interface. This concrete class is commonly referred in this thesis as “Implementor”, though this name is arbitrary. “Implementor” must be manually written by the developers to define the behavior of the server upon receiving calls for the defined methods, namely: “addContact”, “retrieveContact”, “updateContact” and “deleteContact”.

After creating the concrete class that implements the methods, the developers must import this new class into the skeleton, which is a concrete class generated for the server, named “CWEBServer”. The server would delegate the calls received via the WS interface to this concrete class.

On the side of the client, a concrete class is generated for the stub, named as “CWEB-Service”. This class has complete implementation for invoking the web server for the calls performed by the client.

5.6.5 Usage of the Resulting System

By employing any compatible platform, including those presented in the previous subsections (though not limited neither to practical examples nor to their platforms), it is possible to consider the usage of the resulting system. Clients and servers based on MOWS would maintain compatibility without being dependent on a specific platform since MOWS systems also comply to WS standards or recommendations. From the perspective of a user that wishes to use a client, the behavior of a MOWS system would provide the same transparency as any other WS system that follows the given standards or recommendation.

From the perspective of a developer that wishes to develop a client to connect to a MOWS system server, they would perceive the server as a usual WS server. This is possible since the only actual differences of a MOWS system when compared to a typical WS system are the data-types and the interface specifications, which would be completely provided to the developers wishing to create new clients. Once the clients are written to comply with these specifications, they would also comply to the MOWS system features of model transmission and handling.

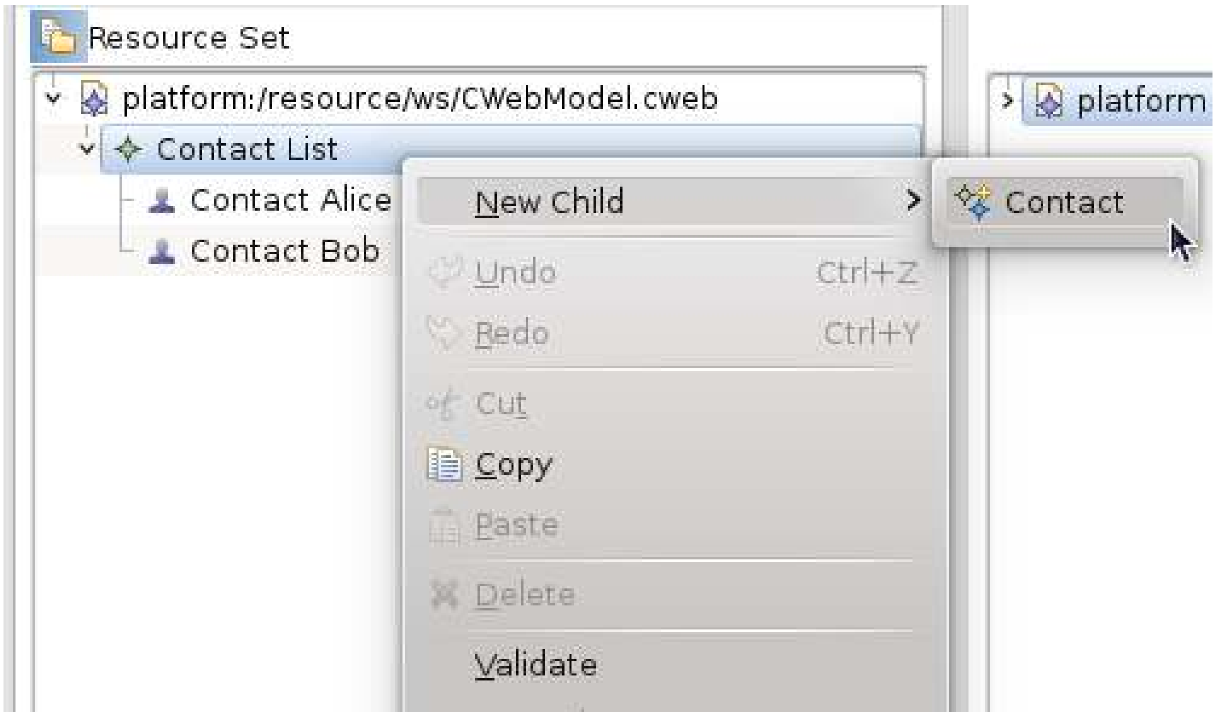
In addition to maintaining the development processes of these developers, the usage of model transmission allows developers to visualize the transmitted data using modeling tools. The first model editor is the abstract syntax tree editor. For instance, in Figure 33, we provide an example of the model editor based on Eclipse EMF (Eclipse Foundation, 2015b). Eclipse Modeling Framework (EMF) is a project for modeling tool development, part of Eclipse IDE (Integrated Development Environment) project (Eclipse Foundation, 2017).

In this editor, it is possible to create every kind of object according to the metamodel specification, while following the same structure of the respective XML file.

Besides an abstract syntax, modeling languages can have multiple concrete syntax instances. In Figure 34, we provide an example of a diagram tool for editing the data handled by the CWEB system, which also employs tools based on Eclipse EMF. The back canvas of the diagram represents the contact list, while the rectangles placed on the canvas represent a contact in the list. Each rectangle contains an icon that represents the contact type and two labels. The first label presents the name, while the second contains the address.

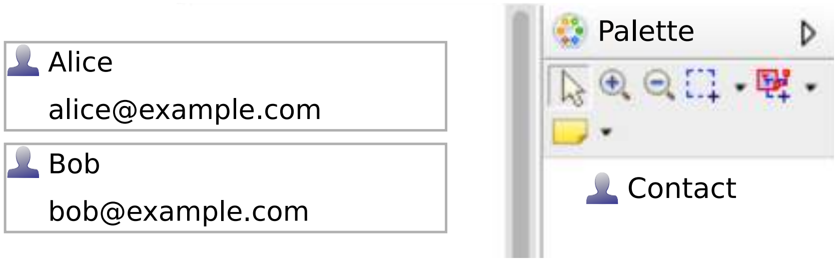
Among possible concrete syntax examples, in Figure 35 we also provide an example of a textual representation for the same model handled by the CWEB system, created by using XText, a language engineering tool-kit (Eclipse Foundation, 2016). In the concrete syntax created for the CWEB model, the list declaration is initiated by the "contacts" word, then each contact is declared sequentially in a comma-separated list. The addresses and names must be written between double-quotes. Each name should follow the "name:" word, while the address should follow the "address:" word.

Figure 33 – CWEB Model Editor



Source: Created by the author

Figure 34 – CWEB Diagram Editor



Source: Created by the author

5.7 MOWS Generic Development Method

This section includes suggestion on how a development method could be defined for MOWS systems. It was created considering that the metamodel is the key artifact of the development. According to this process, as shown in Figure 36, in order to develop a MOWS system, the metamodel must be defined first.

The metamodel is defined by performing a domain analysis in the context of the system under development. The metamodel is also a model itself, which allows model transformation. This process involves three model transformations.

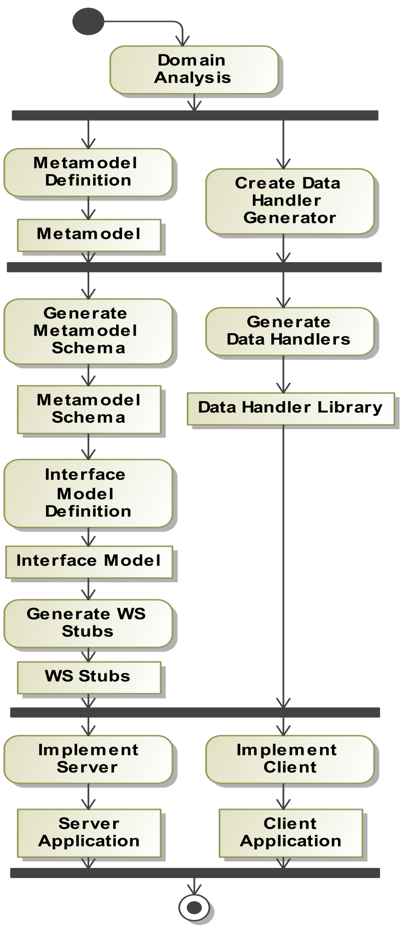
The first transformation is defined as "model to model" by taking the metamodel as

Figure 35 – CWEB Text Model Editor

```
contacts
      name: "Alice" address: "alice@example.com",
      name: "Bob"   address: "bob@example.com",
;
```

Source: Created by the author

Figure 36 – Simplified Development Process of a Model-Oriented Web Service



Source: Created by the author

input and outputting an XML schema, which is used to define the format of model files and intra-modular messages. The second transformation is a code generation used to generate the system structure, by inputting both the schema and the system design model, which is similar to many other approaches documented in reviewed studies (GOTTARDI; BRAGA, 2015). The third and last transformation is the creation of a comprehensive set of data handlers or domain specific application frameworks to support the development of the system.

The first and second model transformations are generic and can be performed by existing tools. However, the third transformation must be defined specifically for MOWS (GOTTARDI; BRAGA, 2016). Further details on the existing and newly created tools for MOWS development are presented in Appendix B.

5.8 MOWS Definition Language

This chapter also includes languages and tools to assist developers who wish to implement MOWS systems. MOWSDL (Model-Oriented Web Service Definition Language) was established by the author of this thesis as a complete language for specifying data structures and interfaces as required by MOWS systems. MOWSDL was initially created as a model for replacing the WSDL, therefore its name was defined as Model-Oriented-WSDL. It provides declarative structures to allow the developer to create metamodels required by these systems.

During initial attempts to explain MOWS development to our research group, it has been reported that it was complicated to follow without tools that guide the developer to create systems as intended. Therefore, new efforts on creating intermediary languages and tools were performed, despite not being mandatory for proper development. Tool support for the language was made available and has been evaluated as part of the project presented within this thesis.

This explains why MOWSDL was created. It is used to represent the required definitions for metamodeling, schema and service interfaces. This avoids the need of manually editing Ecore, WSDL and XSD files, which are mandatory when developing SOAP implementations of MOWS. MOWSDL also became a common language to develop REST implementations.

Similarly to WSDL and WADL, MOWSDL has an XML structure. It also follows modeling principles by complying to the XMI standard. On top of that, an alternate textual representation (concrete syntax) has been defined, allowing developers to write MOWSDL in a text without XML idiosyncrasies. This concrete syntax is described herein as the recommended syntax for defining MOWSDL instances.

MOWSDL can be used to replace WSDL and WADL documents for MOWS systems. This language can provide a common SOAP and REST document interface. Therefore it is possible to generate both SOAP and REST systems, without changes. It can also be used to replace ECore Metamodels and XSD files.

Since MOWSDL is not mandatory for MOWS development, MOWS systems based on SOAP can be defined by using the traditional WSDL artifact. Despite this, a MOWSDL artifact can be automatically translated to WSDL and vice-versa, further tools for this workflow are described in Section 5.9.

MOWSDL is represented as a textual language designed to be edited by human developers. The file is commonly structured in three blocks, which are exemplified in the following subsections. The first block indicates the “imports” and “uses” directives (further described in

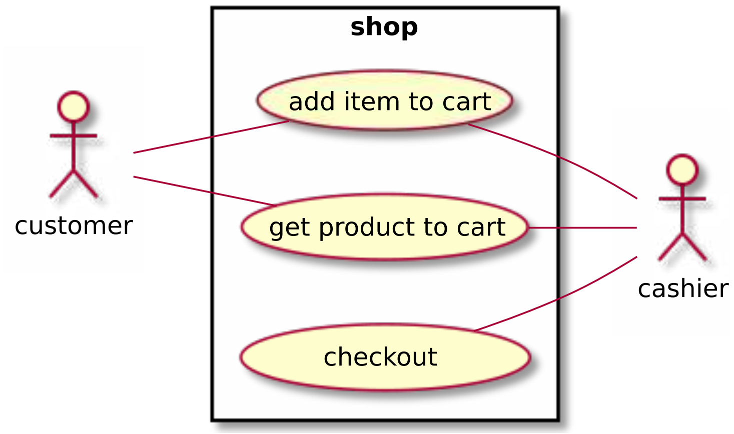
Subsection 5.8.2). Its usage is usually implied and is often not present in the provided examples of MOWSDL. The second block is employed to declare the metamodel, which is also the data schema for the MOWS system (further described in Subsection 5.8.3). The third block is composed by the interface declaration, that supposedly employs the data schema for data transmission (further described in Subsection 5.8.4). All these blocks are completely optional, which implies that an empty file is also a valid MOWSDL instance. More details about the three blocks are given in the following subsections. Grammar specification for this language is provided as part of Appendix B.

5.8.1 Retail System Example

In this section, a retail system is used as an example for developing MOWS systems using MOWSDL. This system was developed as part of the MOWSDL development manual used in experimental study (Appendix E).

This system is structured as a MOWS system following a client-server architecture. The client is responsible to manage the cart of the customer while the server stores the product information, collects the final cart managed by the client and executes the final checkout, as illustrated in the use case diagram shown in Figure 37.

Figure 37 – Retail System Use Case Diagram



Source: Created by the author

This system differs from the CWEB system described in Section 5.6 for not being simply focused on data. The MOWS system is used with interfaces provided by the server that allows clients to query product types, add them to carts and proceed to check-out by using methods provided by the interface. Both products and carts are modelled as metaclasses as shown in Subsection 5.8.3. The interface is shown in Subsection 5.8.4.

5.8.2 Import Declarations

The first declaration is used to import referenced metamodels and XSD files that are used to define the data types to be employed by the MOWS system, as illustrated in Figure 38.

Figure 38 – MOWSDL Import Example

```
using xsd("http://www.w3.org/2001/XMLSchema");
using xmi("http://www.omg.org/XMI");
import ecore("http://www.eclipse.org/emf/2002/Ecore","Ecore.xsd","Ecore.ecore");
```

Source: Created by the author

The import syntax starts either by the “uses” or “import” reserved word. It was planned to differentiate the XSD and Metamodel imports despite the fact that they are treated as synonyms in the language. Still, their applications are encouraged for two different scenarios: for the “usage” scenario, only a XSD “namespace” is provided, which indicates that it is being used by the metamodel or interface without changing their default location. Attached to the “uses” reserved word, an internal name to reference the imported file is added, which is then followed by path specifications inside parenthesis. These paths should point to the target namespace, optional source XSD file, and optional source metamodel file. For the “import” scenario, the declaration is used exclusively to provide a XSD “namespace” with full paths to their XSD file and, optionally, metamodel. Both cases start with a reserved word and then an identifier set by the developer to register the name of the schema.

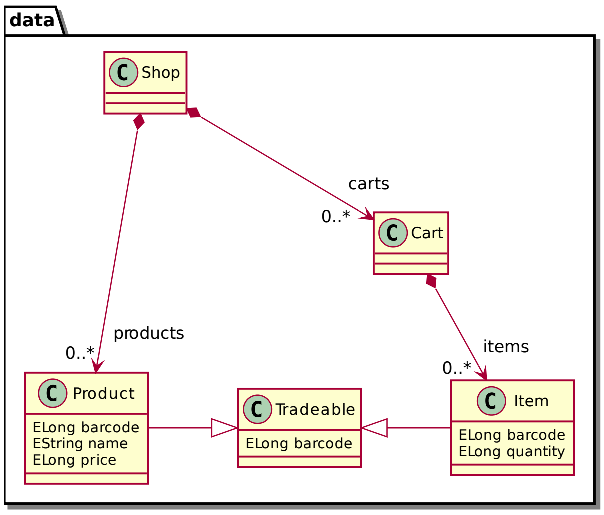
Therefore, when specifying a metamodel, its related XSD file must be provided. This is not a limitation, since XSD generators for metamodels have been provided. The EMF project has similar tools for this concern, however, they are limited when it is required to refer metamodel imports. This has led the author to implement new tools to overcome this limitation. Further details on tools are presented in Section 5.9.

5.8.3 Metamodel Declarations

The metamodel block is declared similarly to the “import” directive, as visible on the right side of Figure 39, next to a corresponding UML class diagram showing the metaclasses as classes. While names are optional for the import directives, it is required to write the names of the new metamodel, as well as its XSD file name. This information is used to output the XSD and metamodel upon transformation.

The metamodel block also includes a body for declaring metaclasses. A metaclass can inherit another metaclass (including imported), as well as their features with name and multiplicity. There are three categories of features: attributes, which take fundamental types (e.g. String) as features; references, which are simply a reference to a model object within the same model; and compositions, which indicate objects to be stored inside the current object type.

Figure 39 – MOWSDL Metamodel Example



```
metamodel store("http://store/","store.xsd","store.ecore")
{
        metaclass Shop
        {
                composition carts : Cart[0..*];
                composition products : Product[0..*];
        }

        abstract metaclass Tradeable
        {
                attribute barcode : ecore::ELong[0..1];
        }

        metaclass Cart
        {
                composition items : Item[0..1];
        }

        metaclass Product extends Tradeable
        {
                attribute name : ecore::EString[1..1];
                attribute price : ecore::ELong[1..1];
        }

        metaclass Item extends Tradeable
        {
                attribute quantity : ecore::ELong[1..1];
        }

}
```

Source: Created by the author

Imported types must be referenced by using the scope operator (‘::’) and referring the name used when registering the imported schema, metamodel or metametamodel.

5.8.4 Interface Declarations

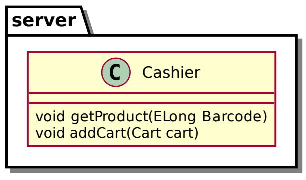
The interface block is similar to the metamodel block. This block is shown on the right side of Figure 40, next to a corresponding UML class diagram representing the same declared interface. However, the goal of declaring the interface is to establish the operations and ports of a MOWS server. One advantage of using MOWSDL for interface modeling instead of WSDL is that MOWSDL generator can infer web service bindings, whereas WSDL requires them to be specified verbosely. Another advantage is its ability to detect whether the referenced types are fundamental inside the message declarations.

Despite these differences, MOWSDL interface declaration still follows the “message” declaration similarly to WSDL 1.0, and can also be transformed into WSDL 1.1 (CHRISTENSEN *et al.*, 2001) and WADL (SCHREIER, 2011). This “message” declaration is employed to define a group of data types used for parameters, which can then be used within zero or more operations either as input or as output.

The operations are declared within a named “port”, which is the published set of operations to be provided by the server. The generators provided for MOWSDL are capable of identifying multiplicity of parameters, effectively allowing the developer to use zero or more input/output parameters.

In the end of the interface block, the name for the service name is provided, which is

Figure 40 – MOWSDL Interface Example



```
interface storews("http://store/wsdl/","store.wsdl")
{
      port storewsp
      {
            operation getProduct(BarcodeMessage):ProductMessage;
            operation addCart(CartMessage): ResponseSuccess;
      }

      message BarcodeMessage
      {
            part barcode1 : xsd::integer;
      }

      message ProductMessage
      {
            part product : store::Product;
      }

      message CartMessage
      {
            part cart : store::Cart;
      }

      message ResponseSuccess
      {
            part response : xsd::boolean;
      }

      service StoreService("storewsb","action","http://localhost:9000/");
}
```

Source: Created by the author

used to generate the server implementation. However, the interface operations would still lack proper behavior, which must be completed afterwards. Despite this limitation, the MOWSDL compiler and generation tools are capable of maintaining code blocks edited by the developers, unlike the tools provided with CXF and gSOAP.

MOWSDL is provided with an editor and compilers developed by the author, as specified in Section 5.9.

5.9 MOWS Tool-chain

After establishing MOWS systems as a specific case of MOSS, the purpose of this section is to present tools and intermediary languages available for MOWS development.

Therefore, this section contains a technical view on the tools that can be used to create MOWS systems. These tools are not mandatory for the development, since these systems follow requirements for MOSS, which rely on standards for modeling instead of forcing developers to use specific tools.

Despite avoiding this dependency, tools and languages were defined within this work to help the developers to create MOWS systems correctly, i.e. the tools would assist the developers to follow the requirements and to develop a valid system as expected.

In order to explain how these languages collaborate in the development and execution of MOWS systems, Figure 41 provides an overview on the languages and running system. The figure is divided into five levels, namely Meta, Automatic, Manual, Execution, and Persistence &

Transfer. As the figure is composed by blocks, each block position represents a relationship to neighboring blocks.

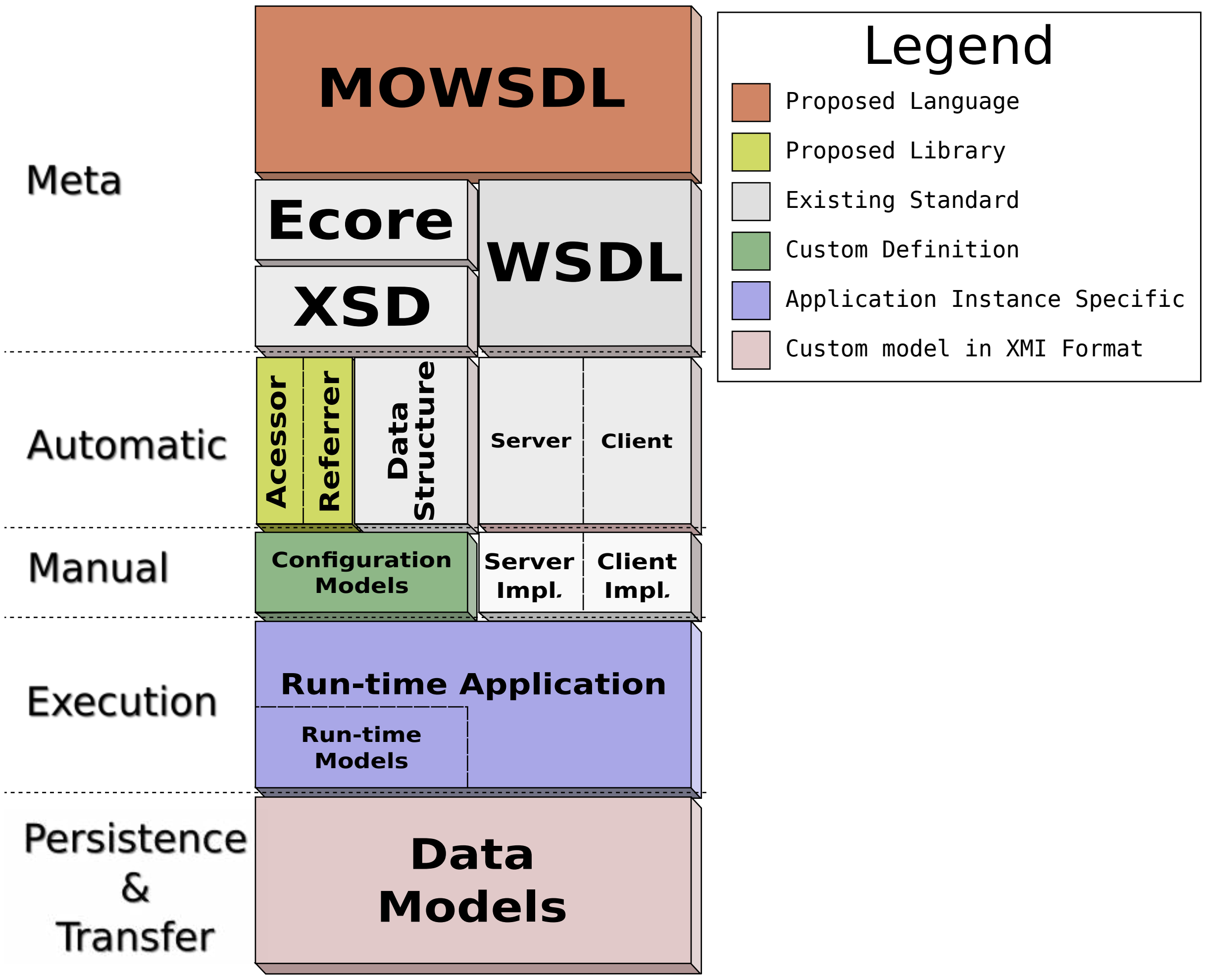
Each block in this figure is also colored for categorization. There are six categories: “Proposed Languages” marks the artifact types that are defined according to a language proposed within this project. More specifically, in this work it was presented the MOWSDL (Section 5.8). “Proposed Library” indicates the libraries that were defined within this project to assist developers when programming code to handle models with cross references. Block marked as “Existing Standard” represent software artifacts that are based on existing formats or techniques for WS development. The “Custom Definition” blocks specify which blocks are dependent to each application, therefore, they cannot be completely generalized. “Application Instance Specific” are used to point out which data formats are dependent on the platform of the running application, and they may not be compatible to other instances of the same software. Finally, “Custom model in XMI Format” specifies the models that are to be handled in the format recommended by OMG (Object Management Group, 2014). This format is both platform independent and inter-operable, allowing MOWS systems to work with other MOWS systems and modeling tools regardless of their platform (e.g. operating system and/or programming language).

When a block is placed above another block, it represents that the upper block may be used to implement the block below. For example, the meta level artifacts are all used to deal with metamodeling, schema and interface documentation without any behavioral coding. Among these artifacts, the MOWSDL language was created to cover the semantics from both WSDL and Ecore, hence its position above these artifact types.

In a MOWS system, the data schema and configuration are specified as in the meta level, which justifies this level position above all others. There are several ways to define a metamodel, as represented inside the meta level. Therefore, this level compromises the artifacts used to specify metamodels and schema definitions. Ecore is a meta-metamodel language which is part of the EMF project (Eclipse Foundation, 2015b) and is employed to define metamodels. XSD is the W3C standard for XML schema, which is employed to specify XML formats (W3C XML Working Group, 2012). WSDL is the W3C standard for WS definition (W3C Working Group, 2004). WSDL employs the XSD syntax for the data schema while adding more semantics to allow WS interface specification.

The automatic level involves source code levels that are generated automatically without requiring any manual intervention. They are divided into the data specification (below XSD) and the interface specification (below WSDL). The data structure represents the data structure code that allows developers to handle models as data. Some complex models also take advantage of helper modules. The Accessor and Referrer are helper codes that help developers to handle complex modules, as further explained within Subsection 5.9.1. The interface specification is composed by the server and client (headers), which could still require further implementation, which would be performed manually.

Figure 41 – MOWS Representation Overview



Source: Created by the author

Therefore, the manual level is composed by the implementation of the server and client. The software could also take advantage of configuration models, which are models to input variables by the software administrators to be used by the software during run-time. Common examples include the server address and maximum number of connections allowed.

The execution level represents the application at run-time. The models are used as data at run-time following Models at Run-time principles and managed by the application seamlessly without conversion.

This seamless usage of models as data allows to persist and transfer data as models completely compatible to the XMI specification (Object Management Group, 2014). Therefore, MOWS systems are capable of inter-operating with modeling tools to visualize and edit data.

Namely, there are several artifact types that the developer should be aware when developing and managing MOWS systems, as presented on Table 23.

Table 23 – MOWS Artifact List

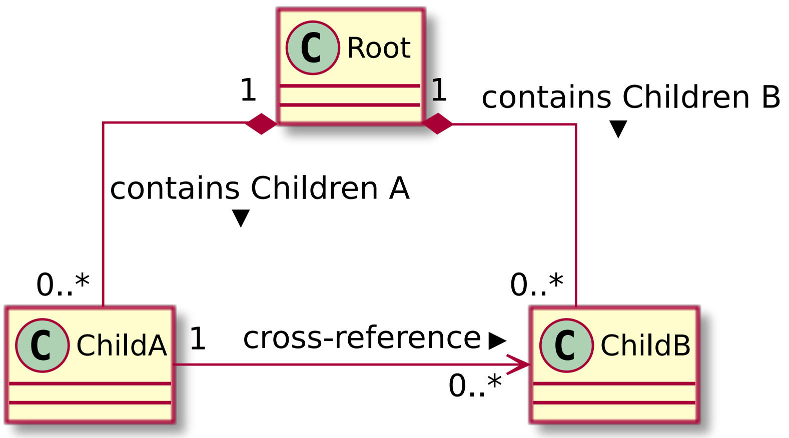
Name	Description
MOWSDL	Proposed Format
ECore	Existing Format (EMF)
metamodelDiagram	Existing Format (EMF)
XSD	Existing Format (W3C)
AcRefCode	Proposed Format
DataCode	Specific Format (example: JAX based)
WSDL	Existing Format (W3C)
modelInstance	Proposed Format (example: XMI based)
InterfaceCode	Existing Format (example: JAX-WS or JAX-RS)
modelObjectDiagram	Specific Format
Application	Specific Format

5.9.1 Accessor and Referrer Modules

The Accessor and Referrer modules are created specifically for referencing (Referrer) or dereferencing (Accessor) existing object cross-references found within models. These modules can be generated for MOSS/MOWS applications automatically.

The XMI specification defines the format of models as data trees, which cannot have closed loops. Therefore, instead of references that would cause loops, the model is structured with cross-references.

Figure 42 – Cross-Reference Example



Source: Created by the author

For instance, in Figure 42 there is a class diagram including three classes. The *Root* represents the root of the model, which is composed by two child classes (*ChildA* and *ChildB*). In order to solve this issue, as part of this project, a generator for *AcRefCode* was provided to handle the cross-references.

The referrer is responsible for creating references according to the XMI specification. For each referrer method, it takes the root of the model tree and the object to be referenced and returns the reference for the object if it is found in the tree.

The accessor has the opposite behavior, i.e., it allows to dereference the object according to a reference. Its parameters are the root of the model tree and the reference for an object, which returns the actual object (or pointer, depending on the implementation) according to the reference, if it is found.

Two generators for these codes were provided as part of this project. The first was developed to generate C++ to be included with MOWS systems created with Genivia gSOAP 2, while the second was developed in Java for Apache CXF 3. Generators for other languages could be developed in an analogous way whenever needed.

The C++ version creates classes with static methods for accessor and referrer. The accessor is capable of identifying whether the return variable is constant or not at compilation time and returns the object as a pointer. The Java version complies to objects according to JAX specification, making it detached from the Apache CXF itself. However, it does not support constant objects.

5.10 Final Remarks

This chapter concludes a view on how MDSE could be evolved into more than a generic development method, rather, it includes a specific software category that blends concepts from Model-Oriented programming with Models at Run-time, as well as adding further requirements.

In this view, this method could become a paradigm by employing models throughout analysis, design, implementation towards the final application execution. This means that we are not limiting the models within MDSE as "Models are Code", in this case, models are software, data structures and the actual data instances. These models are able to store the state representation of the running software. In our method, both code generation and code interpretation can be employed for the same final application.

This software category was validated in experimental studies and case studies, which are presented in Chapter 7 and Appendix C, respectively. A more comprehensive list of tools to assist MOWS development is presented in Appendix B.

CHAPTER 6

COMPARATIVE CASE STUDIES

6.1 Initial Remarks

MOWS systems use and transfer data natively as models. While these could be discussed as benefits for the developers who wish to edit, visualize and transform application data as models, new questions rose regarding the impact of the data formats and transfer employed by MOWS systems. Therefore, it is necessary to compare the performance of MOWS systems against traditional WS systems. This question is deemed important, since the transfer would impact not only the developers but the users and the performance of running systems.

In this chapter, we describe two comparative studies regarding the differences of a MOWS system compared to WS systems that are not dependent on modeling specifications. The first study was conducted to compare the data length (as in byte count) that is transmitted by MOWS systems against traditional WS systems (Section 6.2). The data length of transmitted messages affects the transmission time, thus, affecting the system performance. This study is further extended by employing mathematical analysis, which allows us to identify the implementation differences algebraically, without depending on a specific data structure. The second study is an analysis that compares the compatibility of formats among a set of MOWS and traditional WS systems. (Section 6.2). Finally, in Section 6.4, there are the conclusions for this chapter.

6.2 Data Length Study

6.2.1 Objectives

The objective of this study is to compare and analyze how MOWS features impact the message size and transmission. This comparison is intended to analyze the data length of messages exchanged by systems that adhere or not to the MOWS requirements.

6.2.2 Method

The method is related to the study design defined for this comparison. In order to perform the intended comparisons, we have selected a context of possible systems and their applications, varying protocol (or architectural recommendation) and whether or not MOWS is employed.

The systems were compared in pairs (all possible combinations). These systems are then compared regarding the length of the produced data.

6.2.2.1 Context Selection

For the context selection, we have selected possible data schema specifications for alternate implementations of the WS systems. For the sake of exemplification, the examples were based upon the implementations of Section 5.6, i.e., variations of the CWEB sample system were used to develop these systems.

On Table 24, there are six possible systems that we refer as applications. All the presented systems were implemented by the author by using Java JAX and/or Apache CXF according to the design specified in Subsection 5.6.4. All the applications are XML based and we compare the lengths of generated XML files. Among these, the described MOWS systems are represented by letters ‘D’ and ‘E’, based on REST recommendation and SOAP (protocol), respectively.

The REST-Style HTTP WS systems were developed without WSDL. This is important to point out because WSDL imply the schema reference, which affects the data length. In this study, we have considered that it is not possible to create a SOAP WS without a WSDL and without a schema. Therefore, on Table 24, there is no SOAP WS without a schema. The letter ‘F’ is only provided to allow comparing the default format exported by a Model Editor Tool.

Table 24 – Alternate Implementations for the CWEB System

System Name	Description	Application
‘A’	Without Schema	REST-Style HTTP WS
‘B’	Based on a Generic Schema	REST-Style HTTP WS
‘C’	Based on a Generic Schema	SOAP Compliant WS
‘D’	Metamodel based (MOWS)	REST-Style HTTP WS
‘E’	Metamodel based (MOWS)	SOAP Compliant WS
‘F’	Model Editor Tool	Modeling Tool

6.2.3 Operation

The operation of analyzing the data lengths is described within this subsection. Since XML files are also text files, these files were split into lines of text. The possible lines of all XML files generated by the studied systems were combined into Table 25. Each line is referred by a number (first column). Lines are also linked to the system which generates them and their lengths

Table 25 – Comparison of Exported XML

#	XML File	System						Length	Description
		A	B	C	D	E	F		
1	<?xml version="1.0" encoding="UTF-8"?>	P	P	P	P	P	P	38	XML header with encoding.
2	<pim:ContactList	P	P	P	P	P	P	$2+p+t$	Contact list root node.
3	• xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"			P		P		59	Schema for envelope used by SOAP WS.
4	• xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"			P		P		59	Schema for encoding used by SOAP WS.
5	• xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"		P	P	P	P	P	54	Schema for XML Schema.
6	• xmlns:xsd="http://www.w3.org/2001/XMLSchema"		P	P	P	P		45	Schema for XML Schema.
7	• xmlns:xmi="http://www.omg.org/XMI"				P	P	P	35	Schema for Models.
8	• xmi:version="2.0"						P	18	Version of Model Format.
9	• xmlns:pim="http://generic.pim/xsd"		P	P				$10+p+x$	Generic PIM Schema.
10	• xmlns:pimws="http://generic.pim/wsdl"			P				$10+s+w$	Generic PIM Interface.
11	• xmlns:pim="http://model.pim/xsd"				P	P		$10+p+x$	Schema for PIM Metamodel.
12	• xmlns:pimws="http://model.pim/wsdl"					P		$10+s+w$	Interface for PIM using Metamodel.
13	>	P	P	P	P	P	P	1	End of node attribute declaration.
14	<contacts name="Alice" address="alice@example.com"/>	P	P	P	P	P	P	$m \cdot (3+l+a \cdot (4+n+c))$	Object data for contact.
15	<contacts name="Bob" address="bob@example.com"/>	P	P	P	P	P	P	same as above; included in m	Object data for contact.
16	</pim:ContactList>	P	P	P	P	P	P	$4+p+t$	End of root node.

in characters.These lines are marked with either 'P' (P for Present) or ' ' (blank for absent), to define if the line is present in the data from a given system. There is also a description for each line. Thanks to the structure of XML, the declarations do not need to be declared separately into lines as shown on the table; they are shown like this on the table to ease the visualization. Therefore, unnecessary line breaks are not counted as part of the length.

The objective of providing the length column on Table 25 is to allow calculating the affected difference on the file length. There are variables for flexibility: let p be the length of the data schema prefix (namespace alias); let t be the length of the name of the root node; let x be the length of the data schema target namespace; let s be the length of the WSDL prefix; let w be the length of the WSDL target namespace; let m be the amount of objects inside the root node; let l be the length of list name; let a be the amount of attributes of the object; let n be the length of attribute name of the object and let c be the length of attribute value of the object. These variables are all positive integers.

The lengths for the node attributes, including schema references, are incremented by one to include the required separator between them, e.g. space, which is also considered as one character and affects the length of the file.

The first line (Line 1) of the XML file is constant for all XML files we have analyzed. Its length is the result of the count of characters, which is always 38. This happens because all systems employ standard XML headers. It is important to declare that the encoding is not important for this study, it simply needs to be the same throughout the analyzed systems.

The other lines may vary depending on the schema structure. For example, Line 2 contains the beginning of root node declaration. Every root node starts with the '<' symbol, followed by the name of the node type as specified by the schema. The ':' is the separator for the alias of the namespace, which is declared as part of the *xmlns* schema declarations. After this separator, it is followed by the actual root type name. Therefore, we need to count the '<' and

':' symbols (constant length of 2), plus the namespace alias (p) and plus the root type name (t). This results into $2+p+t$, as written on the table.

Since we are omitting the length of unnecessary line breaks, we have added a leading '•' symbol whenever a separator is required before the current declaration. Then, on Lines 3 to 12, it is mandatory to include a separator, e.g. space, that separates the declaration from the one on previous line. This separator also takes one character in length, and is therefore counted as part of the constant. For example, on Line 8 there are 17 characters plus one for separating the previous declaration, resulting in 18.

Following this expression construction, all the other rows start with the number of constant characters (including the separators and finishing characters). Whenever there are variable length declarations, they are added to the initial constant in the same order as they appear on the line.

It is important to mention that the lengths were calculated based on character count, not the actual data size. To make it simpler, we are considering that every character has the same data size. This is easily done by restricting to characters that fit into one byte or converting the 'UTF-8' encoded files into 'ASCII' encoded files (W3C Working Group, 2008) with the exact same length. This also conforms the same result of our analysis: coincidentally, both encoding names have the same length in characters.

As the application columns contain 'P' (Present) or 'N' (Not present) indicating whether the line is present in the output of the given system, its possible to come up with the resulting lengths for each system by summing every expression that is marked by a 'P'. The comparison of these lengths are presented within the following subsection.

6.2.4 Results

The results for the length comparison are presented within this section. On Table 26 we compare the lengths of the files. The columns indicate the data lengths which are subtracted from the data lengths at the rows, therefore, each cell contains the length difference. In order to calculate the difference, first, we had to consider Table 25 and take the sum of every line that contains 'P' from there. Then, the difference was calculated by taking the sum of the system referred by the column minus the system referred by the row. The resulting difference was then written into the cell related to these columns and rows. In other words, let S_c be sum of the lengths of application referred by the column and let S_r be sum of the lengths of data referred by the row. Each cell is calculated by the expression $S_c - S_r$.

For instance, let A be the sum of line lengths of system A, and B the sum of line lengths of system B. The first length difference is zero, because it is the result of $A-A$, which is always zero due to arithmetic property. The second cell on the first row is the result of $B-A$.

Table 26 – Length Differences

Negative	Positive					
	+A	+B	+C	+D	+E	+F
-A	0	$109+x+p$	$237+x+w+s+p$	$144+w+p$	$272+x+w+s+p$	107
-B	$-109-x-p$	0	$128+w+s$	35	$163+w+s$	$2+x+p$
-C	$-237-x-w-s-p$	$-128-w-s$	0	$-93-w-s$	35	$-130-x-w-s-p$
-D	$-144-x-p$	-35	$93+w+s$	0	$128+w+s$	$-37-x-p$
-E	$-272-x-w-s-p$	$-163-w-s$	-35	$-128-w-s$	0	$-165-x-w-s-p$
-F	-107	$-2-x-p$	$130+x+w+s+p$	$37+x+p$	$165+x+w+s+p$	0

6.2.5 Discussion

Despite providing the differences for every pair of presented applications, there are a few points to consider when interpreting the resulting expressions.

The primary objective is to identify the differences between using MOWS WS systems to non MOWS WS systems that use the same XML structure. This means that for the length comparison, we are disregarding the comparisons using different protocols.

We did not add the SOAP Envelope length, since our objective was not to compare the overhead of employing SOAP for transmission when compared to raw XML transmission employed by REST-Style WS systems. Therefore, any other overhead caused by the protocol would then be eliminated, as we are only considering the differences between the same protocol usage.

The mathematical analysis was performed to establish a comparison that is not tied to a specific schema or metamodel. It is also important to report that the extra metadata information does not affect the content of data, only the header (XML root node).

Following the objective of MOWS comparison, we need to compare the applications referred as letters D and E to applications that represent non MOWS counterparts that implement the same protocol. Therefore, we have to compare them to B and C, respectively.

Namely $D-B$ and $E-C$ (both) result in a difference of '35', which is seen on Table 26. This is a constant difference because the length of the actual data would be the same in both cases; they only differ on the header, which has a fixed length.

The length comparison was performed by analyzing uncompressed XML. It is important to mention that the evaluated WS platforms also support data compression, which could make this overhead even smaller.

Regarding the results, it is confirmed that MOWS have an increased data overhead, which is due to the meta-metamodel reference declaration. Since only this declaration affects the length, it can be generalized to any system using a similar data structure. However, this declaration is redundant, since it can be found recursively inside the metamodel specification. It has been tested that it is possible to remove this declaration by developing specific metamodel to schema transformers.

The removal of meta-metamodel declarations does not break compatibility with the tested MOWS implementations. Still, its removal might cause trouble on some WS implementations, specially, in case of model instances that use meta-metamodel data types. MOWS usage is discouraged for some cases, yet, the constant overhead might be tolerable for many applications. MOWS systems employ DSL or ontology languages to represent the data and knowledge, allowing humans and machines to communicate in a common language.

6.2.6 Threats to Validity

The advantage of performing analysis on the data outputs based on the XSD is that it is possible to prove whenever it matches its specification. This advantage eliminates any threats related internal validity, since the analyzed data is valid for the actual implementation. The threats related to measure validity are also eliminated because we have performed a mathematically exact analysis on the length of the messages. It was possible to identify and count every text character transmitted in every presented application.

However, there is still a threat to external validity because of a possible difference between the real world applications and the study applications. It is possible that a real world application would employ a very different data structure. To mitigate this threat, the algebraic expressions were defined with the intent of making the names of types abstract, i.e., we compared the different systems using the same set of paired type names, thus, avoiding subjectivity on the comparison. Therefore, the performed comparisons were paired between similarly structured applications that employ different technologies.

6.3 Data Format Compatibility Study

6.3.1 Objectives

The objective of this study is to compare and analyze how different MOWS implementations impact the data format compatibility. In this manner, it is compared the compatibility between pairs of these systems with the intent of identifying if they can be employed in conjunction, i.e., we verify the interoperability of different WS systems that were developed either with or without MOWS requirements. We also verify their compatibility with modeling tools.

6.3.2 Method

This study was designed similarly to the length comparison study (Section 6.2) The same possible systems were used and their output data formats were captured.

These systems have been compared in pairs to verify which system is able to read the data output from others by analyzing all possible combinations. For the first objective, we compare the length of the produced data. For the second objective, we compare whenever applications can read other's outputs.

6.3.2.1 Context Selection

The selected systems were the same used on Section 6.2.2. Therefore, the systems as represented on Table 24 are also used and referred with letters from 'A' to 'E'. It is important to mention that the model editor tool represented by the letter 'F' is used to indicate which formats are compatible to Model Editor Tools.

6.3.3 Operation

The operation of analyzing the data formats is described within this subsection. We have considered the proposed formats and identified the compatibility of exporting data by using each system. Besides the fact that they were designed with the objective of following the same XML structure, they differ on the schema dependencies. Each system was compared in pairs and the results are presented as a matrix.

6.3.4 Results

Table 27 – Format Compatibility Matrix

Reader	Writer					
	'A'	'B'	'C'	'D'	'E'	'F'
'A'	True	True	True[1]	True	True[1]	True
'B'	False	True	True[1]	False	True[1]	False
'C'	False	False	True	False	True[1]	False
'D'	False	False	False	True	True[1]	True
'E'	False	False	False	True	True	True
'F'	False	False	False	True	True[1]	True

The columns of Table 27 represent the application that exported the XML file, where each row indicates the application reading the exported XML. Cells marked as "True" indicate that the application accepts the written file. "False" indicates that the application reading the file would refuse to accept it as valid data because of schema mismatch, even though the XML structures are the same. Since the application 'A' ignores the schema, it might accept data from any application, however, it could wrongly accept data not intended for the system, thus, causing defects.

Finally, cells marked as "True[1]" indicate that the writer system also employs the SOAP schema for data transmission. The SOAP schema reference itself is not problematic for the importing application, as long as the data is not inserted into a SOAP Envelope.

6.3.5 Discussion

This comparison study indicates how the different implementations of MOWS handle the data formats.

Regarding the SOAP envelope, it is important to out that it only affects the case when it is desired to capture the raw data transmitted between servers and clients using SOAP. In that instance, it may be required to filter out the SOAP Envelope to be able to use the data with modeling tools. Still, clients and servers of a SOAP WS system are capable of handling files without the SOAP envelope. It is important to remind that the compatibility matrix is only used to compare the data format, i.e., file handling, not how it is transmitted over the web. Therefore, employing a common protocol is still required for data transmission.

The matrix also reveals an important advantage of MOWS, which is the native and seamless compatibility to modeling tools. This means that modeling tools can be employed to edit and visualize data handled by MOWS systems without requiring conversion.

We have also identified that non-MOWS systems can be compatible to MOWS systems, which indicates that MOWS do not break principles meant for traditional WS systems. This study also shows that MOWS systems are the only with reciprocal compatibility to modeling tools.

6.3.6 Threats to Validity

This study included all combinations of MOWS with REST and SOAP. The advantage of performing the analysis of all combinations eliminates threats related to internal validity, since the analyzed data is valid for the actual implementations. The threats related to measure validity are also eliminated because all the possible results for reading the formats are also presented exhaustively

However, there is still a threat to external validity because of a possible difference between the real world applications and the study applications. It is possible that a real world application would require different protocols, encoding standards, and data formats. This threat was mitigated by following format standards as recommended by W3C (W3C Working Group, 2004) and OMG (Object Management Group, 2014).

6.4 Final Remarks

In this chapter, two comparative studies were presented to compare the MOWS systems to traditional WS systems. These systems provide both advantages and disadvantages, but it is not argued that they should replace all systems. It was possible to show that the increased length disadvantage (overhead) is constant and may not be significant for many systems. Besides analyzing the impact of MOWS systems, Chapter 7 contains experiments to evaluate developers, other than the author of this thesis, to build MOWS systems.

CHAPTER

7

EXPERIMENTAL STUDIES

7.1 Initial Remarks

In Chapter 5 it has been presented the requirements of MOSS/MOWS. These systems require programmers to work on metamodels instead of traditional source code, which could be a complex task for common developers. As these systems rely on metamodel as design, it became important to evaluate if developers are able to comprehend and create metamodels in software development with and without proposed tools. As discussed in the empirical studies presented in Chapter 4, it has been suggested that higher abstraction levels could be more difficult for developers. Indeed, the surveys discussed by Teppola, Parviainen and Takalo (2009) and Whittle (2013) also list related challenges faced by MDSE developers.

Therefore, experimental studies have been conducted to evaluate if common developers would be able to implement required metamodels for MOWS systems by using the proposed tools or existing frameworks. In this chapter, two separate sets of experimental studies are presented on this context. It is important to establish that these studies are focused on evaluating MOWS development and are unrelated to the studies discussed in Chapter 4. The first experiment evaluates a metamodel language created for MOSS/MOWS specification (Section 7.2) and the second experiment evaluates coding with Java Programming language as required for implementing MOWS systems in that platform (Section 7.3). Finally, in Section 7.4 there are the conclusions for this chapter.

7.2 Data Structure Verification Experiment

7.2.1 Objectives

In this section it is presented an experiment which was conducted to evaluate whether the MOWSDL (language introduced in Section 5.8) was capable of representing metamodels as well as Java code.

This was planned to verify the required efforts to define metamodels required by MOWS development performed by different developers. By evaluating MOWS, we are also evaluating a specific kind of MOSS. In this manner, this study includes activities to read models and code in order to identify its semantics.

Throughout the research history of this project, one major concern was whether developers would be able to define metamodels as required by MOWS systems. Since MOWSDL is used to define metamodels, the ability of developers to use it would further indicate its feasibility for real software production. This study was planned to evaluate the ability of developers to implement MOWS systems.

Therefore, the rationale behind comparing Java and MOWSDL for data schema definition was beyond simply identifying whenever a design language or a programming language was preferable for activities; it was intended to verify if programmers could accept MOWSDL as a roughly equivalent to Java when identifying classes, relationships, properties and overall class design, since this is required for metamodel definition.

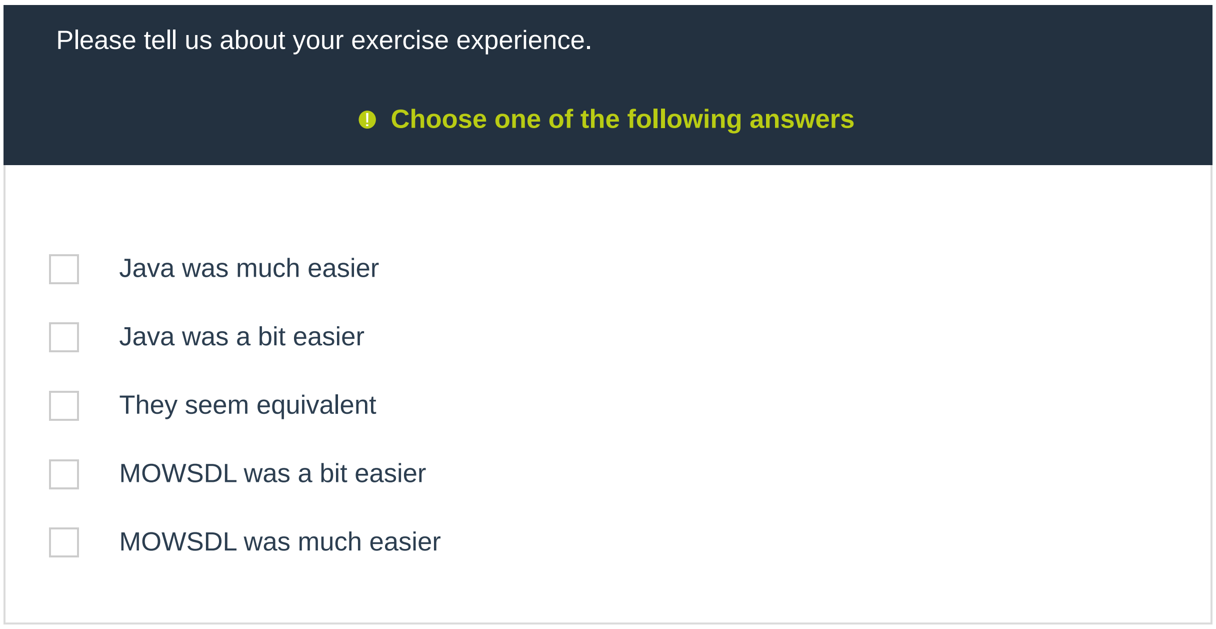
7.2.2 Method

The tasks that the participants were expected to perform involve identifying the semantics from textual MOWSDL model and Java code, referred herein as study artifacts. Considering that developers are required to identify the concepts (semantics) that are represented by modeling or code artifacts, this study was designed with a training session plus four tasks that take the two kinds of artifacts in pairs. These tasks were planned after what was required to declare when creating metamodels. The original task plan involved the identification of these concepts: Classes (very similar in both languages but kept for being the most basic task), Inheritance (removed from the study for being equal in both languages); Multiplicities (removed because Java does not have explicit multiplicity, later changed to list identification); Attributes (merged with relationship task) and Relationships. Besides identifying these concepts using either language, it was also planned to verify if the participants could find the class diagram that matches each textual language instance. These artifacts are used to assess which language is more adequate for identifying the required semantics from languages.

Table 28 – Study Design

Task	Description	Java Sub-tasks	MOWSDL Sub-tasks
Training	General Training	1	1
First	Class Counting	4	4
Second	List Counting	4	4
Third	Relationship Counting	4	4
Fourth	Diagram Matching	4	4

Figure 43 – Task Feedback Question



Please tell us about your exercise experience.

Choose one of the following answers

- ☐ Java was much easier
- ☐ Java was a bit easier
- ☐ They seem equivalent
- ☐ MOWSDL was a bit easier
- ☐ MOWSDL was much easier

Source: Created by the author

This study is composed by a training section followed by the four tasks, as presented on Table 28. This table is composed by rows (tasks) and four columns: namely, the task ordinal, the description for each task, the number of Java sub-tasks in the task and the number of MOWSDL sub-tasks in the task. The training task included text and diagrams to instruct the participants on how to detect the required semantics during the execution of the tasks.

The first task involves identifying classes in the provided artifacts. The second task aims at identifying lists, including properties and relationships between classes with multiplicity higher than '1'. For the third task, the participants were asked to identify relationships between classes (all multiplicities). During the final task, the participants were requested to identify the UML class model that represented the provided artifact. After each task, the participants were allowed to answer feedback questions, as in Figure 43. After all tasks were completed the participants could also enter text as to provide qualitative comments.

The dependent variables are the variables collected by the researchers from the study execution. Table 29 contains the variables which are used for hypothesis testing. There are two types of variables: the number of correct answers and the time taken to complete each sub-task. This table contains five columns. The first column contains the variable name, which is used to reference this variable throughout the study. The second column contains a textual description for the variable. The third column is used to indicate the type of the variable. The fourth column shows the valid range for the variable. The fifth column is specifies the minimum possible value for the variable, which is used to measure its precision. This single table is composed by variables for all tasks that have been measured.

Table 29 – Study Dependent Variables

Name	Description	Type	Range	Precision
CCJ	Correct Java	Count (Ratio)	0 to 4	1 count
CCM	Correct MOWSDL	Count (Ratio)	0 to 4	1 count
TCJ	Time Completion Java	Real (Ratio)	0 or greater	1/100 second
TCM	Time Completion MOWSDL	Real (Ratio)	0 or greater	1/100 second
CLJ	Correct Java	Count (Ratio)	0 to 4	1 count
CLM	Correct MOWSDL	Count (Ratio)	0 to 4	1 count
TLJ	Time Completion Java	Real (Ratio)	0 or greater	1/100 second
TLM	Time Completion MOWSDL	Real (Ratio)	0 or greater	1/100 second
CRJ	Correct Java	Count (Ratio)	0 to 4	1 count
CRM	Correct MOWSDL	Count (Ratio)	0 to 4	1 count
TRJ	Time Completion Java	Real (Ratio)	0 or greater	1/100 second
TRM	Time Completion MOWSDL	Real (Ratio)	0 or greater	1/100 second
CDJ	Correct Java	Count (Ratio)	0 to 4	1 count
CDM	Correct MOWSDL	Count (Ratio)	0 to 4	1 count
TDJ	Time Completion Java	Real (Ratio)	0 or greater	1/100 second
TDM	Time Completion MOWSDL	Real (Ratio)	0 or greater	1/100 second

The hypotheses tables are divided according to the task they are related. These tables are composed by three columns. The first column contains the name given to the hypothesis. The second column has the description for the hypothesis. The third column documents the logic/arithmetic expression used for comparison, which is considered under hypothesis testing, i.e. they are applicable as null or alternate depending on the statistical test used.

Table 30 contains the hypotheses defined for the possible outcomes for the class identification task.

Table 30 – Study Hypotheses (Class Identification)

Name	Description	Predicate
Class Negative Count	Java is more precise for class identification	$CCM - CCJ < 0$
Class Zero Count	Languages are equivalent when identifying classes	$CCM - CCJ \approx 0$
Class Positive Count	MOWSDL is more precise for class identification	$CCM - CCJ > 0$
Class Negative Time	Java takes more time for class identification	$TCM - TCJ < 0$
Class Zero Time	Languages are equivalent when identifying classes	$TCM - TCJ \approx 0$
Class Positive Time	MOWSDL takes more time for class identification	$TCM - TCJ > 0$

Similarly, Table 31 contains the hypotheses defined for the possible outcomes for the list identification task.

The hypotheses for the relationship identification task are presented on Table 32.

Finally, the possible hypotheses that are considered as outcomes for the diagram matching

Table 31 – Study Hypotheses (List Identification)

Name	Description	Predicate
List Negative Count	Java is more precise for list identification	$CLM - CLJ < 0$
List Zero Count	Languages are equivalent when identifying lists	$CLM - CLJ \approx 0$
List Positive Count	MOWSDL is more precise for list identification	$CLM - CLJ > 0$
List Negative Time	Java takes more time for list identification	$TLM - TLJ < 0$
List Zero Time	Languages are equivalent when identifying lists	$TLM - TLJ \approx 0$
List Positive Time	MOWSDL takes more time for list identification	$TLM - TLJ > 0$

Table 32 – Study Hypotheses (Relationship Identification)

Name	Description	Predicate
Relationship Negative Count	Java is more precise for relationship identification	$CRM - CRJ < 0$
Relationship Zero Count	Languages are equivalent when identifying relationships	$CRM - CRJ \approx 0$
Relationship Positive Count	MOWSDL is more precise for relationship identification	$CRM - CRJ > 0$
Relationship Negative Time	Java takes more time for relationship identification	$TRM - TRJ < 0$
Relationship Zero Time	Languages are equivalent when identifying relationships	$TRM - TRJ \approx 0$
Relationship Positive Time	MOWSDL takes more time for relationship identification	$TRM - TRJ > 0$

task are presented on Table 33.

7.2.3 Operation

7.2.3.1 Context Selection

The invitations to participate in the experiment were sent via different mail-lists, including professionals, undergraduate students and graduate students, effectively reaching 117 participants.

The participants were openly invited, however, their profile was employed for selection. The basic profile requirement was to select participants with experience on programming. Still, the participants also had the right to quit at any time, which reduced the number of participants who completed the study.

The operation was carried by employing online forms that did not reject participants without the basic selection requirements. This allowed 42 participants to complete the form, however, 33 were selected after discarding the participants who lacked programming skills.

The list of selected participants are shown on Table 34. The participants were treated as

Table 33 – Study Hypotheses (Diagram Matching)

Name	Description	Predicate
Diagram Negative Count	Java is more precise for diagram identification	$CDM - CDJ < 0$
Diagram Zero Count	Languages are equivalent when identifying diagrams	$CDM - CDJ \approx 0$
Diagram Positive Count	MOWSDL is more precise for diagram identification	$CDM - CDJ > 0$
Diagram Negative Time	Java takes more time for diagram identification	$TDM - TDJ < 0$
Diagram Zero Time	Languages are equivalent when identifying diagrams	$TDM - TDJ \approx 0$
Diagram Positive Time	MOWSDL takes more time for diagram identification	$TDM - TDJ > 0$

anonymous, the provided participant numbers are just for identifying them inside this chapter. This table includes columns for the participant number (P), their occupation (Occupation), programming experience in years (Exp.) and the training material language (Language) which is available in English (en) and Brazilian Portuguese (pt-BR).

7.2.3.2 Preparation

The participants received training material as part of the preparation.

7.2.3.3 Instrumentation

The participants were required to read the training material, which included all tasks with preceding examples. The examples illustrated how to identify classes, lists, relationships and match diagrams for both languages.

The instruction page only included vague description on the study objectives, thus avoiding the participants expectations from affecting the actual objective of the study.

In this manner, it was described that the study objective was simply to compare both languages. The following sentences were also added to further divert from these expectations: *"We are not here to push them for adoption, we just want to identify whenever each kind is most recommended for each activity." "For the programming language, we are studying the Java programming language. The declarative language is named MOWSDL. It was created to bridge a semantic gap between programming languages and design languages."*

The training included examples of the artifacts to be used within the tasks. This included the diagram in Figure 44, the Java code in Figure 45, and the MOWSDL code in Figure 46. Huge warnings stating that "this is simply an example" were added since it was reported that

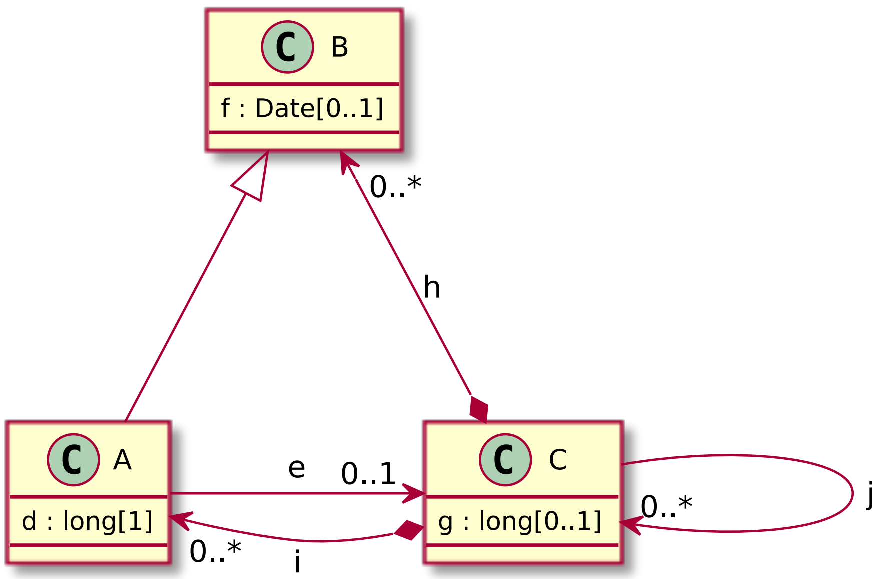
Table 34 – Participant List and Profile

P	Occupation	Exp.(years)	Language
1	Graduate Student	3	pt-BR
2	Graduate Student	6	pt-BR
3	Graduate Student	4	pt-BR
4	Graduate Student	5	en
5	Graduate Student	5	pt-BR
6	Graduate Student	2	pt-BR
7	Graduate Student	3	pt-BR
8	Graduate Student	10	pt-BR
9	Professional Developer & Systems Analyst	17	pt-BR
10	Undergraduate Student & Professional Developer & Systems Analyst	4	pt-BR
11	Professional Developer & Systems Analyst	8	pt-BR
12	Professional Developer & Systems Analyst	8	en
13	Tester	4	pt-BR
14	Professor & Professional Developer & Systems Analyst	10	pt-BR
15	Professor	10	pt-BR
16	Software Support	6	pt-BR
17	Professor & Professional Developer	22	pt-BR
18	Software Support & Researcher	2	pt-BR
19	Professor	12	pt-BR
20	Professional Developer & Systems Analyst & Tester	20	pt-BR
21	Professional Developer & Systems Analyst & Software Support & Researcher	5	en
22	Software Support & Researcher	4	pt-BR
23	Professor	20	pt-BR
24	Professor	10	pt-BR
25	Professor	10	pt-BR
26	Professor	30	pt-BR
27	Professor & Researcher	26	pt-BR
28	Professor & Researcher	10	pt-BR
29	Systems Analyst	8	pt-BR
30	Teacher & Systems Analyst	7	pt-BR
31	Graduate Student	13	pt-BR
32	Professor & Researcher	12	pt-BR
33	Professor	20	pt-BR

some participants during a pilot study tried to answer the example questions. Some of them complained to the researchers that there were “non-functioning buttons”.

The instructions text included instructions on how to identify the classes, properties and relationships. The relationships and properties had proper instructions on how to identify each kind of multiplicity, i.e. [0..1] indicates 0 or 1; [1] indicates always 1; [1..*] indicates at least one (or more); [0..*] indicates any natural number.

Figure 44 – UML Example used in training introduction



Source: Created by the author

Following the diagram, a design report presenting the names and count of Classes, Attributes (Properties), Compositions, References (Association links) and Generalizations (Inheritance) was presented, as shown on Table 35.

Table 35 – Design report for class diagram used in training

Classes	Attributes	Compositions	References	Generalizations
3 { A, B, C }	3 { d, f, g }	2 { i, h }	2 { e, j }	1 { A B }

This is the most important knowledge required to answer the questions of the tasks, since they are focused on identifying the (design) semantics of the languages. Therefore, language constructs that are unrelated to the tasks (i.e., classes with attributes, lists and relationships) were not required during the study and were omitted, since this study only focuses on visible and common semantics.

All classes, properties and relationships had single letter names and were randomly generated. These artifacts were generated in triples (UML class diagram, Java code and MOWSDL text model) to represent the same intended semantics to be identified by participants, however, their declaration orders (i.e. the order which classes appear on page) were changed, which does not impact the semantics.

Besides the randomness, they were fixed throughout all participants to avoid inserting new variables. This change on order and simple letter names were used as planned to make it harder to the participants to remember the previously used artifacts. Indeed, all participants to

Figure 45 – Java Example used in training introduction

A.java
```java
package model;

 class A extends B {

 B  d;

 }
```

B.java
```java
package model;

 class B {

 Date  e;

 List<C>  f;

 }
```

C.java
```java
package model;

 class C {

 String  g;

 char  h;

 List<B>  i;

 List<B>  j;

 }
```

Source: Created by the author

whom the researchers could talk to after the study did not even identify that the tasks had paired artifacts.

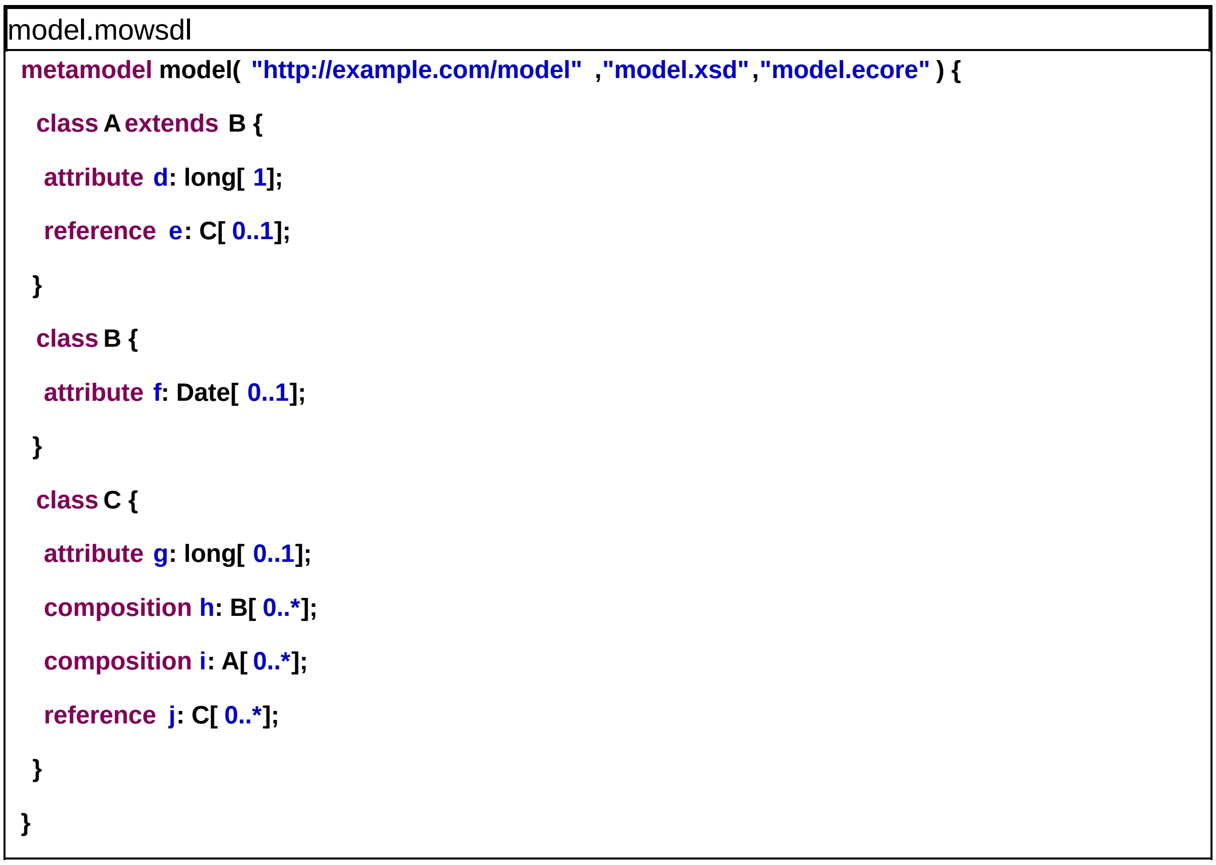
Besides the description of this instrumentation, the verbatim copy of the questions as well as the correct answers are provided as part of Appendix D. After concluding the experiment, it is not open for more participants, yet, the survey address is still provided for historical purposes ("Declarative Language Study" link is available on the referred page[1]).

The first three tasks were focused on counting elements from the text languages. As the first task was focused on class counting, it involved four Java artifacts paired with four MOWSDL artifacts, however, they were shown in random order to the participants to avoid learning that they would result into the same answer.

The simplest possible design that includes a class, a property and a relationship is provided as part of the training. This same design is illustrated in Figure 47 for UML class

[1] http://limesurvey.labes.icmc.usp.br

Figure 46 – MOWSDL Example used for training introduction

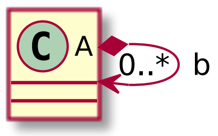


```
model.mowsdl
 metamodel model( "http://example.com/model" ,"model.xsd","model.ecore" ) {
   class A extends B {
    attribute d: long[ 1];
    reference e: C[ 0..1];
   }
   class B {
    attribute f: Date[ 0..1];
   }
   class C {
    attribute g: long[ 0..1];
    composition h: B[ 0..*];
    composition i: A[ 0..*];
    reference j: C[ 0..*];
   }
 }
```

Source: Created by the author

diagram, Figure 48 for Java code and Figure 49 for MOWSDL text. The first page of training examples includes these artifacts. It is important to observe that the highlight was shown on the figures only for the training, i.e., the actual artifacts used for the tasks did not include them. Also, all tasks had more complex class structures, ranging from 2 to 9 possible counts. The possible answer was a single decimal number to be entered, as shown in Figure 50. Null answers were treated as incorrect.

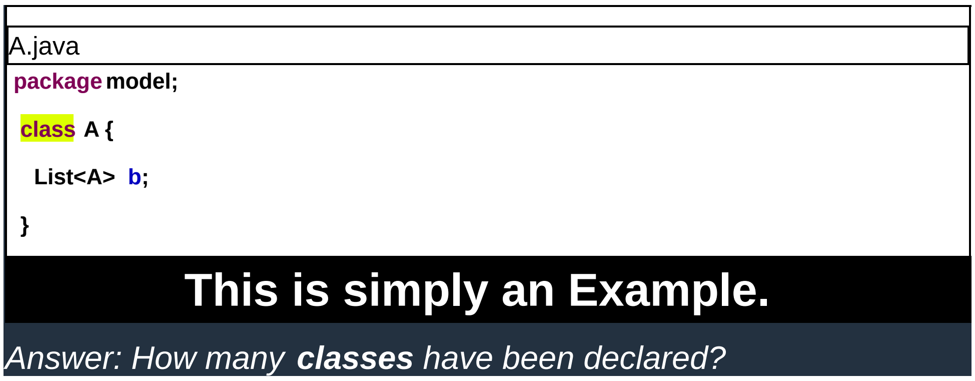
Figure 47 – Simple UML Example used in training



Source: Created by the author

Similarly to the class counting task, the second study was carried to measure how the participants could identify lists. This task followed a similar instrumentation of the previous task, including the number of artifacts. It included a new set of artifacts generated specifically

Figure 48 – Simple Java Example used for Class Counting



A.java

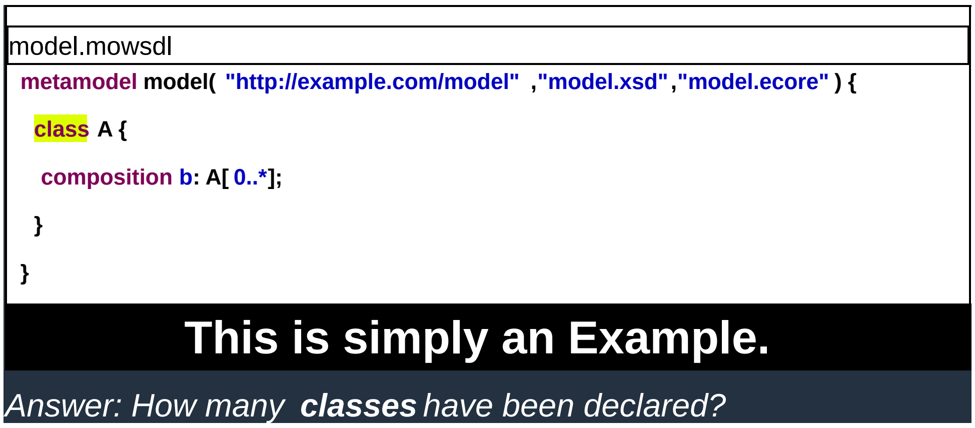
```
package model;
 class A {
   List<A>  b;
 }
```

This is simply an Example.

*Answer: How many **classes** have been declared?*

Source: Created by the author

Figure 49 – Simple MOWSDL Example used for Class Counting



model.mowsdl

```
 metamodel model( "http://example.com/model"  ,"model.xsd","model.ecore" ) {
   class A {
    composition b: A[ 0..*];
   }
 }
```

This is simply an Example.

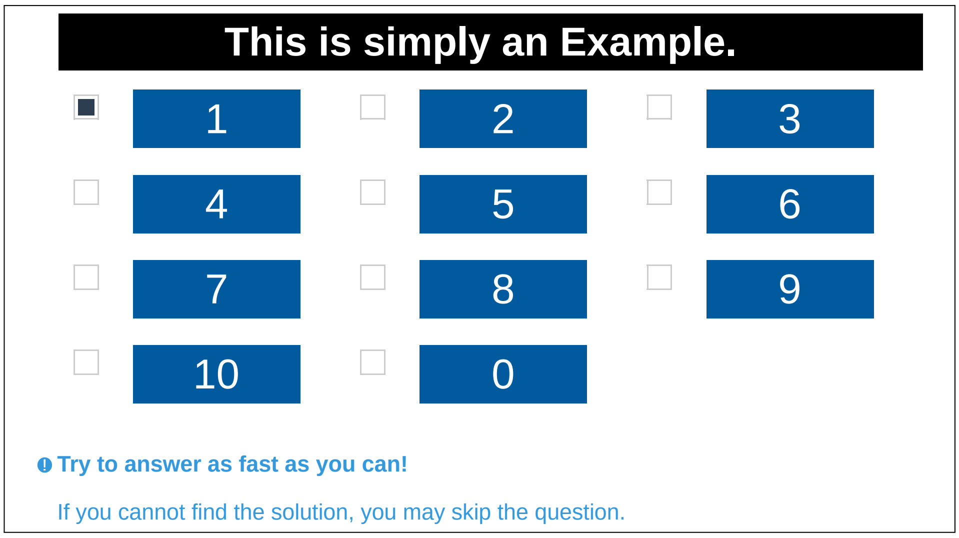
*Answer: How many **classes** have been declared?*

Source: Created by the author

for the task. The major difference is that this task is focused on identifying lists. In the case of Java code, the participants had to identify the “List” (a Java Interface) usage on code. In the case of MOWSDL, the participants had to search for [0..*] and [1..*] multiplicities. The training examples for Java and MOWSDL artifacts are provided in Figure 51 and Figure 52, respectively, but the actual study artifacts had more lists to be identified. The possible answer was a single decimal number to be entered as shown in Figure 50.

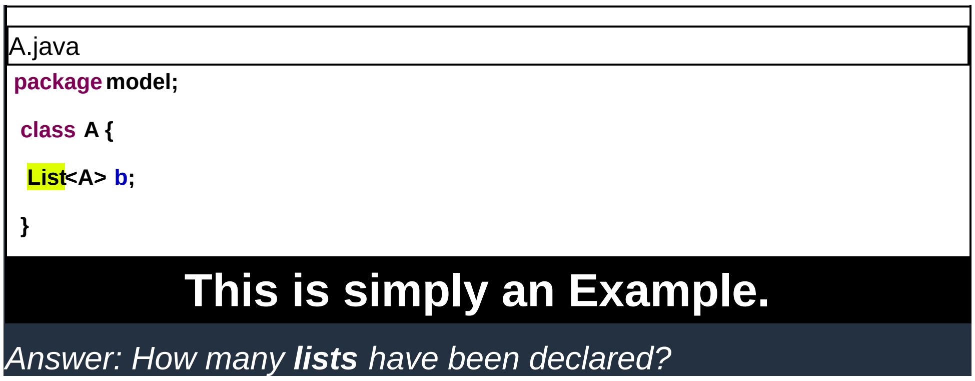
The third task was the last one involving counting. Similar to the previous tasks, it had a specifically generated set of artifacts provided in the same number. In this task, the participants had to identify the relationships that referred to a class type. In the case of Java, they had to identify the type name and find it among the classes. In the case of MOWSDL, the participants could either verify the type name or simply search for the usage of the reserved words “reference” and “composition”. The training examples for Java and MOWSDL artifacts are also provided herein as Figure 53 and Figure 54, respectively, but the actual study artifacts

Figure 50 – Buttons for Decimal Digit Answers


This is simply an Example.
1
2
3
4
5
6
7
8
9
10
0
Try to answer as fast as you can!
If you cannot find the solution, you may skip the question.

Source: Created by the author

Figure 51 – Simple Java Example used for List Counting


A.java

```
package model;
 class A {
  List<A> b;
 }
```

This is simply an Example.

*Answer: How many **lists** have been declared?*

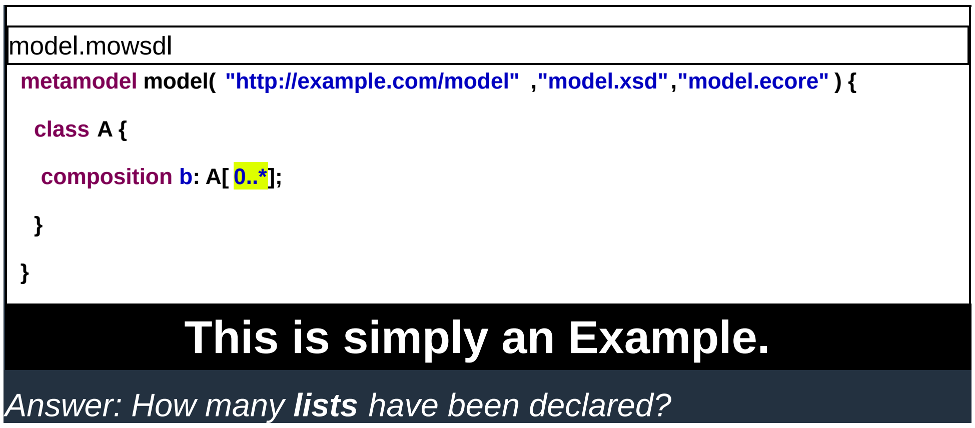
Source: Created by the author

had more relationships to be identified. The possible answer was a single decimal number to be entered, as shown in Figure 50.

The last task involved comparing a Java or MOWSDL artifact against a UML class diagram. The participants were provided a single MOWSDL or Java artifact and a list of three UML diagrams. They were asked to select the matching diagram from the list. The artifacts for this task were specifically generated by using the same class names, which makes the task more challenging, since the participants would need to verify all the properties and relationships instead of simply disregarding a diagram with different class names.

The training examples for Java and MOWSDL artifacts are also provided herein as Figure 55 and Figure 56, respectively. There is also the simple example of a UML class diagram shown in Figure 47. The actual study artifacts had three different diagrams per question with

Figure 52 – Simple MOWSDL Example used for List Counting


model.mowsdl

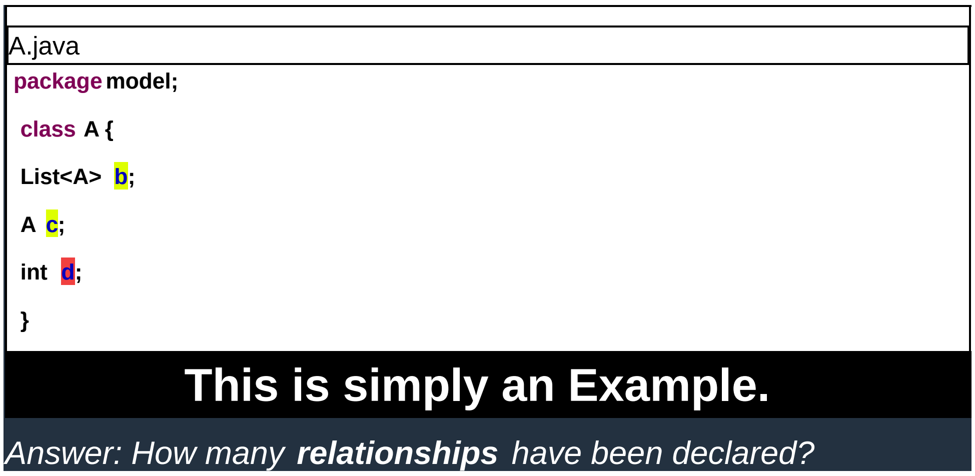
```
metamodel model( "http://example.com/model" ,"model.xsd","model.ecore" ) {
  class A {
   composition b: A[ 0..*];
  }
}
```

This is simply an Example.

*Answer: How many **lists** have been declared?*

Source: Created by the author

Figure 53 – Simple Java Example used for Relationship Counting


A.java

```
package model;
 class A {
 List<A>  b;
 A  c;
 int  d;
 }
```

This is simply an Example.

*Answer: How many **relationships** have been declared?*

Source: Created by the author

more classes and relationships.

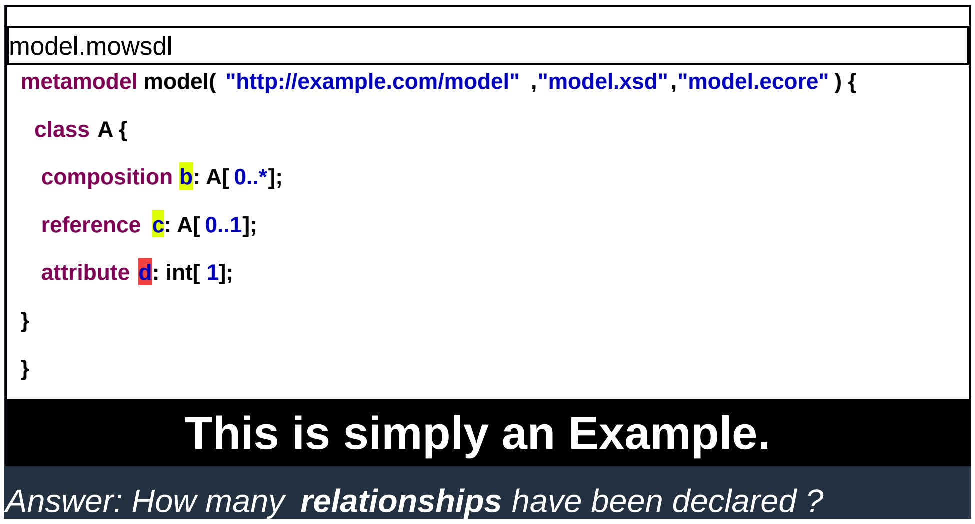
At the end of each task, the participants were also asked their opinion by using five ordinals, from first to fifth namely: (A1) “Java was Much Better”, (A2) “Java was a Bit Better”, (A3) “Seem equivalent”, (A4) “MOWSDL was a Bit Better”, (A5) “MOWSDL was Much Better”.

After these tasks, the participants were invited to provide qualitative feedback on the survey, which is further discussed in Subsection 7.2.6.

7.2.3.4 Execution

Initially, the participants had to accept a consent form and then answered a profile characterization form. The characterization form had questions regarding knowledge on programming,

Figure 54 – Simple MOWSDL Example used for Relationship Counting



model.mowsdl

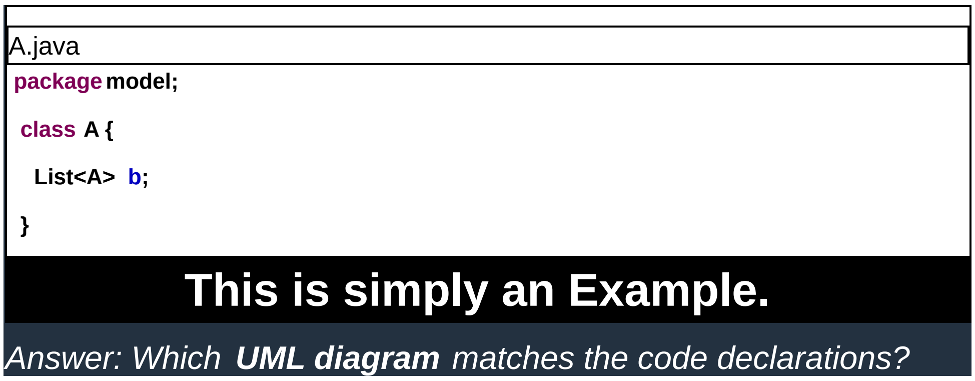
```
metamodel model( "http://example.com/model" ,"model.xsd","model.ecore" ) {
  class A {
   composition b: A[ 0..*];
   reference c: A[ 0..1];
   attribute d: int[ 1];
 }
}
```

This is simply an Example.

Answer: How many **relationships** have been declared ?

Source: Created by the author

Figure 55 – Simple Java Example used for Diagram Matching



A.java

```
package model;
 class A {
  List<A> b;
 }
```

This is simply an Example.

Answer: Which **UML diagram** matches the code declarations?

Source: Created by the author

UML diagrams and database development.

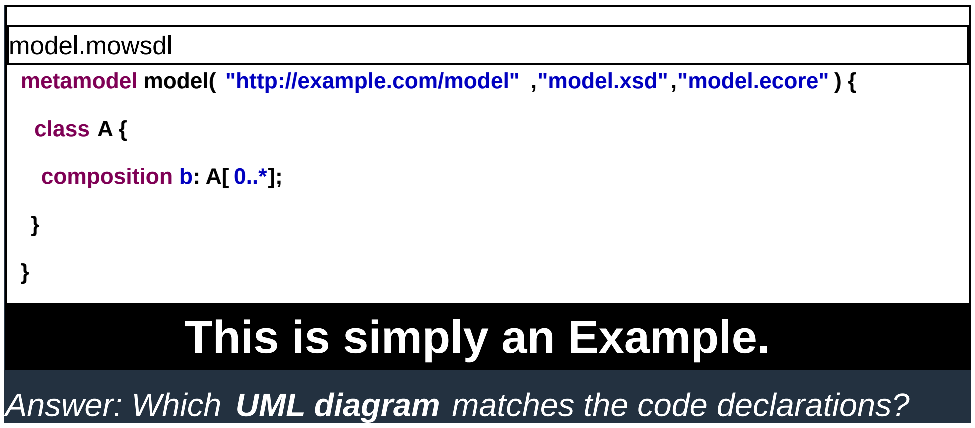
After concluding the profile characterization, the participants were led to a general instruction for the training. Afterwards, each task had their specific training session shown to the participant.

Each task was carried in sequence, however, the order of the sub-tasks were random, including the treatment type. This means that the participants knew which was the task yet were oblivious about which language (Java or MOWSDL) would be used for the next sub-task.

7.2.3.5 Data Validation

The forms filled by the participants have been programmed to restrict the input data to acceptable values, e.g. number inputs required valid number inputs and range.

Figure 56 – Simple MOWSDL Example used for Diagram Matching


model.mowsdl

```
metamodel model( "http://example.com/model" ,"model.xsd","model.ecore" ) {
  class A {
   composition b: A[ 0..*];
  }
}
```

This is simply an Example.

*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

Besides the data type validation, characterization and feedback forms allowed the researchers to cross-examine the answers in order to find contradictions, which would indicate invalid answers.

7.2.3.6 Data Collection

Since the study was performed via online forms, the data was captured on each page submit. This included their answers for characterization and tasks, as well as the timings required to submit each page. The study also included a qualitative feedback form, which is not used for the quantitative analyses.

7.2.4 Data

The purpose of this subsection is to present the raw data collected from the study executions. The data was collected from the survey system employed for this study[2]. Afterwards, this data was stored on a relational database and exported to spreadsheet and statistical analysis tools.

As the purpose of this section is to present the data without any assumption, Table 36 contains the raw data used for the analysis. This table has ten columns and specifies the answers provided by each identified participant (column P). As this task is divided into four sub-tasks, J1 lists the correctness result for the first task using Java while M1 has the result for MOWSDL instead. The correctness values are treated as Boolean variables where 0 represents incorrect and 1 represents correct. The same rule is applied to the next columns, J for Java, M for MOWSDL followed by the sub-task number. The participants completed these sub-tasks in random order, therefore it is not identified which was performed first. This table also includes the final opinion

[2] Limesurvey setup at http://limesurvey.labes.icmc.usp.br

answers by the participants for the task outcome (column O). The possible answers are the ordinals A1 to A5 explained in Section 7.3.3.

The last two rows contain aggregates for the raw data. C/T states for Correct/Total (counts). "≈ %" stands for approximate percent for the Correct/Total ratio.

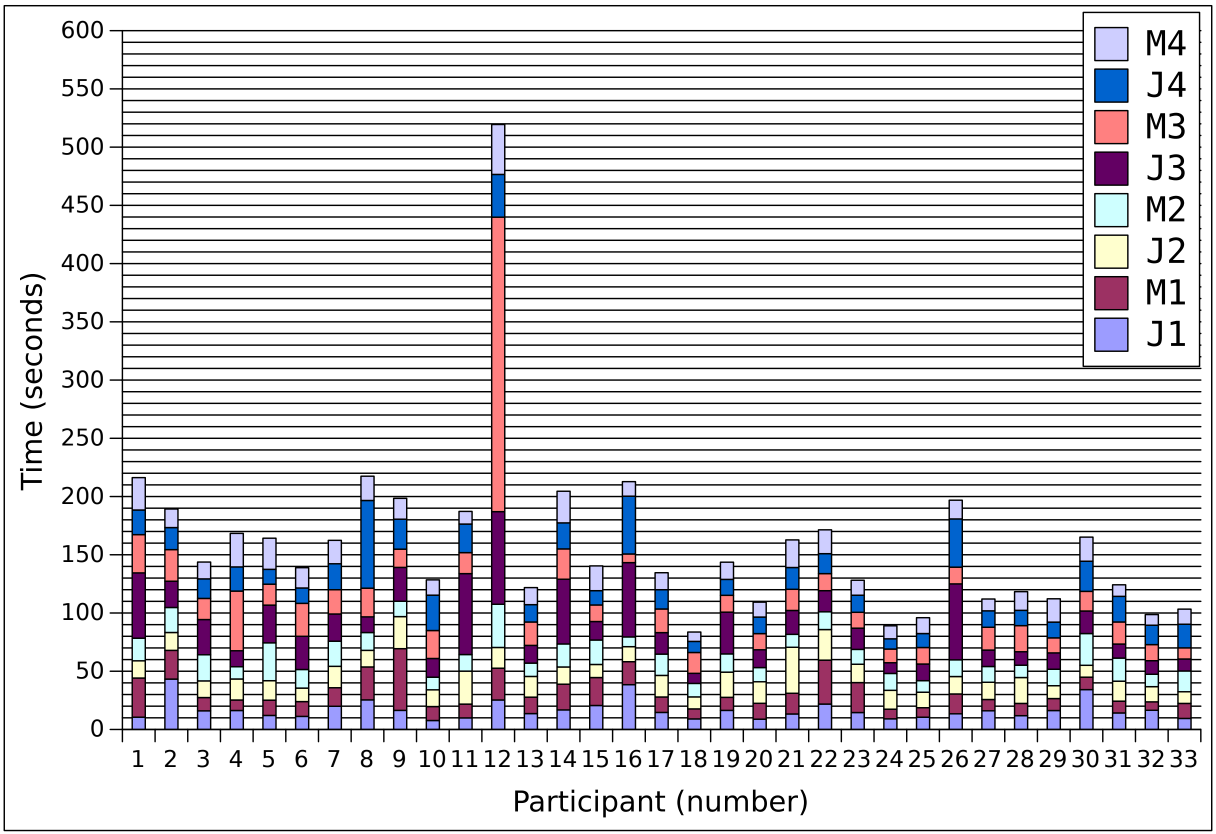
7.2.4.1 Class Identification

Table 36 – Class Counting Answers Data

	Task correctness (boolean)								
P	J1	M1	J2	M2	J3	M3	J4	M4	O
1	1	1	1	1	1	1	1	1	A3
2	1	1	1	1	1	1	1	1	A3
3	1	1	1	1	1	1	1	1	A4
4	1	1	1	1	1	1	1	1	A3
5	1	1	1	1	1	1	1	1	A3
6	1	1	1	1	1	1	1	1	A3
7	1	1	1	1	1	1	1	1	A3
8	1	1	1	1	1	1	1	1	A3
9	1	1	1	1	1	1	1	1	A3
10	1	0	1	1	1	1	1	1	A2
11	1	1	1	1	0	1	1	1	A3
12	1	1	1	0	1	0	1	1	A3
13	1	1	1	1	1	1	1	1	A3
14	1	1	1	1	1	1	1	1	A1
15	1	1	1	1	1	1	1	1	A3
16	1	1	1	1	0	1	1	1	A3
17	1	1	1	1	1	1	1	1	A3
18	1	1	1	0	1	1	1	1	A3
19	1	1	1	1	1	1	1	1	A3
20	1	1	1	1	1	1	1	1	A1
21	1	1	1	1	1	1	1	1	A3
22	1	1	1	1	1	0	1	1	A3
23	1	1	1	1	1	1	1	1	A4
24	1	1	1	1	1	1	1	1	A3
25	1	1	1	1	1	1	1	1	A3
26	1	1	1	1	1	1	0	1	A4
27	1	1	1	1	1	1	1	1	A3
28	1	1	1	1	1	1	1	1	A3
29	1	1	1	1	1	1	1	1	A3
30	1	1	1	1	1	1	1	1	A3
31	1	1	1	1	1	1	1	1	A3
32	1	1	1	1	1	1	1	1	A3
33	1	1	1	1	1	1	1	1	A3
C/T	33/33	32/33	33/33	31/33	31/33	31/33	32/33	33/33	–
≈ %	100.00%	96.97%	100%	93.94%	93.94%	93.94%	96.97%	100%	–

Table 37 contains the raw data for the timings recorded for the participants. It indicates the number of seconds required to complete each sub-task. The presentation of the columns is similar to the correctness: P for participant, J for Java, M for MOWSDL. The number suffix indicate the sub-task number. The final three rows are aggregates for the numbers, “Sum” lists the sum for the whole column, Avg. shows the average for the columns while Med. shows the median for the column.

Figure 57 – Plot for Class Counting Task Time



Source: Created by the author

The timings data is also provided as a plot in Figure 57. The bar sizes represent the allotted time in seconds. they are also stacked to represent the total time each participant took to complete the task. All colors/shades that appear on the plots are vertically ordered to ease their comprehension.

7.2.4.2 List Identification

Answers provided by the participants while taking the list identification task are presented on Table 38. This table contains ten columns and is structured similarly to the previous tables. As this task is divided into four sub-tasks, J1 presents the correctness result for the first task using Java while M1 has the result for MOWSDL instead. The correctness values are treated as Boolean variables where 0 represents incorrect and 1 represents correct. The same rule is

Table 37 – Class Counting Task Time

	Task Elapsed Time (seconds)							
P	J1	M1	J2	M2	J3	M3	J4	M4
1	10.55	33.61	14.78	19.42	56.08	32.85	21.14	27.8
2	43.25	24.66	15.35	21.48	22.55	27.13	18.93	15.95
3	15.98	11.42	14.28	22.44	30.25	18.14	16.77	14.46
4	16.29	8.99	18.05	10.5	13.74	51.24	20.76	28.78
5	12.04	13.12	16.71	32.48	32.42	17.97	12.74	26.74
6	11.22	12.71	11.59	15.97	28.51	28.28	13.19	17.44
7	19.91	15.91	18.39	21.57	23.36	20.86	22.27	20.12
8	25.38	28.29	14.25	15.31	13.46	24.61	75.31	20.85
9	16.39	52.98	27.53	13.18	29.07	15.66	25.74	17.89
10	7.69	11.98	14.36	10.85	16.07	24	30.36	13.14
11	9.92	11.85	28.32	14.24	69.47	18.03	24.54	10.86
12	25.31	27.3	17.8	37.1	79.51	252.86	36.69	42.86
13	13.64	14.1	17.79	11.44	15.21	20.08	14.9	14.65
14	16.79	22.08	14.77	19.78	55.58	26.01	22.36	27.17
15	20.59	23.99	11.23	20.98	15.89	14.2	12.28	21.28
16	38.47	19.7	12.92	8.35	63.81	7.41	49.64	12.49
17	14.63	13.25	18.53	18.29	18.45	20.29	16.55	14.59
18	9.04	8.61	10.3	11.6	8.65	17.89	9.52	8.05
19	16.38	11.2	21.63	15.65	35.87	14.45	13.67	14.78
20	8.83	13.63	18.49	12.2	15.27	13.89	14.2	12.8
21	13.29	17.79	39.49	11.1	20.51	18.17	18.72	23.69
22	21.81	37.71	26.26	15.23	18.14	14.66	17.13	20.47
23	14.53	25.71	15.77	12.74	18.23	13.62	14.67	12.85
24	9.11	8.3	16.19	14.5	9.18	11.76	8.77	11.19
25	10.53	8.13	13.31	9.93	14.29	14.18	11.99	13.61
26	13.45	17.02	14.98	14.31	65.23	14.36	41.41	16.09
27	16.03	9.7	14.79	13.55	14.1	19.58	14.1	10.16
28	11.84	10.53	22.22	10.6	11.54	22.42	13.13	16.04
29	16.26	10.28	11.02	14.15	14.04	12.87	13.55	20.03
30	34.22	10.69	10.18	27.28	19.33	16.87	25.88	20.7
31	14.17	10.13	17.15	19.83	12.06	18.94	22.01	9.92
32	16.47	7.24	13.04	10.65	11.63	13.83	16.66	9.13
33	9.47	12.9	10.01	17.83	10.29	9.55	20.32	12.95
Sum	553.480	565.510	561.480	544.530	881.790	866.660	709.900	579.530
Avg.	16.772	17.137	17.015	16.501	26.721	26.262	21.512	17.562
Med.	14.630	13.120	15.350	14.500	18.230	18.030	17.130	15.950

applied to the next columns, J for Java, M for MOWSDL followed by the sub-task number. The participants completed these sub-tasks in random order, therefore it is not identified which was performed first. This table also includes the final opinion answers by the participants for the task outcome (column O). Once again, the last two rows contain aggregates for the raw data.

Table 39 contains the raw timings for the list identification task (number of seconds

Table 38 – List Counting Answers Data

Partici-pant	Task correctness (boolean) J1	M1	J2	M2	J3	M3	J4	M4	Opin-ion
1	1	0	1	0	1	0	1	0	A2
2	1	0	1	0	1	0	1	0	A3
3	0	1	0	0	0	0	0	0	A2
4	1	0	1	0	1	0	1	0	A3
5	1	1	1	1	1	1	1	1	A3
6	1	0	1	0	1	0	1	0	A1
7	1	1	1	1	1	0	1	0	A3
8	1	1	1	1	1	1	1	1	A2
9	1	0	1	0	1	0	1	0	A3
10	1	0	1	0	1	1	1	0	A1
11	1	0	1	0	1	1	1	0	A2
12	1	1	1	1	1	1	1	1	A3
13	1	1	1	0	1	1	1	1	A1
14	1	1	1	0	1	0	1	0	A1
15	1	0	1	0	1	1	1	0	A2
16	1	1	1	1	1	0	1	0	A3
17	1	1	1	1	1	1	1	1	A2
18	1	0	0	0	1	0	1	0	A2
19	1	1	1	1	1	1	1	1	A2
20	1	0	1	0	1	0	1	0	A3
21	1	0	1	0	1	0	1	0	A1
22	1	1	1	1	1	1	1	1	A2
23	1	1	1	1	1	1	1	1	A2
24	1	0	1	0	1	0	1	0	A2
25	1	0	1	0	1	1	1	0	A2
26	1	0	1	1	1	0	1	0	A3
27	1	0	1	1	1	1	1	1	A2
28	1	1	1	1	1	1	1	1	A2
29	1	0	1	0	1	0	1	0	A3
30	1	0	1	0	1	0	1	0	A3
31	1	1	1	0	1	1	1	1	A3
32	1	0	1	0	1	1	1	0	A3
33	1	1	1	1	1	1	1	1	A3
C/T	32/33	15/33	31/33	13/33	32/33	17/33	32/33	12/33	–
≈ %	96.97%	45.45%	93.94%	39.39%	96.97%	51.52%	96.97%	36.36%	–

required to complete each sub-task).

In the same sense, the timings data is also provided as a plot in Figure 58. The bar sizes represent the allotted time in seconds and are stacked to represent the total time each participant took to complete the task. All colors/shades that appear on the plots are vertically ordered to ease their comprehension.

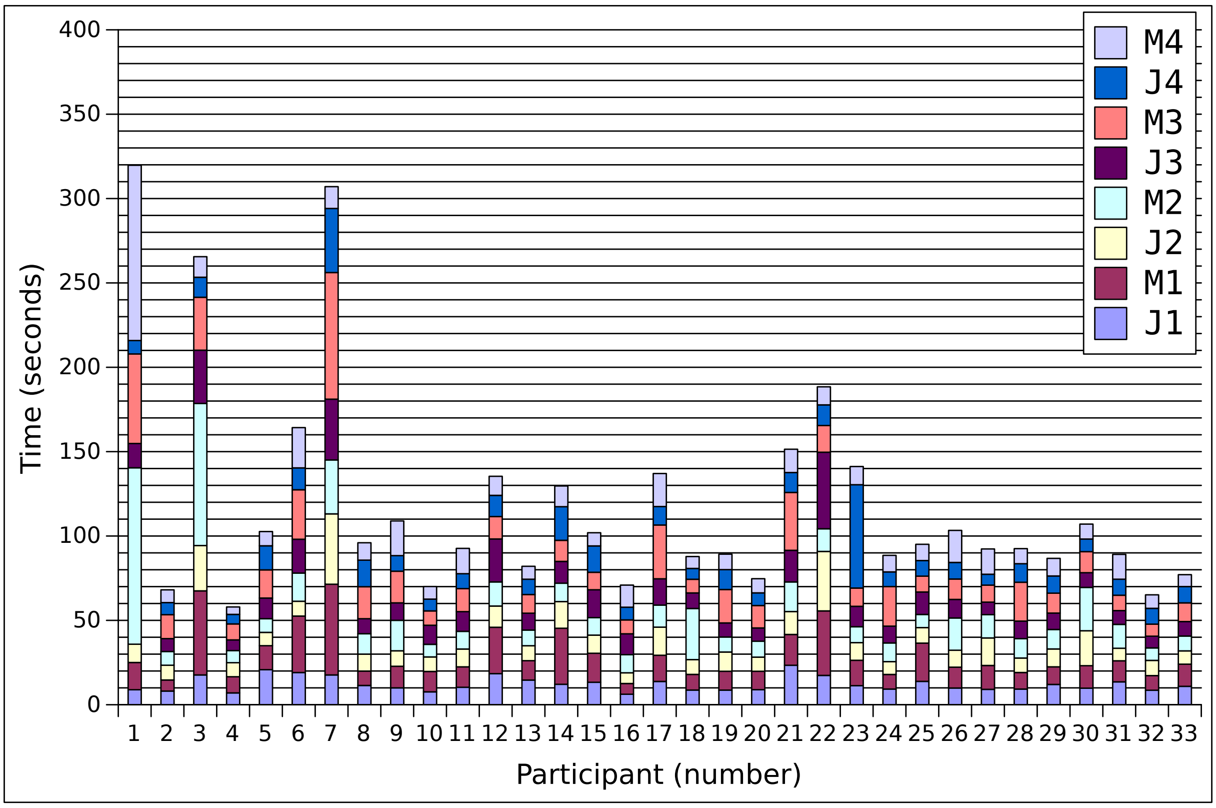
Table 39 – List Counting Task Time

	Task Elapsed Time (seconds)							
P	J1	M1	J2	M2	J3	M3	J4	M4
1	8.89	16.15	10.83	104.56	14.40	53.06	7.96	103.88
2	8.11	6.56	8.74	8.16	7.62	14.16	7.19	7.55
3	17.66	49.83	26.86	84.23	31.55	31.36	11.87	12.19
4	7.01	9.58	8.31	7.20	6.33	9.46	5.66	4.40
5	20.77	14.21	7.91	8.08	12.26	16.70	14.26	8.46
6	19.09	33.47	8.76	16.70	20.06	29.32	12.98	23.86
7	17.66	53.76	41.70	31.95	36.03	75.04	37.98	12.94
8	11.42	8.53	10.05	12.07	8.91	18.95	15.77	10.30
9	10.00	12.79	9.17	18.07	10.35	18.74	9.24	20.59
10	7.63	12.09	8.61	7.47	11.30	8.51	6.98	7.48
11	10.34	12.11	10.57	10.48	11.67	13.64	8.85	15.01
12	18.41	27.50	12.59	14.29	25.46	13.26	12.58	11.30
13	14.62	11.49	8.85	9.29	10.01	11.06	9.07	7.66
14	12.11	33.22	15.80	10.89	12.93	12.50	19.93	12.28
15	13.31	17.20	10.74	10.36	16.53	10.36	15.63	7.81
16	6.22	6.36	6.34	10.74	12.38	8.23	7.52	13.12
17	13.80	15.52	16.66	13.07	15.59	31.88	10.96	19.56
18	8.70	9.28	8.77	30.29	9.23	8.08	6.43	7.06
19	8.67	11.16	11.42	8.89	8.34	19.81	11.79	9.16
20	8.92	10.91	8.38	9.42	7.88	13.28	7.51	8.39
21	23.38	18.27	13.58	17.55	18.74	34.30	11.79	13.82
22	17.38	38.23	35.25	13.42	45.38	15.81	12.25	10.71
23	11.29	15.05	10.45	9.42	12.12	10.82	61.25	10.79
24	9.29	8.66	7.59	11.08	9.96	23.45	8.70	9.81
25	13.85	22.66	9.20	7.80	13.31	9.38	9.26	9.63
26	9.91	12.35	10.04	19.12	11.00	12.09	9.83	18.99
27	9.11	14.16	16.32	13.79	7.43	10.07	6.45	15.00
28	9.28	9.83	8.57	11.48	10.41	23.02	10.99	8.94
29	12.03	10.43	10.60	11.54	9.73	11.79	10.19	10.44
30	9.82	13.31	20.71	25.71	8.70	12.46	7.49	8.86
31	13.56	12.47	7.44	14.16	8.16	9.10	9.51	14.71
32	8.58	8.69	9.01	7.40	6.94	7.14	9.37	8.01
33	10.90	13.14	7.88	8.79	8.58	11.09	9.66	7.04
Sum	401.720	568.970	417.700	597.470	459.290	607.920	416.900	469.750
Avg.	12.173	17.242	12.658	18.105	13.918	18.422	12.633	14.235
Med.	10.900	12.790	10.040	11.480	11.000	13.260	9.660	10.440

7.2.4.3 Relationship Identification

Table 40 provides the data regarding the relationship identification task. This table is structured similarly to the previous tables. The participants completed these sub-tasks in random order, therefore it is not identified which was performed first. This table also includes the final opinion answers by the participants for the task outcome (column O).

Figure 58 – Plot for List Counting Task Time



Source: Created by the author

7.2.4.4 Relationship Identification

Table 41 contains the raw data for the timings recorded for the participants while answering the relationship identification task.

The plot for the the timings data is provided in Figure 59. The bar sizes represent the elapsed time in seconds and are stacked to represent the total time each participant took to complete the task.

The correctness results from the diagram matching task are presented on Table 42. Similarly to the other tasks, the participants completed these sub-tasks in random order, therefore it is not identified which was performed first. This table also includes the final opinion answers by the participants for the task outcome (column O).

7.2.4.5 Diagram Matching

Table 43 contains the raw data for the timings recorded during the diagram matching task (number of seconds required to complete each sub-task). Similarly to all provided plots, their colors/shades are vertically ordered to ease their comprehension.

The plot for the the diagram matching timings data is provided in Figure 60. The bar sizes

Table 40 – Relationship Counting Answers Data

	Task correctness (boolean)								
P	J1	M1	J2	M2	J3	M3	J4	M4	O
1	0	0	0	1	0	1	1	1	A5
2	1	1	0	1	1	1	1	1	A3
3	1	1	1	1	1	1	1	1	A3
4	0	1	0	1	1	1	1	1	A4
5	0	0	0	1	0	0	0	1	A2
6	0	1	0	0	0	1	1	1	A3
7	1	1	1	1	1	1	1	1	A4
8	0	0	0	0	0	0	0	0	A2
9	0	0	0	0	0	0	0	0	A4
10	0	0	0	0	0	0	0	0	A1
11	1	1	1	1	1	1	0	1	A3
12	1	1	1	1	1	1	1	1	A3
13	1	1	1	1	1	1	1	1	A4
14	0	1	0	1	0	1	0	1	A3
15	0	0	0	0	0	0	1	1	A4
16	0	0	0	0	0	0	1	0	A4
17	0	1	0	1	1	1	1	1	A2
18	1	1	1	0	1	0	1	0	A2
19	1	0	1	1	1	1	1	1	A4
20	0	0	0	0	0	0	0	0	A3
21	0	1	0	1	1	1	1	1	A3
22	0	1	0	1	0	1	0	1	A4
23	1	1	1	1	1	1	1	1	A4
24	1	1	1	1	1	1	1	1	A3
25	0	0	0	0	0	0	0	0	A2
26	1	1	1	1	1	1	1	1	A3
27	1	1	0	1	1	1	1	1	A4
28	1	1	0	1	1	1	1	1	A3
29	1	1	1	1	1	1	0	1	A3
30	1	1	1	1	1	1	1	1	A3
31	1	1	1	1	1	1	1	1	A2
32	1	1	1	1	1	1	1	1	A3
33	1	1	1	1	1	1	1	1	A3
C/T	18/33	23/33	15/33	24/33	21/33	24/33	23/33	26/33	–
≈ %	54.55%	69.70%	45.45%	72.73%	63.64%	72.73%	69.70%	78.79%	–

represent the time in seconds and are stacked to represent the total time each participant took to complete the task. The plot colors/shades are also vertically ordered to ease its comprehension.

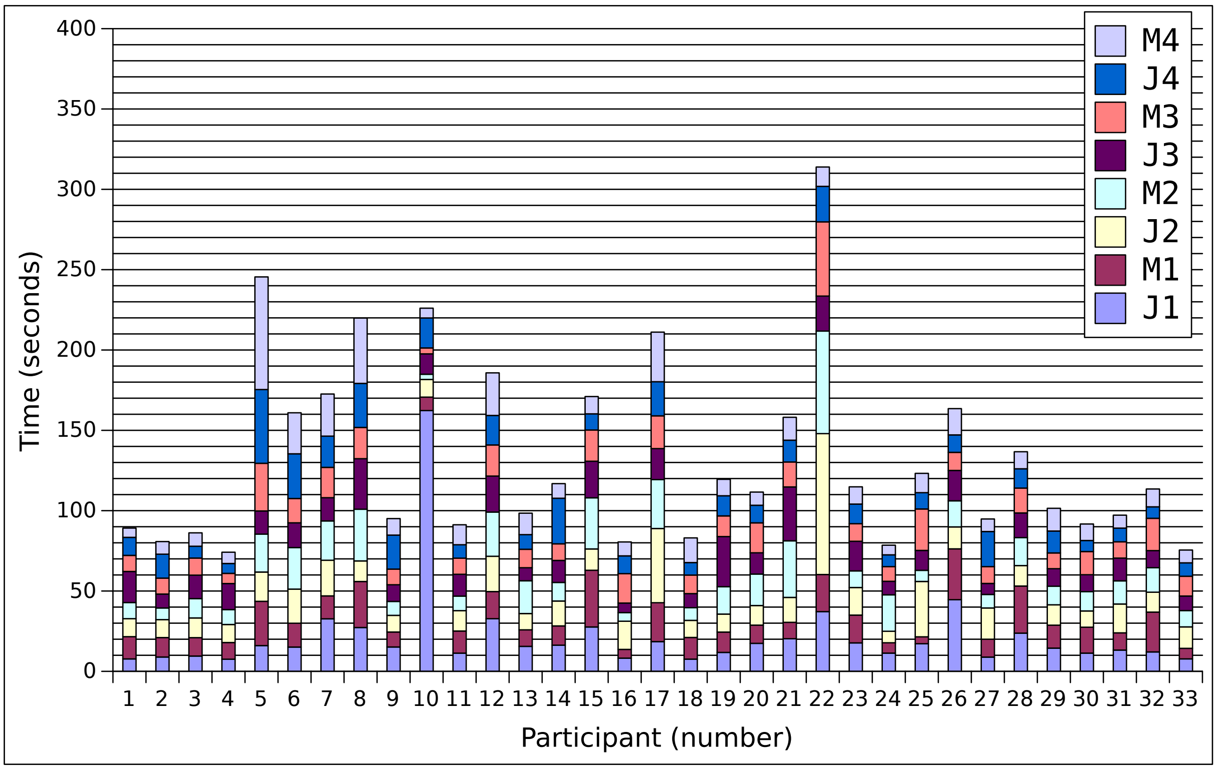
Table 41 – Relationship Counting Task Time

	Task Elapsed Time (seconds)							
P	J1	M1	J2	M2	J3	M3	J4	M4
1	7.74	13.84	11.19	10.07	19.33	10.01	11.22	5.76
2	8.92	12.11	11.15	7.27	8.65	9.92	14.90	7.85
3	9.52	11.44	12.28	12.01	14.63	10.63	7.35	8.34
4	7.55	10.39	11.22	9.27	16.22	6.33	6.15	7.02
5	16.03	27.55	18.23	23.59	14.42	29.67	45.91	70.09
6	15.12	14.83	21.23	25.84	15.43	15.12	27.82	25.53
7	32.69	14.27	22.18	24.49	14.44	18.89	19.43	26.21
8	27.28	28.64	12.81	32.16	31.46	19.46	27.41	40.67
9	15.16	9.32	10.37	8.67	10.38	9.73	21.16	10.28
10	162.30	8.37	11.00	3.24	12.69	3.67	18.66	6.10
11	11.33	13.77	12.67	9.06	13.59	10.02	8.35	12.43
12	32.75	16.90	21.97	27.50	22.42	19.36	18.37	26.50
13	15.55	10.29	10.14	20.38	8.13	11.48	9.11	13.37
14	16.29	11.98	15.51	11.57	13.69	10.33	28.37	9.08
15	27.62	35.29	13.27	31.83	22.77	19.42	10.07	10.76
16	8.22	5.44	17.54	5.37	5.93	18.29	11.08	8.66
17	18.49	24.22	46.20	30.43	19.29	20.39	21.24	30.86
18	7.60	13.50	10.63	7.89	8.75	11.56	7.77	15.36
19	11.79	12.65	11.20	17.01	31.24	12.86	12.52	10.24
20	17.42	11.34	12.20	19.59	13.24	18.64	10.94	8.21
21	20.29	10.22	15.51	35.22	33.53	15.53	13.56	14.27
22	37.16	23.11	87.68	63.83	21.78	46.18	22.10	12.05
23	17.71	17.29	17.18	10.35	18.43	10.92	12.22	10.74
24	11.35	6.43	7.17	22.67	8.43	9.11	7.35	6.01
25	17.18	4.31	34.46	6.92	12.40	25.79	10.14	12.03
26	44.58	31.66	13.66	16.24	18.91	11.25	10.85	16.36
27	8.87	11.13	19.45	8.42	6.85	10.43	21.87	7.81
28	23.78	29.31	12.72	17.41	15.36	15.47	12.02	10.61
29	14.45	14.29	12.67	11.55	10.93	9.81	13.63	14.15
30	11.34	16.13	10.12	12.01	10.54	14.40	6.90	10.26
31	13.24	10.71	17.96	14.41	14.17	10.14	8.51	8.07
32	12.09	24.78	12.39	15.23	10.66	20.10	7.14	11.11
33	7.76	6.56	13.35	10.17	9.03	12.30	8.36	7.97
Sum	709.170	512.070	587.310	581.670	507.720	497.210	492.480	494.760
Avg.	21.490	15.517	17.797	17.626	15.385	15.067	14.924	14.993
Med.	15.160	13.500	12.810	14.410	14.170	12.300	12.020	10.740

7.2.5 Results

The results of analysis and hypotheses testing considering the quantitative data gathered for this study are presented in this section.

Figure 59 – Plot for Relationship Counting Task Time



Source: Created by the author

7.2.5.1 Class Identification

The first results consider the data from the class identification task. The input data for this test are the correct and incorrect answers (presented as ones and zeros, respectively). The hypotheses were statistically tested by using two paired tests: Sign Test and Exact Wilcoxon Ranked Sign Test. The calculated values are presented on Table 44. The considered null hypothesis is the "Zero Count" while the considered alternate Hypothesis is the "Negative Count". The negative count is the alternate hypothesis since the sum of correct results for Java is greater than for MOWSDL. The test results indicate that the probability of the Null hypothesis being true is over 70%. Therefore, it is highly likely that the languages are equivalent for this task, however, it is not possible to reject any hypothesis.

Table 45 contains the hypothesis testing results for the class identification timings. Since the estimated mean is negative, the alternate hypothesis is "Negative Time", which would imply that Java takes longer for this task. As usual, the null hypothesis is the zero difference, i.e. "Zero Time". The calculations for this test applied the paired t-test. The test results suggest that the probability for the null hypothesis is also high, as the previous comparison for class identification.

The participants answered an opinion form after the task. As shown on Table 46, over 80% believe that the languages were equivalent for the task.

Table 42 – Diagram Matching Counting Answers Data

	Task correctness (boolean)								
P	J1	M1	J2	M2	J3	M3	J4	M4	O
1	1	1	1	1	1	1	1	1	A5
2	1	0	1	1	1	1	1	1	A3
3	1	1	1	1	1	1	1	1	A4
4	1	0	1	1	1	1	1	1	A2
5	1	1	1	1	1	1	1	1	A2
6	0	0	0	1	1	0	1	1	A3
7	1	1	1	1	1	1	1	1	A3
8	1	1	1	1	1	1	1	1	A3
9	1	0	0	1	1	0	1	1	A4
10	0	0	1	0	0	0	1	0	A1
11	1	1	1	1	1	1	1	1	A3
12	0	1	1	0	0	1	1	0	A5
13	1	1	1	1	1	0	1	1	A2
14	0	0	0	0	0	0	0	0	A3
15	0	1	1	0	0	1	0	0	A4
16	0	0	0	1	1	0	0	1	A3
17	0	0	0	0	0	0	0	0	A3
18	1	0	1	1	1	1	1	1	A3
19	1	1	1	1	1	1	1	1	A4
20	0	0	1	1	0	1	1	1	A4
21	1	1	1	1	1	1	1	1	A5
22	0	1	1	1	1	1	1	1	A5
23	1	1	1	1	1	1	1	1	A3
24	1	1	1	1	0	1	1	1	A3
25	1	1	1	1	1	1	1	1	A3
26	1	1	1	1	1	1	1	1	A4
27	1	1	1	1	1	1	1	1	A4
28	1	1	1	1	1	1	1	1	A3
29	1	1	0	0	1	1	1	0	A3
30	1	1	1	1	1	1	1	1	A3
31	1	1	1	1	1	1	1	1	A3
32	1	1	1	1	1	1	1	1	A3
33	1	1	1	1	1	1	1	1	A3
C/T	24/33	23/33	27/33	27/33	26/33	26/33	29/33	27/33	–
$\approx$ %	72.73%	69.70%	81.82%	81.82%	78.79%	78.79%	87.88%	81.82%	–

7.2.5.2 List Identification

Table 47 contains the statistical testing results for the list identification task. It compares the correct and incorrect answers. The alternate hypothesis is the negative count, since the correct answers for the Java artifacts were higher. According to both statistical tests presented herein, (namely, Sign Test and Exact Wilcoxon Ranked Sign Test) it is clear that the probability of the

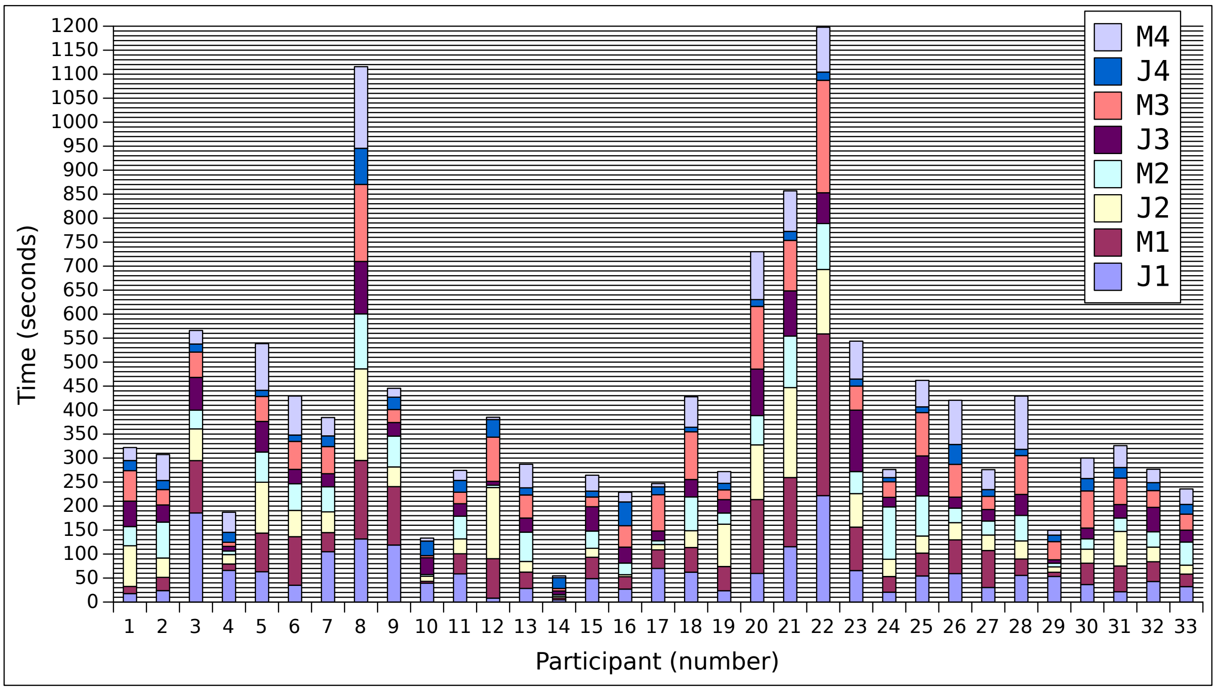
Table 43 – Diagram Matching Counting Task Time

	Task Elapsed Time (seconds)							
P	J1	M1	J2	M2	J3	M3	J4	M4
1	17.65	14.92	84.72	40.05	53.00	63.41	35.49	27.08
2	23.84	27.65	40.26	74.84	35.77	32.04	66.51	54.02
3	185.69	109.07	66.11	39.06	68.27	52.72	39.60	28.29
4	65.95	13.36	19.63	7.46	9.63	8.49	11.67	41.83
5	63.07	80.62	105.97	62.95	63.82	52.10	84.03	97.17
6	34.95	101.26	54.78	55.44	30.10	58.27	38.16	81.35
7	104.82	39.93	43.24	52.12	27.36	56.50	51.60	38.31
8	131.49	163.60	190.82	114.62	109.26	160.43	211.21	170.15
9	118.80	121.75	40.98	64.19	28.41	27.06	35.15	18.53
10	39.42	3.80	10.42	3.75	34.94	4.43	4.05	6.18
11	58.73	41.90	30.96	47.33	25.99	24.06	143.28	20.92
12	8.15	82.23	148.34	4.72	8.13	92.04	155.70	4.82
13	28.41	34.00	22.22	60.98	29.45	48.07	21.96	49.08
14	4.54	4.12	3.37	3.63	7.44	5.28	3.87	3.83
15	48.91	44.42	18.66	35.95	50.71	20.26	21.12	33.12
16	27.01	25.84	4.16	24.24	33.19	44.54	38.90	20.11
17	70.08	38.74	11.15	7.56	20.69	75.30	8.50	6.79
18	62.22	51.49	34.95	70.42	36.31	99.36	74.34	63.45
19	23.46	50.62	88.15	23.32	27.86	20.35	20.81	24.46
20	59.71	153.93	113.88	61.14	96.78	130.63	146.33	99.77
21	115.25	144.20	187.39	107.43	94.20	105.21	178.64	84.42
22	221.69	337.11	134.21	95.68	64.19	234.35	106.23	93.45
23	65.64	90.52	69.74	45.91	128.12	50.10	65.36	78.97
24	20.54	32.66	36.01	109.02	20.06	32.29	21.01	16.84
25	54.36	47.63	35.60	83.64	83.26	90.27	84.79	55.20
26	59.38	70.22	35.83	30.28	22.93	68.33	14.22	92.10
27	30.36	77.07	32.25	29.18	24.06	27.32	40.54	41.56
28	55.55	34.23	37.61	53.19	43.75	80.82	46.33	110.94
29	53.36	8.73	11.18	7.72	7.07	37.87	9.97	10.00
30	36.55	44.41	29.19	21.49	22.58	77.33	27.16	42.58
31	21.36	53.80	72.07	27.86	28.20	55.07	37.01	45.60
32	42.87	41.17	30.39	31.89	51.05	34.88	42.67	27.73
33	32.11	26.17	18.61	48.16	24.78	33.36	23.50	32.20
Sum	1985.920	2211.170	1862.850	1545.220	1411.360	2002.540	1909.710	1620.850
Avg.	60.179	67.005	56.450	46.825	42.768	60.683	57.870	49.117
Med.	53.360	44.420	36.010	45.910	30.100	52.100	38.900	41.560

null hypothesis being valid is very low, favoring Java in this comparison.

Table 48 contains the statistical testing results for the timings required to complete the list identification task. The alternate hypothesis is the positive time, since the time required for the MOWSDL version was higher. The result for the paired t-test suggest a low probability for the null hypothesis, favoring Java in this study.

Figure 60 – Plot for Diagram Matching Task Time



Source: Created by the author

The opinion of the participants after completing this task is presented on Table 49, where 19 out of 33 believe that Java was indeed better.

7.2.5.3 Relationship Identification

Table 50 lists the results for the statistical testing used to verify the hypothesis for the relationship identification task. The "Positive Count" is the alternate hypothesis, since the correct rate for the MOWSDL version was higher. Both presented statistical tests (namely, Sign Test and Exact Wilcoxon Ranked Sign Test) returned a very low probability for the null hypothesis, which suggests that MOWDL indeed outperformed Java in this comparison.

Table 51 contains the paired t-test results for the timings required to complete the relationship identification task. The alternate hypothesis is the negative time, since Java version took longer to complete in average. Despite this result favoring MOWSDL, the probability for the null hypothesis is not as low, which does not allow hypothesis rejection.

The results for the opinions answered by the participants for the relationship identification task are shown on Table 52. This table provides a scattered opinion frequency unlike previous answers. 15 out of 33 of the participants indicated that the languages are equivalent. 11 out of 33 of the participants favored MOWSDL while 7 out of 33 of the participants prefer Java for this task.

Table 44 – Hypotheses Testing for Class Task Correct Answers

Property	Value
Null Hypothesis	Zero Count;
Alternate Hypothesis	Negative Count
Variables and Difference Expression	CCM - CCJ;
Test type	Sign Test;
Medians	1;1;
N	8;
Test Statistic	3;
p-value	0.7265625;
confidence percent	95%;
Test type	Exact Wilcoxon Ranked Sign Test;
Test Statistic	13.5;
p-value	0.7265625;
confidence percent	95%;
confidence interval	-1 to 1;
difference in location	-0.5;

Table 45 – Hypotheses Testing for Class Task Timing

Property	Value
Null Hypothesis	Zero Time;
Alternate Hypothesis	Negative Time
Variables and Difference Expression	TCM - TCJ;
Test type	Paired T-Test;
Estimated Mean	-1.139545;
Degrees of Freedom	131;
t-value	-0.6329999;
p-value	0.5278371;
confidence percent	99%;
confidence interval	-5.845131 to 3.56604;

7.2.5.4 Diagram Matching

Table 53 contains the hypothesis testing considering the correct/incorrect answers of the diagram matching task. The alternate hypothesis is the “Negative Count” since the Java version had a slightly better correct count. The probability for the null hypothesis is extremely high for this study, which could indicate that the languages are equivalent.

When we consider the timings for the task, the time for identifying the diagram for the MOWSDL version took slightly longer than Java. Therefore, the alternate hypothesis considered for the statistical testing shown on Table 54 considers the “Positive Time”. Following the same conclusion as the previous test, the probability for the null hypothesis is also extremely high,

Table 46 – Class Task Opinion Frequency Distribution

Answer Code	Answer Text	Frequency	Percent
A1	Java is Much Better	2	06.06%
A2	Java is a Bit Better	1	03.03%
A3	Seem Equivalent	27	81.82%
A4	MOWSDL is a Bit Better	3	09.09%
A5	MOWSDL is Much Better	0	00.00%
	Total	33	100.00%

Table 47 – Hypotheses Testing for List Task Correct Answers

Property	Value
Null Hypothesis	Zero Count;
Alternate Hypothesis	Negative Count
Variables and Difference Expression	CLM - CLJ;
Test type	Sign Test;
Medians	0;1;
N	72;
Test Statistic	1;
p-value	$3.091670257478174{\cdot}10^{-20}$;
confidence percent	95%;
Test type	Wilcoxon Ranked Sign Test with Continuity Correction;
Test Statistic	36.5;
p-value	$1.633055{\cdot}10^{-16}$;
confidence percent	not applicable;
confidence interval	not attainable;
difference in location	not attainable;

which favors the equivalent hypothesis (“Zero Time”).

The opinion answers for the diagram matching task is provided on Table 55. Despite the Java version attaining a better result for the correct answers, only 4 out of 33 of the participants voted Java as a better option. Most participants think they are equivalent, however, 11 votes for MOWSDL as better option repeats the result for the relationship counting.

7.2.6 Discussion

The purpose of this subsection is to discuss the experiment presented in this section. This experiment was planned to verify if developers could use the MOWSDL language, which was created to assist programmers while defining metamodels, with the intent of empowering them to create software to handle models, e.g. MOSS/MOWS.

MOWSDL was never published or made public prior to the experiment, which implies

Table 48 – Hypotheses Testing for List Task Timing

Property	Value
Null Hypothesis	Zero Time;
Alternate Hypothesis	Positive Time
Variables and Difference Expression	TLM - TLJ;
Test type	Paired T-Test;
Estimated Mean	4.155303;
Degrees of Freedom	131;
t-value	2.973047;
p-value	0.003510379;
confidence percent	99%;
confidence interval	0.5019925 to 7.808614;

Table 49 – List Task Opinion Frequency Distribution

Answer Code	Answer Text	Frequency	Percent
A1	Java is Much Better	5	15.16%
A2	Java is a Bit Better	14	42.42%
A3	Seem Equivalent	14	42.42%
A4	MOWSDL is a Bit Better	0	00.00%
A5	MOWSDL is Much Better	0	00.00%
	Total	33	100.00%

that this was the first contact the participants had with it. Therefore, these results also suggest that developers could use the language without much training. Indeed, the selected participants had years of programming experience, they knew the Java Programming language prior to the study and were able to understand MOWSDL after their first contact with the language.

MOWSDL usage was positive for relationship identification thanks to its visible declaration of relationships and compositions. Following the same logic, the explicit usage of “List” in Java also made it clear when Java code had lists. This was not clear for other tasks, which had very high probability for the null hypotheses.

It is completely possible to write MOSS/MOWS systems without MOWSDL. The MOWSDL language was not created to replace Java or any other language, it was simply defined as an add-on to the tool-chain presented in Section 5.9, assisting developers to create code that would otherwise require extensive checking to certify that it is compatible to models as used by modeling tools.

It is important to mention that proper Java code to handle models as data also requires a few model annotations which could pollute the code and cause mistakes. This was not required for this study, which causes further advantages towards the programming language.

Besides these results, the study indicates that developing code for handling models as data could be a feasible task for common programmers. Another study is provided in this chapter

Table 50 – Hypotheses Testing for Relationship Task Correct Answers

Property	Value
Null Hypothesis	Zero Count;
Alternate Hypothesis	Positive Count
Variables and Difference Expression	CRM - CRJ;
Test type	Sign Test;
Medians	1;1;
N	30;
Test Statistic	5;
p-value	0.0003249142;
confidence percent	95%;
Test type	Exact Wilcoxon Ranked Sign Test;
Test Statistic	387.5;
p-value	0.0003249142;
confidence percent	95%;
confidence interval	0 to 1;
difference in location	0.5;

Table 51 – Hypotheses Testing for Relationship Task Timing

Property	Value
Null Hypothesis	Zero Time;
Alternate Hypothesis	Negative Time
Variables and Difference Expression	TRM - TRJ;
Test type	Paired T-Test;
Estimated Mean	-1.598258;
Degrees of Freedom	131;
t-value	-1.143369;
p-value	0.2549705;
confidence percent	99%;
confidence interval	-5.252068 to 2.055553;

to evaluate how programmers use both languages to develop components for MOWS systems (Section 7.3).

Regarding the feedback provided as qualitative answers by the participants, there were several participants who praised how the forms were designed and their looks, which could have encouraged them to complete the form and share it to more participants. A few participants reported that the quick tasks were engaging and even fun, making them curious for results.

Several participants also wrote criticisms. They have reported that it was very cumbersome to answer the diagram matching task, which required them to go back and forth several times to compare the text to diagram. Other participants were contrary to the ones who have

Table 52 – Relationship Task Opinion Frequency Distribution

Answer Code	Answer Text	Frequency	Percent
A1	Java is Much Better	1	03.03%
A2	Java is a Bit Better	6	18.18%
A3	Seem Equivalent	15	45.46%
A4	MOWSDL is a Bit Better	10	30.30%
A5	MOWSDL is Much Better	1	03.03%
	Total	33	100.00%

Table 53 – Hypotheses Testing for Diagram Matching Task Correct Answers

Property	Value
Null Hypothesis	Zero Count;
Alternate Hypothesis	Negative Count
Variables and Difference Expression	CDM - CDJ;
Test type	Sign Test;
Medians	1;1;
N	25;
Test Statistic	11;
p-value	0.690038;
confidence percent	95%;
Test type	Exact Wilcoxon Ranked Sign Test;
Test Statistic	143;
p-value	0.690038;
confidence percent	95%;
confidence interval	-1 to 1;
difference in location	-0.5;

found it engaging and criticized its length and found it boring and tiring.

It is important to report criticisms by some participants on the apparent objective of the survey. These participants were led to believe that MOWSDL was just a language to help developers to write data structures and interfaces while adding some semantics that are lost when using Java. This is the major point of MOP (FORWARD; BADREDDIN; LETHBRIDGE, 2010). The argument by these participants that it could be nonsensical to approximate design to code further confirms assumptions discussed after the systematic mapping presented in this thesis (Chapter 3).

Programmers might dislike new languages which are often created to solve the same problems. The proposition of MOP suggests that programmers should use another language while they might refuse this request. This could be also applied to MDSE tools, which might be seen as just another tool for the same needs.

Table 54 – Hypotheses Testing for Diagram Matching Task Timing

Property	Value
Null Hypothesis	Zero Time;
Alternate Hypothesis	Positive Time
Variables and Difference Expression	TDM - TDJ;
Test type	Paired T-Test;
Estimated Mean	1.590455;
Degrees of Freedom	131;
t-value	0.4151639;
p-value	0.6787011;
confidence percent	99%;
confidence interval	-8.42308 to 11.60399;

Table 55 – Diagram Matching Task Opinion Frequency Distribution

Answer Code	Answer Text	Frequency	Percent
A1	Java is Much Better	1	03.03%
A2	Java is a Bit Better	3	09.09%
A3	Seem Equivalent	18	54.55%
A4	MOWSDL is a Bit Better	7	21.21%
A5	MOWSDL is Much Better	4	12.12%
	Total	33	100.00%

Indeed, in the case of Java, annotations could be used to keep the required semantics to generate design models correctly. The annotations are also employed for MOWS coding when using Java for interface and data structures.

It has been cited that the experience could change the results. This implies that highly experienced Java programmers could feel more confident while using Java, thus affecting their answers.

Further qualitative analysis are planned as future works. Data was stored by the survey system for participant opinion. By analyzing their background and opinion, it could be possible to identify new variables and explain how specific preferences could impact the usage of the studied languages. This data could also be validated by cross-examining the feedback according to their experience and background.

7.2.7 Threats to Validity

This subsection presents the threats to validity for this study. These threats are distributed into Internal Validity, Validity by Construction, External Validity and Conclusions Validity.

7.2.7.1 Internal validity

Experience Level of Participants. The different levels of knowledge of the participants could have compromised the data. To mitigate this threat, the experiment tasks included a training session. Also, it was expected that the participants had prior programming experience, therefore, they were selected according to their profile. As all participants were experienced Java programmers, there is also a MOWSDL learning factor that could have biased this study. According to the results, MOWSDL had balanced successes, which indicates that despite this bias, the created language was successfully understood by the overall participants. This also suggests that MOWSDL could have in fact been more successful if the participants had more experience. We claim that these points are not problematic because they balance the researcher's bias, which is the bias that could suggest that we intended to make the study partial towards MOWSDL, which did not happen. It also balances the threat of hypothesis expectations, which is further described in this subsection.

Facilities used during the study. Different computers, devices, connections and configurations could have affected the recorded logs. This threat is mitigated since each participant had to complete all the tasks without changing their device and connection, allowing the researchers to capture the control data in proportion to the treatment data.

7.2.7.2 Validity by construction

Hypothesis expectations. the participants' expectations could have affected the results. To mitigate this threat, we have collected as much data as possible from the form execution. We also asked the participants to complete their answers as fast as possible regardless of the task. The sub-tasks were also performed in random order to avoid the participant from changing their execution pace. Also, we have concealed the objective of the experiment and their impact on it to avoid them to actively affect their data towards a specific result. In the forms, it is not specified that MOWSDL was built specifically for MOWS and MOSS development activities, these forms were written to make the participants to believe that it was simply a language comparison study.

7.2.7.3 External validity

Interaction between configuration and treatment:. It is possible that the exercises were not accurate for real world applications. The experimental application had simple requirements and random (meta-) model object names. To mitigate this threat, the range of the number of objects and relations were based on the case studies. Still, the randomness was used to avoid the memory effect of participants, which would cause another threat to validity.

7.2.7.4 Conclusion validity

Measure reliability. It refers to how precise was the data collection and measurements. To mitigate this threat, all data was captured automatically as soon as the participants concluded each activity in order to allow better precision;

Low statistic power. Since we have a small population, we applied statistical tests to analyze the experiment data while avoiding issues with low statistic power. Besides that, we are working on larger scale experiments and applications for the proposed methods, i.e. the study will remain open for more participants.

7.3 Data Structure Implementation Experiment

7.3.1 Objectives

In this section, an experiment conducted to compare methods in the domain of web services development is presented. This study allows to compare the efforts of using code-first (Java code) and model-first (MOWSDL) development methods. The rationale behind this study is beyond this simple comparison, as it allows the researchers to identify if other developers would be able to develop MOWS systems based on metamodels while also evaluating the newly created tools.

In order to develop these systems, a service interface and the shared data structures must be defined. This definition is common to both server and client parts within the development effort.

Prior to planning this study, the author interviewed professors involved into web services development. They have described the effort of developing data structures of distributed servers and clients as significant during initial implementation and repetitive throughout the maintenance. Indeed, in the literature, authors compare different web services technologies, discussing that they affect how the interfaces are defined, while data structuring is required for all technologies (PAUTASSO; ZIMMERMANN; LEYMANN, 2008; PAUTASSO; WILDE, 2010).

In the scope of MOWS definition, the author has proposed a technique for creating data structures based on metamodeling and model based transmission. Therefore, this study was planned to compare the efforts to implement and to maintain the data structures required for the correct operation of web services systems.

It is also employed to analyze the efforts involved in the data structure definition, which are treated differently whenever the development method is code-first (Java) or model-first (MOWSDL).

It is expected that the outcome of this study helps developers and researchers to identify the proper method when developing software related to this domain.

7.3.2 Method

In this experiment, the development of portions of service-oriented systems are considered using different development methods. The experiment was applied in the context ofgraduate students properly trained in software development activities to perform the role of software developers. The study was carried out from the perspective of the researchers, however, it was also intended to cover the expectations from software project managers.

The object of this study is the effort and comprehension of the maintenance of software artifacts that are employed as part of a service-oriented software system. For the web service implementation, Apache CXF (Apache Foundation, 2017) was used, which is compliant to JAX-WS/JAX-RS Java specifications. Therefore, this object is not restricted to the specific case of CXF, but applicable to other JAX-WS/JAX-RS compliant frameworks.

The object was treated with two different development methods. The first method is the original development technique as defined by the developers of Apache CXF for web services software system instantiating in a code-first approach.

Apache CXF (Apache Foundation, 2017) includes a tool-chain for web services development in both code-first and model-first (also referred as contract-first in its documentation) sequences, however, they were not planned for MOWS development. In this project, a new tool-chain for model-first sequence has been developed specifically for MOWS, as described in Chapter 5 (Section 5.9).

Table 56 – Study Design

Phase	Group 1	Group 2
General Training	Development Training	
	Cashier Application	
1st Execution Development Phase	Code-First	Model-First
	Deliveries Application	
2nd Execution Development Phase	Model-First	Code-First
	Flights Application	
1st Execution Development Phase	Code-First	Model-First
	Medical Clinic Application	
2nd Execution Development Phase	Model-First	Code-First
	Restaurant Application	

The complete study design is presented on Table 56. This table is composed by three columns: the phase indicates the application development task, the other two columns indicate the treatment used for each group.

The dependent variables that were captured and analyzed in this study are shown on Table 57. This table includes the name of the variable, description, along with range and collection

Table 57 – Study Dependent Variables

Name	Description	Type	Range	Precision
TCJ	Time Completion Java	Real (Ratio)	0 or greater	1/1000 second
TCM	Time Completion MOWSDL	Real (Ratio)	0 or greater	1/1000 second

precision. The first variable captures the time for the Java treatment, while the second variable captures the time for MOWSDL treatment. Both variables are the time taken to complete the given task correctly, i.e. the participants had to reach the required implementation, which implies that all tasks led to the same final quality.

Table 58 – Study Hypotheses

Name	Description	Predicate
Negative Time	Java takes longer for data structure coding	$TDIM - TDIJ < 0$
Zero Time	Languages are equivalent for data structure coding	$TDIM - TDIJ \approx 0$
Positive Time	MOWSDL takes longer for data structure coding	$TDIM - TDIJ > 0$

Table 58 contains the hypotheses presented as possible outcomes for the study. The names of the hypotheses are defined as the resulting difference for each task. Negative Time ($TDIM - TDIJ < 0$) suggests that MOWSDL takes less while Java takes longer for data structure coding (therefore the difference MOWSDL minus Java is negative). Positive Time ($TDIM - TDIJ > 0$) suggests that MOWSDL takes longer while Java takes less for data structure coding (therefore the difference MOWSDL minus Java is positive). If the resulting difference is close to zero, then they would be approximately equivalent (Zero Time). These hypotheses are taken as alternate or null, depending on the hypotheses testing, as presented in Subsection 7.3.5.

7.3.3 Operation

7.3.3.1 Participant Selection

Graduate Students with prior programming experience were selected for the study. Invitations were sent to dozens of students. Eleven qualified students accepted the invitation, however only five completed the study, because the others used their right to quit at any time. The participants numbers are the same from Table 34 (Section 7.2), however, only participants numbers from 1 to 5 have completed this specific study.

7.3.3.2 Preparation

In preparation for the experiment, the participants were trained on web services development methods. Besides the instructions on how to implement MOWS servers and clients using both model-first and code-first sequences with MOWSDL and Java, this activity included the introduction, data structures, services interfaces and components.

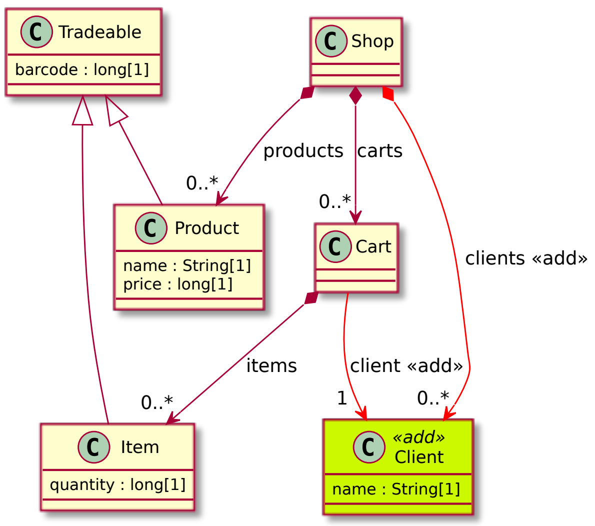
7.3.3.3 Instrumentation

The participants received a printed instruction booklet on the method required for the implementation. The participants were also provided with components along with a test case to be executed after performing their intended activities. The instrumentation documents were all written in English. Verbatim copy of the documents are provided within Appendix E.

One of the components would be modified by the participants, while the test case was employed to verify whether they succeeded or not in their activity. The provided components were created for specific applications, which were all similar in complexity.

Every participant had to modify a component from each application by using only one of the techniques in equal numbers. The participants were grouped in two groups to permute the order the methods were executed. At any moment of the experiment, each group was using a different method than the other group.

Figure 61 – Cashier Application Diagram



Source: Created by the author

The applications were provided according to the study design (refer to Table 56). All employed applications had the same number of classes and required changes. For instance, the training application is the Cashier application, which represents a shop management system. Its class diagram is represented in Figure 61. The study applications are provided as examples of feasible MOWS systems within Appendix C.

7.3.3.4 Execution

Initially, the participants signed a consent form and then answered a profile characterization form. The characterization form had questions regarding knowledge about web services development and XSD edition.

After concluding the profile characterization forms, participants were trained on how to implement web services servers and clients by using the code-first (Java) sequence and then the model-first (MOWSDL) sequence with the tool proposed for model-oriented web services. It is important to note that every participant already had a basic experience with web services coding and component modeling.

Following the training, the experiment was executed. The participants were split into two balanced groups considering the results of the characterization forms. During the experiments, the participants had to work with two applications composed by the provided components. They would be asked to start with a different technique for each group. The secondary executions were replications of the primary executions with another two applications. They were created in order to avoid the risk of getting unbalanced results during the primary execution.

7.3.3.5 Data Validation

The characterization forms filled by the participants were confirmed with data collected during the study execution. The participants were observed throughout the executions and the test cases allowed to verify the conclusion of each activity.

7.3.3.6 Data Collection

The participants had to implement the required definitions and run test cases to identify whether they have completed the task successfully. Data for this study was collected by instrumenting these test cases, which allowed the researchers to capture data without transmission delays and in milliseconds precision. Therefore, the data was recorded before notifying the participants the result, thus avoiding affecting the data.

7.3.4 Data

Table 59 – Data Structure Implementation Task Time

	Task Elapsed Time				
P	J1	M1	J2	M2	G
1	239.194s (04m59s)	218.779s (03m39s)	68.123s (01m08s)	33.472s (00m33s)	1
2	1697.891s (28m18s)	120.1s (02m00s)	1677.143s (27m57s)	168.105s (02m48s)	1
3	530.537s (08m51s)	919.077s (15m19s)	1692.456s (28m12s)	144.94s (02m25s)	1
4	301.608s (05m02s)	86.097s (01m26s)	133.583s (02m14s)	93.144s (01m33s)	2
5	580.555s (09m41s)	253.383s (04m13s)	729.155s (12m09s)	195.718s (03m16s)	2

The purpose of this subsection is to present the raw data collected from the study executions. The elapsed time data for completing each implementation task is presented on Table 59. This table contains six columns: the first column identifies the participant number (P); the second column includes data from the first task by using Java coding (J1), while the third presents the data from the same task by using MOWSDL instead (M1).

The following columns (fourth and fifth) present the data for the second task by using both methods (J2 & M2). Finally, the last column indicates the group of the participant (G). Therefore, the table allows the viewer to compare the data of Java and MOWS tasks in pairs. It is also important to remind that all tasks were designed to have the same complexity, however, the experience gain from participants could help them to complete the second task faster. The rows of the table represent each participant.

Basic data analysis suggest that only the third participant took longer to implement a task (the first task) by using MOWSDL.

7.3.5 Results

Table 60 – Hypotheses Testing

Property	Value
Null Hypothesis	Zero Time;
Alternate Hypothesis	Negative Time
Variable and Expression	TDIM - TDIJ;
Test type:	Paired T-Test;
Estimated Mean	-541.743;
Degrees of Freedom	9;
t-value	-2.341229;
p-value	0.04393179;
confidence percent	99%;
confidence interval	-1293.731 to 210.2449;

Table 60 contains the statistical testing results for the collected data. The considered null hypothesis is the “Zero Time”, while the considered alternate Hypothesis is the “Negative Time”. The test was performed by taking pairs of the results and computing their differences. The test results indicate that the probability of the Null hypothesis being true is below 5%, thus favoring the Alternate Hypothesis. However, the confidence interval includes the estimated mean, which suggests that attaining higher confidence level of 99% could require more data.

As the estimated mean is negative, it is suggested that it is highly likely that the alternate hypothesis is true, stating that it is faster to use MOWSDL for data structure maintenance and development.

7.3.6 Discussion

The purpose of this study was to compare the data structure coding effort and evaluating if participants could develop MOWS systems, which require specific rules for their data structure code, since they handle models as data. In order to handle models as data, programmers are required to program according to metamodel definitions.

The first assumption would suggest that MOWSDL could be easier and take less effort, since it was created for this purpose. However, this study could indicate further conclusions. After the first definitions of the MOSS/MOWS proposal, the major concern was whether programmers would be able to work on metamodeling level while coding. As the major concern related to the MOSS and MOWS development relates to the doubt of whether programmers could be able to use metamodels as design, the study has suggested that programmers are not only able to work on MOWS systems but to also develop code to handle metamodels either by using Java or MOWSDL.

Although confirming the possibility of more programmers developing MOWS systems, the validity of this study has several threats to discuss, as presented within Subsection 7.3.7.

7.3.7 Threats to Validity

Similarly to the study presented in Section 7.2, this section presents the threats to validity for this study. They are distributed into the following categories: Internal Validity, Validity by Construction, External Validity and Conclusions Validity.

7.3.7.1 Internal validity

Experience Level of Participants:. The different levels of knowledge of the participants could have compromised the data. To mitigate this threat, the participants were throughly trained before the study tasks.

Facilities used during the study: Different computers and configurations could have affected the recorded logs. However, participants worked by using the computers with the same make and model in the same room and at the same time.

7.3.7.2 Validity by construction

Hypothesis expectations: the participants' expectations could have affected the results. To mitigate this threat, we have collected as much data as possible and encouraged the participants to perform as natural as possible. Also, we have concealed the objective of the experiment and their impact on it to avoid them to actively affect their data towards a specific result.

7.3.7.3 External validity

Interaction between configuration and treatment:. It is possible that the exercises were not accurate for real world applications. The experimental application had simple requirements. To mitigate this threat, we designed the exercises based on functional case studies based on real world applications.

7.3.7.4 Conclusion validity

Measure reliability. It refers to how precise was the data collection and measurements. To mitigate this threat, all data was captured automatically as soon as the participants concluded each activity in order to allow better precision;

Low statistic power. Since we have a small population, we applied T-Tests to analyze the experiment data statistically to avoid the issues with low statistic power. Besides that, we are working on larger scale experiments and applications for the proposed methods.

7.4 Final Remarks

In this chapter, two experimental studies were presented. Their results have indicated that other developers were able to use the specified language, as well as use an existing programming language for MOWS development. While some could argue that the experiments were too short to represent actual systems, in Appendix C there are more examples of possible software systems that can be implemented according to MOSS principles. All these studies conclude that MOWS and MOSS are feasible as a proposal for evolving MDSE despite not being a replacement. Further conclusions are provided in Chapter 8. Further packing including verbatim copies of the instrumentation are provided as appendixes: Appendix D contains the documents of the data verification experiment while Appendix E contains the documents of the implementation study. Related documents and papers besides this thesis are also available[3].

[3] <http://tiny.cc/gottardi-doc>

CHAPTER

8

CONCLUSIONS

8.1 Initial Remarks

This chapter concludes the main portion of this thesis. Besides the introduction, the theoretical foundation included a balanced set of topics that were effectively used as part of the presented research efforts. The general conclusions of the thesis are presented in Section 8.2. Conclusions drawn specifically from the MDSE challenges and possible evolution are discussed in Section 8.3. Conclusions from MOSS/MOWS systems are presented in Section 8.4. Future works are presented in Section 8.5. They also include plans to deal with limitations of this thesis. Finally, the publications derived from the project presented in this thesis are cited in Section 8.6.

8.2 General Conclusions

This thesis involved different research efforts focused on studying and extending the MDSE as a method. The research efforts were distributed into different types, the idea was to include exploratory, descriptive and explanatory types of research regarding MDSE. Therefore, it was intended to conduct literature surveys and reviews to explain and describe the research context, as well as presenting an overview of the state of art on MDSE research.

This thesis also included empirical studies (Chapter 4) that served as basis for proposing a plausible software category that was evolved from concepts present in MDSE, MOP and MRT. This software can be built starting from metamodel specification. This was also referred as a top down method, according to a method engineering exercise.

Therefore, among contributions presented within this thesis it is possible to highlight the usage of process analysis, which led to a method engineering exercise with the intent to create a new software category evolved from MDSE. The proposed evolution is composed by a software category referred as Model-Oriented Software System, along with its web services application, Model-Oriented Web Services. We claim that they foster a step forward on how MDSE tools could be employed outside MDSE methods, allowing to use Domain Specific Languages (DSL) for data handling and communication. Thanks to the usage of a concrete syntax, these languages can assume alternate representations, allowing humans to use a common language with the software besides code-level, including data visualization.

The analysis techniques that were used for method engineering were based on process discovery techniques (Chapter 4). This thesis includes an algorithm formalization and a metric with analytic and experimental validation. While this algorithm was beneficial for the development of the proposal of MOSS, it was later identified that it was not mandatory.

The proposed software category (MOSS) was applied to web services construction (MOWS), consolidating this category as concrete and feasible for the development of specific kinds of applications (Chapter 5). Sample applications were provided as case studies that include their design and purpose. A set of tools and a design language were created for assisting its development, though they are not mandatory. Studies on the development of MOSS were conducted. They are categorized into comparative (Chapter 6) and experimental (Chapter 7).

The comparative studies involved formalism to provide a sound mathematical evaluation of the context. The experimental studies involved professional developers, graduate and undergraduate students. These studies allowed to verify if other developers would be able to implement these systems.

8.3 MDSE Challenges and Evolution Study Conclusions

The MDSE challenges were studied by conducting a secondary study, referred as a systematic mapping (Chapter 3). This conduction involved the review of 4859 studies, in which 3727 were unique. This review allowed the author to identify the most common application and technological domains where MDSE has been employed. Besides these domains, this secondary study fostered discussions related to challenges developers face while attempting to apply MDSE to projects dealing with uncommon or too specific domains.

As part of results summarizing, we have identified that, as reported by academic publications, the MDSE success domains are clustered into application and technological domains. This data was presented quantitatively considering the success cases that were not only used as case studies. The success cases indicate that MDSE has reached production levels for specific domains. In this manner, it is suggested that MDSE is recommended for specific domains, involving both academia and software industry.

During our searches, we could not find a report on a failure case, still, we identified challenges and presented these qualitatively in a discussion section (Section 3.4).

There are 17 identified studies which are related to MDSE. This discussion involved challenges related to software maintenance and methodology issues. The studies have also been categorized and summarized.

This secondary study presented the review of studies from 1985 to 2018. Besides that, we could find no evidences that the proposed solutions were in fact used. Considering the maintenance problems, we argue that these issues could rise in any project and they should be mitigated since its beginning.

8.4 MOSS and MOWS Conclusions

As a major contribution of this thesis, it has been delineated a view on how to push MDSE and WS development beyond models to generate software, hoping to define paradigm that is perceived by developers and end-users thanks to the creation of model-oriented or model aware software that goes beyond the definition of model-oriented programming.

While MOSS is hypothetical, MOWS is a concrete example, including a development method. Both MOSS and MOWS require a specific metamodel structure that is robust enough to allow partial transmission, while still maintaining security for protected information. Since the metamodels that follow this pattern would share many similarities, it would be possible to reuse code generators.

It is important to remind that the usage of DSLs in the context of MOSS would allow software and humans to communicate using a single language that represents data. In this view, this paradigm is achieved by employing models throughout analysis, design, implementation towards the final application execution. This means that we are not limiting the models within MDSE as "Models are Code", in this case, models are software, data structures, the data itself along with the state representation of the running software. In our method, metamodels and models can be used in code generation. Models can be also interpreted at run-time to store, transfer and represent data. Therefore, both code generation and code interpretation are employed for the same final application.

Because of the application of these MDSE properties since earlier development phases, along with MOWS definitions and its development method, models are not only used for design, implementation or deployment. Models are used throughout the complete development life-cycle and during the execution, being visible to the end-users. This justifies our argument that MOSS represents the specification of this new category based on MDSE, as well as defining it as a new paradigm. In this manner, models are not only the code, they are also the data and can represent the configuration, settings and state of the running application.

It is also possible to increase flexibility for MOWS systems that require adaptability for different settings, while maintaining a state of the system, all defined within models (GOTTARDI; BRAGA, 2016). It is expected that the flexibility of MOWS systems would be evaluated in future studies.

8.5 Future Works

As future work, it is planned to work on the limitations of this thesis and carry out more in depth validations. It is also intended to apply the propositions presented herein into other contexts and report the results. Since this thesis involves several topics, the future works are distributed into subsections.

8.5.1 Process Discovery Algorithm

This thesis is involved in a project that involves studying process tool support that includes process discovery and recommendation. The process discovery support must be flexible enough to allow developers to extend the discovered models to better suit their needs. This flexibility is important because we intend to apply updated model-driven development methods by using model transformations on the discovered process models and devise new studies on their flexibility, adaptability and reuse.

8.5.2 Process Comparison Metric

As future works, we intend to cope with the limitation regarding the proof on the scope of processes. The current proof is valid when focusing on the input matrices. Still, similar matrices would indicate processes that have different number of interactions, which could be categorized into the same original expected process.

The presented threats to validity also encourage future works, since the experiment was too small to assure whether the proposed system is accurate for larger software projects.

This work is part of a research group focused on studying tool support for software reuse and we intend to create tool support for this metric system, as well as experimenting with several software projects.

8.5.3 MOSS and MOWS

Future works are required in order to deal with limitations of MOSS and MOWS. These limitations include lack of more adequate and more formal case studies. The presented comparative, experimental and case studies might not be sufficient to cover every advantage and disadvantage of the software category. Therefore, more validation could be used to confirm when the suggested advantages overcome disadvantages.

In case of a more trustworthy confirmation of the advantages and disadvantages, we also identify the need of a set of guidelines to guide project managers when they have to decide to use the MOSS/MOWS requirements for their software application developments.

It is also planned to provide continuous support to the development teams that accepted to apply the defined methodology in order to help them to reach a stable version of their software systems, which is going to be available to the public to use. It is worth mentioning that this thesis is related to a research project to study system of systems development and reuse, which inspired the development of MOWS systems for this domain. Further details on the development projects are provided as part of the Appendix C. There could be more advantages and disadvantages related to the development efforts that could be studied, and they can go beyond data structure specification.

Documenting reusable valid metamodel designs to assist developers when defining metamodels for MOSS/MOWS is another future work. A preliminary proposition for a pattern was published (GOTTARDI; BRAGA, 2016), however, further validation is required to accept it as pattern.

More tools for MOWS development are also being built. Besides supporting design and coding, it has been suggested to create tools for MOWS testing. Preliminary results indicate that it is possible to create test cases based on models.

The MOSS and MOWS principles could be employed to create services without the explicit usage of WS. Therefore, we also intend to study and validate if these services could be used (and possible advantages) for software architecture specifications that employ services, e.g. service-oriented architectures and micro-services.

8.6 Published Works

The purpose of this section is to cite posters, papers and articles that are derivatives of the project presented in this thesis. This section is divided into subsections according to the topic of the publication.

8.6.1 Secondary Studies

The systematic mapping presented in Chapter 3 has been published as a conference paper (GOTTARDI; BRAGA, 2015). As it was published in 2015, it was a previous version that lacks the most recent updates.

8.6.2 Process Discovery and Metric

The process discovery algorithm was initially published as a poster for a workshop on statistics (GOTTARDI; BRAGA, 2014). Afterwards, the metric was published in another poster for a more recent workshop on statistics (GOTTARDI; BRAGA, 2017a).

An extended version was submitted as an article to "Journal of Software: Evolution and Process", acceptance information is pending.

8.6.3 MOSS and MOWS

The initial proposition of MOWS systems has been published as a conference paper including a few case studies as validation (GOTTARDI; BRAGA, 2016). The comparative studies were published one year after as another conference paper (GOTTARDI; BRAGA, 2017b). An overview on MOWS and MOSS including how they were defined by using method engineering was presented as a poster (GOTTARDI; BRAGA, 2017c).

8.7 Final Remarks

This chapter concludes the main chapters of this thesis. Besides these main chapters, there is a reference section followed by four chapters. In Appendix A contains a systematic review on software process discovery, which was conducted prior to creating the algorithm and analyses presented in Chapter 4. In Appendix B there is a report on the tools created for MOSS/MOWS development. In Appendix C there are feasibility case studies that indicate how MOWS systems could be used in real world applications. The verbatim copies of the documents used during the experimental studies described in Chapter 7 are available as appendixes: Appendix D contains the documents of the data verification experiment while Appendix E contains the documents of the implementation study. Besides these appendixes, the author of this thesis would also like to invite the reader to refer to related documents and papers that are available outside this thesis[1].

[1] <http://tiny.cc/gottardi-doc>

BIBLIOGRAPHY

AALST, W. b. V. D.; ADRIANSYAH, A.; MEDEIROS, A. D.; ARCIERI, F.; BAIER, T. b.; BLICKLE, T.; BOSE, J.; BRAND, P. V. D.; BRANDTJEN, R.; BUIJS, J.; BURATTIN, A.; CARMONA, J.; CASTELLANOS, M.; CLAES, J.; COOK, J.; COSTANTINI, N.; CURBERA, F.; DAMIANI, E.; LEONI, M. D.; DELIAS, P.; DONGEN, B. V.; DUMAS, M.; DUSTDAR, S.; FAHLAND, D.; FERREIRA, D.; GAALOUL, W.; GEFFEN, F. V.; GOEL, S.; GüNTHER, C.; GUZZO, A.; HARMON, P.; HOFSTEDE, A. b. T.; HOOGLAND, J.; INGVALDSEN, J.; KATO, K.; KUHN, R.; KUMAR, A.; ROSA, M. L.; MAGGI, F.; MALERBA, D.; MANS, R.; MANUEL, A.; MCCREESH, M.; MELLO, P.; MENDLING, J.; MONTALI, M.; MOTAHARI-NEZHAD, H.; MUEHLEN, M. Z.; MUNOZ-GAMA, J.; PONTIERI, L.; RIBEIRO, J.; ROZINAT, A.; PéREZ, H. S.; PéREZ, R. S.; SEPúLVEDA, M.; SINUR, J.; SOFFER, P.; SONG, M.; SPERDUTI, A.; STILO, G.; STOEL, C.; SWENSON, K.; TALAMO, M.; TAN, W.; TURNER, C.; VANTHIENEN, J.; VARVARESSOS, G.; VERBEEK, E.; VERDONK, M.; VIGO, R.; WANG, J.; WEBER, B.; WEIDLICH, M.; WEIJTERS, T.; WEN, L.; WESTERGAARD, M.; WYNN, M. Process mining manifesto. **Lecture Notes in Business Information Processing**, v. 99 LNBIP, n. PART 1, p. 169–194, 2012. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84863011087&partnerID=40&md5=480ca64ccc688944f7d185d63564261a>. Citations on pages 87, 211, 212, 220, 221, and 225.

AALST, W. V. D. Business process configuration in the cloud: How to support and analyze multi-tenant processes? In: . [s.n.], 2011. p. 3–10. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-80655149730&partnerID=40&md5=b3dc1adf33632b4c640e4bec44dc5798>. Citations on pages 220 and 221.

AALST, W. van der. Process mining: Overview and opportunities. **ACM Transactions on Management Information Systems**, ACM, New York, NY, USA, v. 3, n. 2, p. 7:1–7:17, Jul. 2012. ISSN 2158-656X. Available: <http://doi.acm.org/10.1145/2229156.2229157>. Citations on pages 46, 79, 220, and 221.

ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. **Knowledge and Data Engineering, IEEE Transactions on**, v. 17, n. 6, p. 734–749, 2005. ISSN 1041-4347. Citation on page 214.

AKMAN, B.; DEMIRöRS, O. Applicability of process discovery algorithms for software organizations. In: . [s.n.], 2009. p. 195–202. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-74549134602&partnerID=40&md5=114544a986739d4280ccc28e060c7d2d>. Citations on pages 87, 212, 220, 221, 222, 224, and 225.

ALKADI, I.; CARVER, D. A testing assistant for object-oriented programs. In: **Aerospace Conference, 1998 IEEE**. [s.n.], 1998. v. 4, p. 149–158 vol.4. ISSN 1095-323X. Available: <http://dx.doi.org/10.1109/AERO.1998.682164>. Citation on page 63.

ALMEIDA, J. a. P.; DIJKMAN, R.; SINDEREN, M. van; PIRES, L. F. Platform-independent modelling in mda: Supporting abstract platforms. In: **Proceedings of the 2003 European Conference on Model Driven Architecture: Foundations and Applications**. Berlin, Heidelberg:

Springer-Verlag, 2003. (MDAFA'03), p. 174–188. ISBN 3-540-28240-8, 978-3-540-28240-2. Available: <http://dx.doi.org/10.1007/11538097_12>. Citations on pages 11, 41, and 42.

AMBLER, S. **Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process**. Wiley, 2002. ISBN 9780471271901. Available: <https://www.wiley.com/en-us/Agile+Modeling%3A+Effective+Practices+for+eXtreme+Programming+and+the+Unified+Process-p-9780471271901>. Citation on page 76.

APACHE. **Apache HTTP Server Project: Modules**. 2016. Available: <https://httpd.apache.org/modules/>. Citation on page 123.

Apache Foundation. **Apache CXF 3.2**. 2017. Available: <http://cxf.apache.org/>. Citations on pages 124, 125, 184, 231, 232, and 233.

ARKHANGEL'SKIĬ, A. V.; FEDORCHUK, V. V. The basic concepts and constructions of general topology. In: ____. **General Topology I: Basic Concepts and Constructions Dimension Theory**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990. p. 1–90. ISBN 978-3-642-61265-7. Available: <http://dx.doi.org/10.1007/978-3-642-61265-7_1>. Citations on pages 86, 88, and 93.

ASADI, M.; RAMSIN, R. Mda-based methodologies: An analytical survey. In: SCHIEFERDECKER, I.; HARTMAN, A. (Ed.). **Model Driven Architecture – Foundations and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 419–431. ISBN 978-3-540-69100-6. Available: <http://dx.doi.org/10.1007/978-3-540-69100-6_30>. Citations on pages 68, 71, 73, and 76.

ASADI, M.; RAVAKHAH, M.; RAMSIN, R. An mda-based system development lifecycle. In: **Proceedings - 2nd Asia International Conference on Modelling and Simulation, AMS 2008**. Kuala Lumpur: [s.n.], 2008. p. 836–842. ISBN 9780769531366. Citations on pages 11, 41, 43, 44, 68, 69, and 76.

ASSMANN, U.; GÖTZ, S.; JÉZÉQUEL, J.-M.; MORIN, B.; TRAPP, M. A reference architecture and roadmap for models@run.time systems. In: ____. **Models@run.time: Foundations, Applications, and Roadmaps**. Cham: Springer International Publishing, 2014. p. 1–18. ISBN 978-3-319-08915-7. Available: <http://dx.doi.org/10.1007/978-3-319-08915-7_1>. Citations on pages 48, 103, and 114.

AUTILI, M.; BELLMAN, K. L.; DIACONESCU, A.; ESTERLE, L.; TIVOLI, M.; ZISMAN, A. Transition strategies for increasing self-awareness in existing types of computing systems. In: ____. **Self-Aware Computing Systems**. Cham: Springer International Publishing, 2017. p. 305–336. ISBN 978-3-319-47474-8. Available: <http://dx.doi.org/10.1007/978-3-319-47474-8_10>. Citation on page 49.

AUTILI, M.; INVERARDI, P.; TIVOLI, M. Automated integration of service-oriented software systems. In: ____. **Fundamentals of Software Engineering: 6th International Conference, FSEN 2015, Tehran, Iran, April 22-24, 2015. Revised Selected Papers**. Cham: Springer International Publishing, 2015. p. 30–45. ISBN 978-3-319-24644-4. Available: <http://dx.doi.org/10.1007/978-3-319-24644-4_2>. Citation on page 49.

BADREDDIN, O.; FORWARD, A.; LETHBRIDGE, T. C. A test-driven approach for developing software languages. In: **2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.: s.n.], 2014. p. 225–234. Citations on pages 49 and 109.

BADREDDIN, O.; LETHBRIDGE, T. C. Model oriented programming: Bridging the code-model divide. In: **2013 5th International Workshop on Modeling in Software Engineering (MiSE)**. [s.n.], 2013. p. 69–75. ISSN 2156-7883. Available: <http://dx.doi.org/10.1109/MiSE.2013.6595299>. Citations on pages 49, 103, and 114.

BAHSOON, R.; EMMERICH, W. Economics-driven software mining. In: **Proceedings of the First International Workshop on The Economics of Software and Computation**. Washington, DC, USA: IEEE Computer Society, 2007. (ESC '07), p. 3–. ISBN 0-7695-2955-0. Available: <http://dx.doi.org/10.1109/ESC.2007.5>. Citations on pages 220, 221, and 224.

BECK, K. **Extreme Programming Explained: Embrace Change**. Addison-Wesley, 2000. (An Alan R. Apt Book Series). ISBN 9780201616415. Available: <https://www.pearson.com/us/higher-education/product/Beck-Extreme-Programming-Explained-Embrace-Change/9780201616415.html>. Citation on page 76.

BELAUNDE, M.; STEINHAU, R.; ALMEIDA, J. P.; GAVRAS, A. **MODA-TEL Model-Driven methodology**. Heidelberg, Germany, 2004. Available: <http://web.archive.org/web/20040721052651/http://www.modatel.org/~Modatel/pub/deliverables/D3.add1-final.pdf>. Citations on pages 41 and 42.

BENDRAOU, R.; DESFRAY, P.; GERVAIS, M.-P. c.; MULLER, A. Mda tool components: A proposal for packaging know-how in model driven development. **Software and Systems Modeling**, v. 7, n. 3, p. 329–343, 2008. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-47149092831&partnerID=40&md5=b14b86e8d869ed3dea69cbe947b2bf26>. Citations on pages 68, 69, and 70.

BENNACEUR, A.; FRANCE, R.; TAMBURRELLI, G.; VOGEL, T.; MOSTERMAN, P. J.; CAZZOLA, W.; COSTA, F. M.; PIERANTONIO, A.; TICHY, M.; AKŞIT, M.; EMMANUELSON, P.; GANG, H.; GEORGANTAS, N.; REDLICH, D. Mechanisms for leveraging models at runtime in self-adaptive software. In: ____. **Models@run.time: Foundations, Applications, and Roadmaps**. Cham: Springer International Publishing, 2014. p. 19–46. ISBN 978-3-319-08915-7. Available: <http://dx.doi.org/10.1007/978-3-319-08915-7_2>. Citations on pages 48 and 114.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. 2nd. ed. Addison-Wesley Professional, 2005. 496 p. ISBN 978-0321267979. Available: <https://www.pearson.com/us/higher-education/program/Booch-Unified-Modeling-Language-User-Guide-The-2nd-Edition/PGM206461.html>. Citation on page 38.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. **Model-driven Software Engineering in Practice**. Morgan & Claypool, 2012. (G - Reference, Information and Interdisciplinary Subjects Series). ISBN 9781608458820. Available: <https://mdse-book.com>. Citations on pages 29, 30, 35, 36, 38, 39, 40, 78, 112, 118, and 239.

BRINKKEMPER, S. Method engineering: engineering of information systems development methods and tools. **Information and software technology**, Elsevier, v. 38, n. 4, p. 275–280, 1996. Available: <http://doc.utwente.nl/18012/1/Brinkkemper96method.pdf>. Citations on pages 44, 45, 46, and 105.

BUIJS, J.; DONGEN, B. V.; AALST, W. V. D. Towards cross-organizational process mining in collections of process models and their executions. **Lecture Notes in**

Business Information Processing, v. 100 LNBIP, n. PART 2, p. 2–13, 2012. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84856582430&partnerID=40&md5=fd1c7d09a185f73aa4ec1331a684a63b>. Citations on pages 220 and 221.

BUIJS, J. C. A. M.; DONGEN, B. F. V.; AALST, W. M. P. Van der. A genetic algorithm for discovering process trees. In: **Evolutionary Computation (CEC), 2012 IEEE Congress on**. [S.l.: s.n.], 2012. p. 1–8. Citations on pages 220 and 221.

BURATTIN, A.; VIGO, R. A framework for semi-automated process instance discovery from decorative attributes. In: . [s.n.], 2011. p. 176–183. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-79961188189&partnerID=40&md5=4710c56c71958d281b645bdc04cd54bc>. Citations on pages 220 and 221.

CERNICKINS, A.; NIKIFOROVA, O.; OZOLS, K.; SEJANS, J. An outline of conceptual framework for certification of mda tools. In: **Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modelling Theory-Driven Development, MDA and MTDD 2010, in Conjunction with ENASE 2010**. Athens: [s.n.], 2010. p. 60–69. ISBN 9789898425164. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-78650898686&partnerID=40&md5=52aab18285c2d6c360dfbd6e451329ba>. Citations on pages 68, 69, and 70.

CHITFOROUSH, F.; YAZDANDOOST, M.; RAMSIN, R. Methodology support for the model driven architecture. In: **14th Asia-Pacific Software Engineering Conference (APSEC'07)**. [s.n.], 2007. p. 454–461. ISBN 0769530575; 9780769530574. ISSN 1530-1362. Available: <http://dx.doi.org/10.1109/APSEC.2007.69>. Citations on pages 41, 43, 68, 69, and 76.

CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. **Web Services Description Language (WSDL) 1.1**. 2001. Available: <http://www.w3.org/TR/wsdl/>. Citations on pages 50, 132, 231, and 232.

CLAES, J.; POELS, G. Integrating computer log files for process mining: A genetic algorithm inspired technique. **Lecture Notes in Business Information Processing**, v. 83 LNBIP, p. 282–293, 2011. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-79960289591&partnerID=40&md5=01316b5efb9df0cd3b31ff48959eef41>. Citations on pages 220 and 221.

CodePlex. **Mo-Plus**. 2016. Available: <http://moplus.codeplex.com/>. Citation on page 113.

COOK, J. E.; WOLF, A. L. Toward metrics for process validation. In: **Proceedings. Third International Conference on the Software Process, 1994.** [s.n.], 1994. p. 33–44. Available: <https://ieeexplore.ieee.org/document/344426/>. Citation on page 47.

____. Automating process discovery through event-data analysis. In: . [s.n.], 1995. p. 73–82. Available: <https://dl.acm.org/citation.cfm?id=225021>. Citations on pages 79, 86, 220, and 221.

____. Discovering models of software processes from event-based data. **ACM Transactions on Software Engineering and Methodology**, v. 7, n. 3, p. 215–249, 1998. Available: <https://dl.acm.org/citation.cfm?id=287001>. Citations on pages 47, 79, 86, 105, 220, and 221.

DUAN, B.; SHEN, B. Software process discovery using link analysis. In: . [s.n.], 2011. p. 60–63. Available: <http://dx.doi.org/10.1109/ICCSN.2011.6014218>. Citations on pages 220 and 221.

DUAN, S.; BABU, S. Processing forecasting queries. In: **Proceedings of the 33rd International Conference on Very Large Data Bases**. VLDB Endowment, 2007. (VLDB '07), p. 711–722. ISBN 978-1-59593-649-3. Available: <http://dl.acm.org/citation.cfm?id=1325851.1325933>. Citations on pages 220 and 221.

DURELLI, R. S.; SANTIBáñEZ, D. S. M.; DELAMARO, M. E.; CAMARGO, V. V. de. Towards a refactoring catalogue for knowledge discovery metamodel. In: **Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)**. [s.n.], 2014. p. 569–576. Available: <http://dx.doi.org/10.1109/IRI.2014.7051940>. Citation on page 41.

Eclipse Foundation. **ATL Transformation Language**. 2009. Available: <https://eclipse.org/atl>. Citations on pages 78 and 230.

____. **Graphical Modeling Framework, version 1.5.0**. 2011. Available: <http://www.eclipse.org/modeling/gmp/>. Citations on pages 38, 39, and 40.

____. **The CDO Model Repository**. 2013. Available: <http://www.eclipse.org/cdo/>. Citation on page 40.

____. **Edapt – Framework for Ecore model adaptation and instance migration**. 2013. Available: <http://www.eclipse.org/eadapt/>. Citation on page 40.

____. **MoDisco – Framework for Model Based Reengineering**. 2013. Available: <http://www.eclipse.org/modisco/>. Citation on page 40.

____. **Acceleo 3**. 2015. Available: <https://eclipse.org/acceleo>. Citations on pages 230, 231, 232, and 233.

____. **Eclipse Modeling Framework**. 2015. Available: <https://eclipse.org/modeling/emf/>. Citations on pages 126, 134, 231, and 239.

____. **XText: Language Engineering**. 2016. Available: <http://eclipse.org/Xtext/>. Citations on pages 126, 230, and 234.

____. **Eclipse IDE – Oxygen**. 2017. Available: <https://eclipse.org/>. Citation on page 126.

ENGELEN, R. V.; GALLIVAN, K. *et al.* The gSOAP toolkit for web services and peer-to-peer computing networks. In: IEEE. **Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on**. [S.l.], 2002. p. 128–128. Citations on pages 123, 231, and 233.

ER, E.; TEKINERDOGAN, B. Modsel: Model-driven software evolution language. In: MERNIK, M. (Ed.). **Formal and Practical Aspects of Domain-Specific Languages: Recent Developments**. [S.l.: s.n.], 2012. p. 572–594. Citations on pages 68, 69, 70, and 71.

ESPINAZO-PAGáN, J.; CUADRADO, J. S.; GARCíA-MOLINA, J. Morsa: a scalable approach for persisting and accessing large models. In: **Proceedings of the 14th international conference on Model driven engineering languages and systems**. Berlin, Heidelberg: Springer-Verlag, 2011. (MODELS'11), p. 77–92. ISBN 978-3-642-24484-1. Available: <http://dl.acm.org/citation.cfm?id=2050655.2050665>. Citation on page 40.

FABBRI, S.; HERNANDES, E. M.; THOMMAZO, A. D.; BELGAMO, A.; ZAMBONI, A.; SILVA, C. Managing literature reviews information through visualization. In: MACIASZEK, L. A.; CUZZOCREA, A.; CORDEIRO, J. (Ed.). **ICEIS (2)**. [S.l.]: SciTePress, 2012. p. 36–45. ISBN 978-989-8565-11-2. Citation on page 214.

FERREIRA, D. c.; ZACARIAS, M. c. c.; MALHEIROS, M.; FERREIRA, P. Approaching process mining with sequence clustering: Experiments and findings. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4714 LNCS, p. 360–374, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-38049188156&partnerID=40&md5=bd2eea6c3305fb2d55f1692f063b1651>. Citations on pages 220 and 221.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. In: **Proceedings of the 22Nd International Conference on Software Engineering**. New York, NY, USA: ACM, 2000. (ICSE '00), p. 407–416. ISBN 1-58113-206-9. Available: <http://doi.acm.org/10.1145/337180.337228>. Citations on pages 50, 122, 123, and 124.

FINK, G. A. **Markov Models for Pattern Recognition**. Springer, London, 2014. 366 p. (Advances in Computer Vision and Pattern Recognition). ISBN 978-1-4471-6308-4. Available: <https://doi.org/10.1007/978-1-4471-6308-4>. Citation on page 47.

FORWARD, A.; BADREDDIN, O.; LETHBRIDGE, T. C. Umple: Towards combining model driven with prototype driven system development. In: . Fairfax, VA, United states: [s.n.], 2010. ISSN 10746005. Available: <http://dx.doi.org/10.1109/RSP.2010.5656338>. Citations on pages 49 and 180.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: **2007 Future of Software Engineering**. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 37–54. ISBN 0-7695-2829-5. Available: <http://dx.doi.org/10.1109/FOSE.2007.14>. Citations on pages 29, 31, 36, 48, and 93.

GARCÍA-BORGOÑÓN, L.; BARCELONA, M.; GARCÍA-GARCÍA, J.; ALBA, M.; ESCALONA, M. Software process modeling languages: A systematic literature review. **Information and Software Technology**, Elsevier, p. 103 – 116, 2014. Citation on page 225.

GHOSE, A.; KOLIADIS, G.; CHUENG, A. Rapid business process discovery (r-bpd). **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4801 LNCS, p. 391–406, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-38349106534&partnerID=40&md5=436f601ed853474503a8ece2cd6676b2>. Citations on pages 220 and 221.

GIESE, H.; BENCOMO, N.; PASQUALE, L.; RAMIREZ, A. J.; INVERARDI, P.; WÄTZOLDT, S.; CLARKE, S. Living with uncertainty in the age of runtime models. In: ____. **Models@run.time: Foundations, Applications, and Roadmaps**. Cham: Springer International Publishing, 2014. p. 47–100. ISBN 978-3-319-08915-7. Available: <http://dx.doi.org/10.1007/978-3-319-08915-7_3>. Citations on pages 48, 114, and 118.

GOTTARDI, T.; BRAGA, R. T. V. Markov method and software process models: Research opportunities. In: **4th Workshop in Stochastic Modeling**. São Carlos: [s.n.], 2014. Available: <http://wsm.icmc.usp.br>. Citation on page 195.

____. Model driven development success cases for domain-specific and general purpose approaches: A systematic mapping. In: URP,SPC,UCSP. **XVIII CIbSE**. Lima-Peru: UCSP, 2015. p. 432–445. ISBN 978-9972-825-80-4. Citations on pages 71, 118, 128, and 195.

____. Model-oriented web services. In: **2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)**. [s.n.], 2016. p. 14–23. Available: <http://dx.doi.org/10.1109/SOSE.2016.64>. Citations on pages 69, 71, 129, 193, 195, 196, 238, and 247.

____. A formalization for software process discovery and comparison metric. In: **5th WPSM**. São Carlos: [s.n.], 2017. Citation on page 195.

____. Model-oriented web service implementations compared to traditional web services. In: **2017 IEEE International Conference on Information Reuse and Integration (IRI)**. [s.n.], 2017. p. 315–324. Available: <http://dx.doi.org/10.1109/IRI.2017.50>. Citation on page 196.

____. A study on the evolution of model-driven software engineering. In: **1º Encontro Paulista de Estudantes de Pós-Graduação**. São Carlos: [s.n.], 2017. Citation on page 196.

HASSAN, A. The road ahead for mining software repositories. In: . [s.n.], 2008. p. 48–57. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-57849119318&partnerID=40&md5=e25288b632480394b119feae52974b98>. Citations on pages 220 and 221.

HERRINGTON, J. **Code Generation in Action**. Manning, 2003. (In Action Series). ISBN 9781930110977. Available: <https://books.google.co.uk/books?id=VHVC8WnSgbYC>. Citations on pages 41, 77, and 113.

HESS, H. M. Aligning technology and business: Applying patterns for legacy transformation. **IBM Systems Journal**, v. 44, n. 1, p. 25–45, 2005. ISSN 0018-8670. Citation on page 41.

HOFFNAGLE, G. F.; BEREGI, W. E. Automating the software development process. **IBM Systems Journal**, v. 24, n. 2, p. 102–120, 1985. ISSN 0018-8670. Citation on page 61.

HOVSEPYAN, A.; SCANDARIATO, R.; BAELEN, S. V.; BERBERS, Y.; JOOSEN, W. From aspect-oriented models to aspect-oriented code?: the maintenance perspective. In: **AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development**. New York, NY, USA: ACM, 2010. p. 85–96. ISBN 978-1-60558-958-9. Citations on pages 68, 69, and 70.

HUO, M.; ZHANG, H.; JEFFERY, R. An exploratory study of process enactment as input to software process improvement. In: . [s.n.], 2006. p. 39–44. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84885581035&partnerID=40&md5=d7aa6cdcb2f7e0821b5c85e8b5ba0aeb>. Citations on pages 220 and 221.

JACOBSON, I. **Object-oriented software engineering: a use case driven approach**. [S.l.]: ACM Press, 1992. (ACM Press Series). ISBN 9780201544350. Citation on page 112.

JENSEN, C.; SCACCHI, W. Data mining for software process discovery in open source software development communities. In: **26th International Conference on Software Engineering - W17S Workshop "International Workshop on Mining Software Repositories (MSR 2004)**. Edinburgh, UK: [s.n.], 2004. p. 96–100(4). ISBN 0 86341-432 X. Available: <http://digital-library.theiet.org/content/conferences/10.1049/ic_20040484>. Citation on page 46.

____. Experiences in discovering, modeling, and reenacting open source software development processes. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 3840 LNCS, p. 449–462, 2006. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-33745163214&partnerID=40&md5=6315d0d77368c21276f87b98252470fc>. Citations on pages 220 and 221.

____. Guiding the discovery of open source software processes with a reference model. **IFIP International Federation for Information Processing**, v. 234, p. 265–270, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-34547957663&partnerID=40&md5=c5c38d931b86649331b70aed47289ef7>. Citations on pages 220 and 221.

JIN, W.; YANG, J. A flexible graph pattern matching framework via indexing. In: **Proceedings of the 23rd International Conference on Scientific and Statistical Database Management**. Berlin, Heidelberg: Springer-Verlag, 2011. (SSDBM'11), p. 293–311. ISBN 978-3-642-22350-1. Available: <http://dl.acm.org/citation.cfm?id=2032397.2032421>. Citations on pages 220, 221, and 224.

KANG, K. C.; LEE, J.; DONOHOE, P. Feature-oriented product line engineering. **IEEE Software**, v. 19, n. 4, p. 58–65, Jul 2002. ISSN 0740-7459. Available: <http://dx.doi.org/10.1109/MS.2002.1020288>. Citation on page 110.

KEHRER, T.; KELTER, U.; PIETSCH, P.; SCHMIDT, M. Adaptability of model comparison tools. In: **Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering**. New York, NY, USA: ACM, 2012. (ASE 2012), p. 306–309. ISBN 978-1-4503-1204-2. Available: <http://doi.acm.org/10.1145/2351676.2351731>. Citation on page 40.

KIM, G.-W.; LEE, S.; KIM, J.; SON, J. An effective algorithm for business process mining based on modified fp-tree algorithm. In: . [s.n.], 2010. p. 119–123. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-77952373545&partnerID=40&md5=6f4a2cc8504237b1342a0c3fb4afd16b>. Citations on pages 220 and 221.

KIM, K. Mining workflow processes from distributed workflow enactment event logs. **Knowledge Management and E-Learning**, v. 4, n. 4, p. 528–553, 2012. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84873622442&partnerID=40&md5=a834b77779f1b31193f935f7f4d59641>. Citations on pages 220 and 221.

KINDLER, E.; RUBIN, V.; SCHäFER, W. Incremental workflow mining for process flexibility. In: . [s.n.], 2006. v. 236, p. 178–187. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84884344890&partnerID=40&md5=7356e2649029ad3e30f91b8487b22453>. Citations on pages 220 and 221.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. UK, 2007. Available: <http://www.dur.ac.uk/ebse/resources/guidelines/Systematic-reviews-5-8.pdf>. Citations on pages 52, 72, and 212.

KOSKI, T.; NOBLE, J. **Bayesian Networks: An Introduction**. Wiley, 2011. 366 p. (Wiley Series in Probability and Statistics). ISBN 9781119964957. Available: <https://www.wiley.com/en-us/Bayesian+Networks%3A+An+Introduction-p-9781119964957>. Citation on page 47.

KUMAR, K.; WELKE, R. J. Challenges and strategies for research in systems development. In: COTTERMAN, W. W.; SENN, J. A. (Ed.). New York, NY, USA: John Wiley & Sons, Inc., 1992. chap. Methodology Engineering: A Proposal for Situation-specific Methodology Construction, p. 257–269. ISBN 0-471-93175-6. Available: <http://dl.acm.org/citation.cfm?id=133549.133574>. Citation on page 44.

LAKSHMANAN, G.; KHALAF, R. Leveraging process-mining techniques. **IT Professional**, v. 15, n. 5, p. 22–30, 2013. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.

0-84886510439&partnerID=40&md5=9032f80acf747518aa7c01b0c8cf0c92>. Citations on pages 79, 87, 212, 220, 221, 222, and 225.

LEMOS, A.; SABINO, C.; LIMA, R.; OLIVEIRA, C. Using process mining in software development process management: A case study. In: . [s.n.], 2011. p. 1181–1186. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-83755207396&partnerID=40&md5=1f12c81550375c97437e28d5a0bde5c2>. Citations on pages 220 and 221.

LI, J. b.; LIU, D. b.; YANG, B. b. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4537 LNCS, p. 396–407, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-38049066975&partnerID=40&md5=3a00de15e22c0c7085a65dc383ed114e>. Citations on pages 220 and 221.

MAGGI, F.; MOOIJ, A.; AALST, W. M. P. Van der. User-guided discovery of declarative process models. In: **Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on**. [S.l.: s.n.], 2011. p. 192–199. Citations on pages 220 and 221.

MANTZ, F.; TAENTZER, G.; LAMO, Y.; WOLTER, U. Co-evolving meta-models and their instance models: A formal approach based on graph transformation. **Science of Computer Programming**, Elsevier, v. 104, n. 1, p. 2–43, 2015. ISSN 01676423. Available: <http://dx.doi.org/10.1016/j.scico.2015.01.002>. Citations on pages 68, 69, and 70.

MEDEIROS, A. D.; WEIJTERS, A.; AALST, W. V. D. Genetic process mining: An experimental evaluation. **Data Mining and Knowledge Discovery**, v. 14, n. 2, p. 245–304, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-33947207063&partnerID=40&md5=9f312d0040414fbf3036f88f3783b0a5>. Citations on pages 220 and 221.

MENDELSON, E. **Introduction to mathematical logic**. [S.l.]: Wadsworth & Brooks/Cole Advanced Books & Software, 1987. (Wadsworth & Brooks/Cole mathematics series). ISBN 9780534066246. Citations on pages 87, 90, and 91.

NAZARI, P. M. S.; RUMPE, B. Using software categories for the development of generative software. In: **2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.: s.n.], 2015. p. 498–503. Citations on pages 68, 69, and 70.

NIKIFOROVA, O.; NIKULSINS, V.; SUKOVSKIS, U. Integration of mda framework into the model of traditional software development. **Frontiers in Artificial Intelligence and Applications**, IOS Press, v. 187, n. 1, p. 229–239, 2009. ISSN 09226389. Available: <http://dx.doi.org/10.3233/978-1-58603-939-4-229>. Citations on pages 68 and 69.

NIKULSINS, V.; NIKIFOROVA, O. Adapting software development process towards the model driven architecture. In: **2008 The Third International Conference on Software Engineering Advances**. [S.l.: s.n.], 2008. p. 394–399. Citations on pages 68, 69, and 76.

NOYER, A.; IYENGHAR, P.; PULVERMUELLER, E.; ENGELHARDT, J.; PRAMME, F.; BIKKER, G. A model-based workflow from specification until validation of timing requirements in embedded software systems. In: . Institute of Electrical and Electronics Engineers Inc., 2015. p. 166–169. ISBN 9781467377119. Available: <http://dx.doi.org/10.1109/SIES.2015.7185056>. Citations on pages 68, 69, and 70.

Object Management Group. **Software & Systems Process Engineering Metamodel Specification (SPEM)**. [S.l.], 2008. Available: <http://www.omg.org/spec/SPEM/2.0>. Citation on page 224.

____. **MOF Model to Text Transformation Language (MOFM2T), 1.0.** 2008. Available: <http://www.omg.org/spec/MOFM2T/1.0/>. Citation on page 241.

____. **Overview and guide to OMG's Model Driven Architecture**. http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2010. Http://www.omg.org/cgi-bin/doc?omg/03-06-01. Available: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>. Citations on pages 30, 36, 37, and 41.

____. **Unified Modeling Language Infrastructure Specification**. http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/, 2010. Http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/. Available: <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/>. Citations on pages 38 and 39.

____. **XML Metadata Interchange 2.4.2**. 2014. Available: <http://www.omg.org/spec/XMI/2.4.2/>. Citations on pages 50, 115, 118, 134, 135, and 146.

OLLE, T. W.; VERRIJN-STUART, A. Methods and associated tools for the information systems life cycle. In: **Proceedings of the IFIP WG8. 1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle**. Maastricht, The Netherlands, 26-28 september: [s.n.], 1994. Citations on pages 45, 46, and 99.

Oracle Corporation. **Java API for XML Web Services (JAX-WS)**. 2017. Available: <http://jax-ws.java.net>. Citation on page 290.

PARR, T. **The Definitive ANTLR 4 Reference**. 2nd. ed. [S.l.]: Pragmatic Bookshelf, 2013. ISBN 1934356999, 9781934356999. Citation on page 230.

PASTOR, O.; MOLINA, J. C. **Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling**. Secaucus, NJ, USA: Springer-Verlag New York, 2007. ISBN 3540718672. Available: <https://link.springer.com/book/10.1007/978-3-540-71868-0>. Citations on pages 29, 30, 35, 36, 39, 40, 41, 73, 112, and 117.

PAUTASSO, C.; WILDE, E. RESTful web services: Principles, patterns, emerging technologies. In: **Proceedings of the 19th International Conference on World Wide Web**. New York, NY, USA: ACM, 2010. (WWW '10), p. 1359–1360. ISBN 978-1-60558-799-8. Available: <http://doi.acm.org/10.1145/1772690.1772929>. Citation on page 183.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. RESTful web services vs. "big"' web services: Making the right architectural decision. In: **Proceedings of the 17th International Conference on World Wide Web**. New York, NY, USA: ACM, 2008. (WWW '08), p. 805–814. ISBN 978-1-60558-085-2. Available: <http://doi.acm.org/10.1145/1367497.1367606>. Citation on page 183.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: **12th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.: s.n.], 2008. v. 17, p. 1. Citation on page 52.

PONCIN, W.; SEREBRENIK, A.; BRAND, M. V. D. Mining student capstone projects with FRASR and ProM. In: . [s.n.], 2011. p. 87–95. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-81355139714&partnerID=40&md5=74b0f172fdb46aea13c7fecdb6d18ac1>. Citations on pages 220 and 221.

____. Process mining software repositories. In: **Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering**. Washington, DC, USA: IEEE Computer Society, 2011. (CSMR '11), p. 5–14. ISBN 978-0-7695-4343-7. Available: <http://dx.doi.org/10.1109/CSMR.2011.5>. Citations on pages 220 and 221.

PORTER, A.; SELBY, R. Metric-driven analysis and feedback systems for enabling empirically guided software development. In: **Proceedings. ICSE, 13th International Conference on Software Engineering, 1991**. [s.n.], 1991. p. 288–298. Available: <http://dx.doi.org/10.1109/ICSE.1991.130654>. Citation on page 46.

PRESSMAN, R. S.; MAXIM, B. R. **Software Engineering: A Practitioner's Approach**. 8th. ed. [S.l.]: McGraw-Hill Education, 2015. (McGraw-Hill higher education). ISBN 978-0-07-802212-8. Citation on page 211.

PUNITHA, K.; CHITRA, S. Software defect prediction using software metrics - a survey. In: **Information Communication and Embedded Systems (ICICES), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 555–558. Citation on page 47.

PéREZ-CASTILLO, R.; SáNCHEZ-GONZáLEZ, L.; PIATTINI, M.; GARCíA, F.; GUZMáN, I.-R. D. Obtaining thresholds for the effectiveness of business process mining. In: . [s.n.], 2011. p. 453–462. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84858720211&partnerID=40&md5=d775bb6c875c7aa77db8bf1ebe5ec6eb>. Citations on pages 220 and 221.

ROQUES, A. **PlantUML**. 2017. Available: <http://plantuml.com/>. Citations on pages 232 and 233.

RUBIN, V. b.; GüNTHER, C.; AALST, W. V. D.; KINDLER, E.; DONGEN, B. V.; SCHäFER, W. Process mining framework for software processes. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 4470 LNCS, p. 169–181, 2007. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-37149028185&partnerID=40&md5=769432474cfd1c4da469afd643586e7d>. Citations on pages 220 and 221.

SAMALIKOVA, J.; KUSTERS, R.; TRIENEKENS, J.; WEIJTERS, T.; SIEMONS, P. Toward objective software process information: Experiences from a case study. **Software Quality Journal**, v. 19, n. 1, p. 101–120, 2011. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-78751647167&partnerID=40&md5=035dd2959accd1bc35b3bbea6cd2a709>. Citations on pages 220 and 221.

SANCHEZ, P.; BARREDA, J.; OCON, J. Integration of domain-specific models into a mda framework for time-critical embedded systems. In: **2008 International Workshop on Intelligent Solutions in Embedded Systems**. [S.l.: s.n.], 2008. p. 1–15. Citations on pages 68, 69, and 70.

SCHMIDT, D. C. Model-driven engineering. **IEEE Computer**, IEEE, v. 39, n. 2, February 2006. Available: <http://www.truststc.org/pubs/30.html>. Citations on pages 29, 31, and 36.

SCHOFIELD, C.; TANSEY, B.; ZHENCHANG, X.; STROULIA, E. Digging the development dust for refactoring. In: . [s.n.], 2006. v. 2006, p. 23–32. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-33845409754&partnerID=40&md5=9988faf527b285d8f7097df8deaf41b6>. Citations on pages 220 and 221.

SCHONIG, S.; ZEISING, M.; JABLONSKI, S. Adapting association rule mining to discover patterns of collaboration in process logs. In: **Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on**. [S.l.: s.n.], 2012. p. 531–534. Citations on pages 220 and 221.

SCHREIER, S. Modeling RESTful applications. In: **Proceedings of the Second International Workshop on RESTful Design**. New York, NY, USA: ACM, 2011. (WS-REST '11), p. 15–21. ISBN 978-1-4503-0623-2. Available: <http://doi.acm.org/10.1145/1967428.1967434>. Citation on page 132.

SEIFERT, T.; BENEKEN, G.; BAEHR, N. Engineering long-lived applications using MDA. In: **Proceedings of the Eighth IASTED International Conference on Software Engineering and Applications**. [S.l.: s.n.], 2004. p. 241–246. Citations on pages 68, 69, and 70.

SELBY, R. Measurement-driven dashboards enable leading indicators for requirements and design of large-scale systems. In: **Software Metrics, 2005. 11th IEEE International Symposium**. [s.n.], 2005. p. 10 pp.–22. ISSN 1530-1435. Available: <http://dx.doi.org/10.1109/METRICS.2005.31>. Citation on page 224.

____. Analytics-driven dashboards enable leading indicators for requirements and designs of large-scale systems. **Software, IEEE**, v. 26, n. 1, p. 41–49, Jan 2009. ISSN 0740-7459. Available: <http://dx.doi.org/10.1109/MS.2009.4>. Citations on pages 79 and 86.

SELBY, R.; PORTER, A.; SCHMIDT, D.; BERNEY, J. Metric-driven analysis and feedback systems for enabling empirically guided software development. **Software, IEEE**, v. 7, n. 2, p. 46–54, March 1991. ISSN 0740-7459. Available: <http://dx.doi.org/10.1109/52.50773>. Citations on pages 47 and 86.

SELBY, R. W.; PORTER, A. A.; SCHMIDT, D. C.; BERNEY, J. Metric-driven analysis and feedback systems for enabling empirically guided software development. In: . [s.n.], 1991. p. 288–298. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-0026238345&partnerID=40&md5=ef220a74d549511fd32b78e138de0102>. Citations on pages 86, 220, 221, and 223.

SHANG, W.; ZHEN, M.; ADAMS, B.; HASSAN, A. MapReduce as a general framework to support research in mining software repositories (MSR). In: . [s.n.], 2009. p. 21–30. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-70349904977&partnerID=40&md5=e06ad81c31cc0d58875aedcb516ea784>. Citations on pages 220 and 221.

SOMMERVILLE, I. **Software Engineering**. 10th. ed. [S.l.]: Pearson, 2015. 816 p. ISBN 978-0133943030. Citation on page 41.

TAKAHASHI, R.; MURAOKA, Y.; NAKAMURA, Y. Building software quality classification trees: approach, experimentation, evaluation. In: **Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on**. [S.l.: s.n.], 1997. p. 222–233. Citation on page 47.

TEPPOLA, S.; PARVIAINEN, P.; TAKALO, J. Challenges in deployment of model driven development. In: **Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on**. [S.l.: s.n.], 2009. p. 15–20. Citations on pages 68, 69, 71, and 149.

TRAPP, M.; SCHNEIDER, D. Safety assurance of open adaptive systems – a survey. In: ____. **Models@run.time: Foundations, Applications, and Roadmaps**. Cham: Springer International Publishing, 2014. p. 279–318. ISBN 978-3-319-08915-7. Available: <http://dx.doi.org/10.1007/978-3-319-08915-7_11>. Citation on page 49.

VERBEEK, H.; BUIJS, J.; DONGEN, B. V.; AALST, W. V. D. ProM 6: The process mining toolkit. In: **CEUR Workshop Proceedings**. [S.l.: s.n.], 2010. v. 615, p. 34–39. Citations on pages 223 and 224.

W3C Working Group. **Web Services Glossary**. 2004. Available: <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>. Citations on pages 49, 115, 118, 122, 134, and 146.

____. **SOAP Version 1.2**. 2007. Available: <http://www.w3.org/TR/soap12/>. Citations on pages 50, 123, and 124.

____. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. 2008. Available: <http://www.w3.org/TR/2008/REC-xml-20081126/>. Citations on pages 49 and 142.

W3C XML Working Group. **W3C XML Schema Definition Language (XSD) 1.1**. 2012. Available: <http://www.w3.org/TR/xmlschema11-1/>. Citations on pages 50, 117, 118, 122, and 134.

WALICKI, M.; FERREIRA, D. Sequence partitioning for process mining with unlabeled event logs. **Data and Knowledge Engineering**, v. 70, n. 10, p. 821–841, 2011. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-80051787234&partnerID=40&md5=48977d525325c9ebf0fae52de0a09f4c>. Citations on pages 220 and 221.

WANG, D. b.; GE, J. b.; HU, H.; LUO, B. b. A new process mining algorithm based on event type. In: . [s.n.], 2011. p. 1144–1151. Available: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84862964984&partnerID=40&md5=106c1a86bae79aa83c36641ced0cb64c>. Citations on pages 220 and 221.

WESTFECHTEL, B. Merging of EMF models: Formal foundations. **Software and Systems Modeling**, Springer Verlag, v. 13, n. 2, p. 757–788, 2014. ISSN 16191366. Available: <http://dx.doi.org/10.1007/s10270-012-0279-3>. Citations on pages 68, 69, and 70.

WHITTLE, J. Agile versus mde - friend or foe? In: LARA, J. de; RUSCIO, D. D.; PIERANTONIO, A. (Ed.). **XM@MoDELS**. [S.l.]: CEUR-WS.org, 2013. (CEUR Workshop Proceedings, v. 1089), p. 1. Citations on pages 71, 73, 76, 79, and 149.

WHITTLE, J.; HUTCHINSON, J.; ROUNCEFIELD, M.; BURDEN, H.; HELDAL, R. Industrial adoption of model-driven engineering: Are the tools really the problem? In: MOREIRA, A.; SCHÄTZ, B.; GRAY, J.; VALLECILLO, A.; CLARKE, P. J. (Ed.). **MoDELS**. [S.l.]: Springer, 2013. (Lecture Notes in Computer Science, v. 8107), p. 1–17. ISBN 978-3-642-41532-6. Citations on pages 30, 58, 71, 73, and 76.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**. [S.l.]: Springer, 2012. (Computer Science). ISBN 9783642290442. Citations on pages 93 and 98.

YU, Y.; TUN, T. T.; BANDARA, A. K.; ZHANG, T.; NUSEIBEH, B. From model-driven software development processes to problem diagnoses at runtime. In: ____. **Models@run.time: Foundations, Applications, and Roadmaps**. Cham: Springer International Publishing, 2014. p.

188–207. ISBN 978-3-319-08915-7. Available: <http://dx.doi.org/10.1007/978-3-319-08915-7_7>. Citations on pages 68, 69, and 70.

APPENDIX

A

SYSTEMATIC REVIEW ON SOFTWARE PROCESS DISCOVERY

A.1 Initial Remarks

Some software development projects are successful without a preliminary definition of the employed process. The development team may be capable to create an adequate process while the project is being executed. To allow reproducibility of these processes, several authors devised techniques for process discovery. There is a numerous set of approaches suited for several process model types. It is important to identify the most adequate approaches for software development process models. Therefore, we have conducted a systematic review aimed to identify different software process discovery approaches and their suitability to become integrated into project support tools. We have identified the most common process models and discuss about their suitability. As result analysis, we could categorize approaches according to their attributes. Another contribution is providing an overview of the state of the art related to this context, encouraging prospections of trends in software development processes.

A.2 Systematic Review Definition

Software Processes are the foundation of Software Engineering. Processes allow the rational and systematic development of software products, which are beneficial to their quality and effective delivery (PRESSMAN; MAXIM, 2015).

Software development processes can exist without being planned before the start of the life cycle or without proper documentation. Software engineers working on similar projects would be interested on the documentation in order to make this process repeatable. Therefore, it is important to gather enough information about the non documented process execution in order to discover the process model and generate its documentation (AALST *et al.*, 2012).

The main goal of this appendix chapter is to present a cohesive secondary study highlighting the studies that contain adequate algorithms for process mining and their expected output process model formats. We are specifically interested in software development processes in which supporting tools were used and can be a source for discovering the corresponding processes. During this process, we identified primary studies related to process mining or discovery, as well as surveys and algorithm comparisons (AALST *et al.*, 2012; LAKSHMANAN; KHALAF, 2013; AKMAN; DEMIRöRS, 2009).

Process discovery can be also performed while the development is being executed. In this study, we call these approaches "active process discovery". By applying mining algorithms while the development is under way, the intent is to guide the developers to follow specific paths and to design the possible outcomes by using statistical models. These and other factors are reviewed and summarized in this study.

It is possible to highlight main contributions: (i) to provide an overview of existing process discovery approaches; (ii) to identify the most common process models output formats and if they are appropriate; and (iii) to provide software engineers a way to easily identify the present state of the art of process discovery, as well as related secondary studies.

A.3 Systematic Review Method

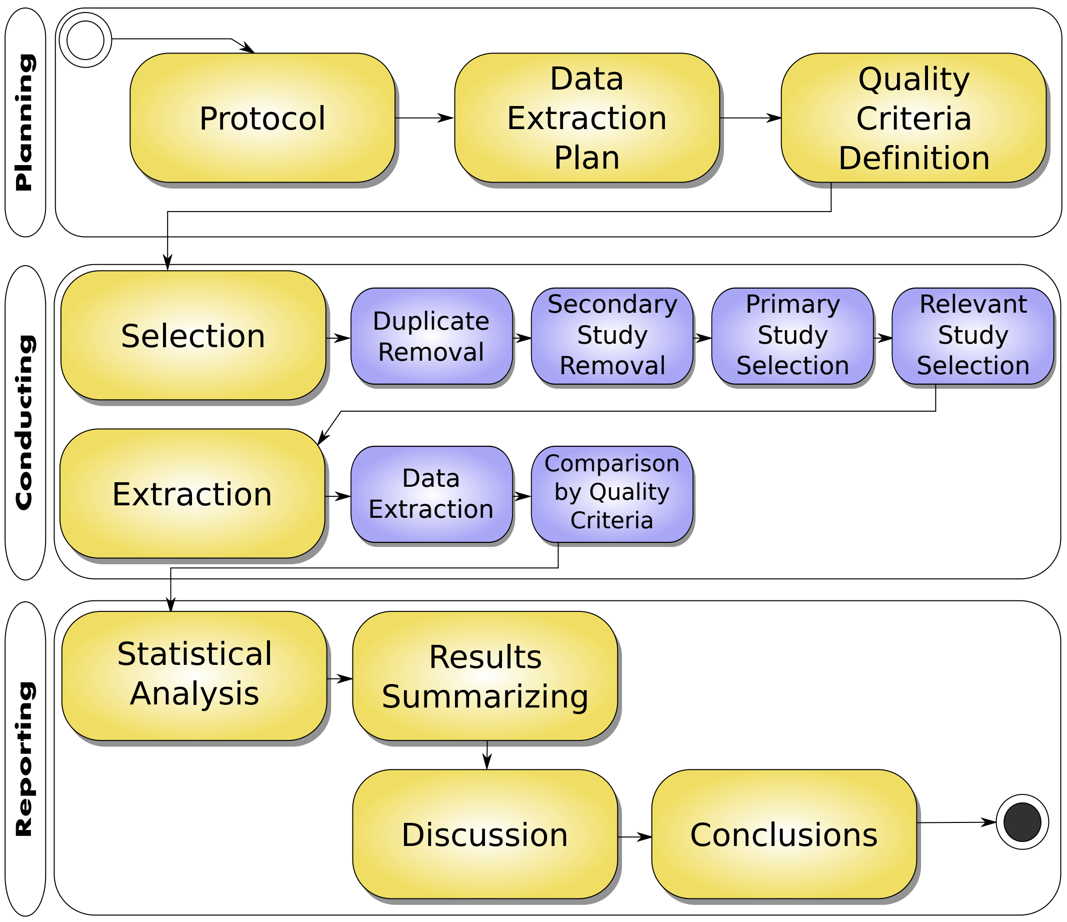
This study has been undertaken as a systematic review based on the guidelines proposed by Kitchenham and Charters (KITCHENHAM; CHARTERS, 2007). According to them, in order to conduct a systematic review, it is advisable to follow three main phases: (*i*) planning the review, (*ii*) conducting the review and (*iii*) reporting the review. Each phase produces an intermediate outcome, as can be seen in Figure 62. More details about each main phase are described in the following sections.

A.3.1 Planning the Systematic Review

During the planning phase, the 'Protocol' is defined, which contains: the Research Questions (RQs), the search strategy, the inclusion and exclusion criteria.

RQs must embody the review study purpose. Moreover, these questions reflect the general scope of the review study. The scope is comprised of population (i.e., population group observed by the intervention), intervention (i.e., what is going to be observed in the context of the planned review study), and outcomes of relevance (i.e., the results of the intervention). Furthermore, during the conduction of this step, it was necessary to establish the scope of the review study. According to the systematic review process (KITCHENHAM; CHARTERS, 2007), the scope has to be established using the PICO criteria (Population, Intervention, Comparison, Outcomes). Thus, herein our **Population** is published scientific literature reporting on some Software Process Discovery approaches. The **Intervention** is published scientific literature regarding existing

Figure 62 – Adopted Review Process Model Diagram



Source: Created by the author

algorithms and implementation studies that should be analyzed. The **Comparison** is made by comparing extracted data as established by a planned extraction form. Finally, the **Outcomes of relevance** is an overview of the studies that have been conducted in the field of algorithms and automated tools for software process discovery, emphasizing primary studies that report on the techniques used in the research area. From observing such an aggregated data set, we also intend to provide insights into the frequencies of publication over time to inspect trends.

The planning also involves the 'Data Extraction Plan', which includes a form definition for extracting data from each primary study. In the 'Quality Criteria Definition', the researchers define the expected attributes of the studies to be considered as high quality and trustworthy.

An excerpt of the protocol definition is visible on Table 61. The two columns of the table are arranged into field name and value pairs that include the objective, questions, intervention, control, population, comparison outcomes, application, keywords, source selection criteria, study language, search engines and study selection criteria. Among these fields, the questions and study selection criteria are frequently referenced during the conduction phase presented in Section A.4.

The most important item of the protocol is its objective, in which we intended to identify studies that present algorithms that would be adequate for a software process reuse project. From this objective we derived two RQs, which are also listed within the protocol (namely, Primary Question and Secondary Question).

A.3.2 Conducting the Systematic Review

Afterwards, an exhaustive search for papers was carried out to answer these RQs. First of all, the keywords that are also part of the protocol were selected for the search. As this selection was known to be relevant for the quality of results, general terms were used with the aim of confirming that most of the research papers were included in the study.

After a pilot testing phase using the keywords shown on Table 62 we could create the search string. As can be seen on this table, for each row, the conjunction operation (represented by "$\wedge$") is applied while the disjunction operation (represented by "$\vee$") is applied to each cell where there are synonyms of each keyword. Therefore, the final search string is $((A_1) \vee (A_2) \vee (A_3) \vee (A_4)) \wedge ((B_1 \vee B_2)) \wedge ((C_1 \vee C_2))$.

Then, the search was carried out through the following search engines: ACM Digital Library, IEEE Xplore, Google Scholar and Elsevier Scopus. Throughout this study we have used a tool called StArt (State of the Art through Systematic Review). It aims to help the researcher, giving support to the application of systematic review (FABBRI *et al.*, 2012). During the extraction process, the data of each primary study was independently gathered by three reviewers. The review was performed in the first semester of 2014 and then updated during early 2015 by two Ph.D. students and an expert in software engineering; the achieved results were crossed and then validated. All the results of the search process are documented in the web material[1]. Therefore, it is clear to others how thorough the search was, and how they can find the same documents.

During the review phases, every paper/study was evaluated considering the inclusion and exclusion criteria. As defined in the protocol, there are five inclusion criteria and six exclusion criteria, defined as I_n and N_m, respectively, as seen on Table 61. The first inclusion criterion is applied to studies that show a process discovery algorithm, whereas the second is applied to studies that present development approaches that allow process discovery while the development cycles are still being executed. The third criterion is applied to studies that present tool implementation of the algorithm; the fourth is used on studies that present analysis of the application of process discovery on real projects. Finally, the last criterion is mapped to any studies that involve recommender systems for process models. Recommender Systems are software tools that employ statistical operations in order to identify the most similar objects to recommend them to interested users (ADOMAVICIUS; TUZHILIN, 2005).

The exclusion criteria are applied to studies that cannot pass to the extraction or sum-

[1] <http://tiny.cc/gottardi-doc>

Table 61 – Protocol Definition

Protocol Item	Item Description
Objective	Identify the most adequate process discovery algorithm for a process reuse project.
Primary Question	What are the existing software process discovery approaches?
Secondary Question	What is the most adequate process discovery approach for project support integration?
Intervention	The existing algorithms and implementation studies must be analysed.
	The search results must involve a list of studies related to the questions that are known by the researchers. This list includes articles by Selby, Porter, Cook, Wolf, Akman and Demirörs.
Population	Software Process discovery approaches
Comparison	The studies must be compared and evaluated according to their presented algorithm and or presented study related to application of process discovery
Outcome	A comprehensive list of algorithms and automated tools for software process discovery
Application	This secondary study is provided as support to research regarding process discovery.
Keywords:	process discovery; process; software.
Source selection criteria:	Source must index studies on Computer Science, Mathematics or Engineering. Source must allow Boolean operators. Source must be accessible by the researchers.
Study Language:	at least title and abstract must be in English.
Search Engines:	ACM Digital Library; IEEE Xplore; Google Scholar; Elsevier Scopus
Selection Criteria:	**Inclusion:** • I_1 - Presents Process Discovery Algorithm; • I_2 - Presents Active Participation on Process Discovery; • I_3 - Presents Automated Tool for Process Discovery; • I_4 - Presents Study Related to Application of Process Discovery; • I_5 - Presents Process Recommender System; **Exclusion:** • E_1 - Not a primary study; • E_2 - Does not fit any inclusion criteria; • E_3 - Unrelated to Software and Business Process; • E_4 - Unrelated to Software Engineering; • E_5 - Inaccessible; • E_6 - Not related to Process Discovery.

Table 62 – Search String Definition

Identifier	Keyword	Synonyms and Related
A	Process Discovery	1. empirical guided process; 2. empirically guided process; 3. process discovery; 4. process mining
B	Software	1. software; 2. software system;
C	Process	1. process; 2. process model;

marizing phases. The first planned exclusion criterion is applied to results that are not primary studies. The second is applied to studies that do not apply to any considered inclusion criteria. The third and forth are applied to studies that deal with process models unrelated to this review. The last two exclusion criteria were only used during extraction phase, they are used to identify studies that could not be accessed or that did not present information related to process discovery, despite being selected during 'Selection' phase.

The data extraction form contains fields that must be filled during the 'Extraction' phase. The planned form contains ten fields: (X_1) Uses Markov Models; (X_2) Uses Bayesian Models; (X_3) Uses Petri-Nets; (X_4) Related to Software Process; (X_5) Related to Business Process; (X_6) Process Discovery Participation Type; (X_7) Adaptative Process Discovery; (X_8) Interactive Process Discovery; (X_9) Incremental Process Discovery; (X_{10}) Identified Algorithm Description.

The valid values for each of the enumerated fields are presented on Table 63. These valid values are defined either as text or as sets of nominals, i.e., groups of enumerated and named items that represent categories important to our study.

Through the first to the fifth field and the seventh to the ninth, we have specified boolean fields that are used to establish whether the reviewed study is related to the usage of Markov Models, Petri-nets, Bayesian Models, Software Process Models, Business Process Models, Interactive, Incremental, and Adaptative process discoveries. The sixth field is used to categorize

Table 63 – Valid Values for Data Extraction Fields

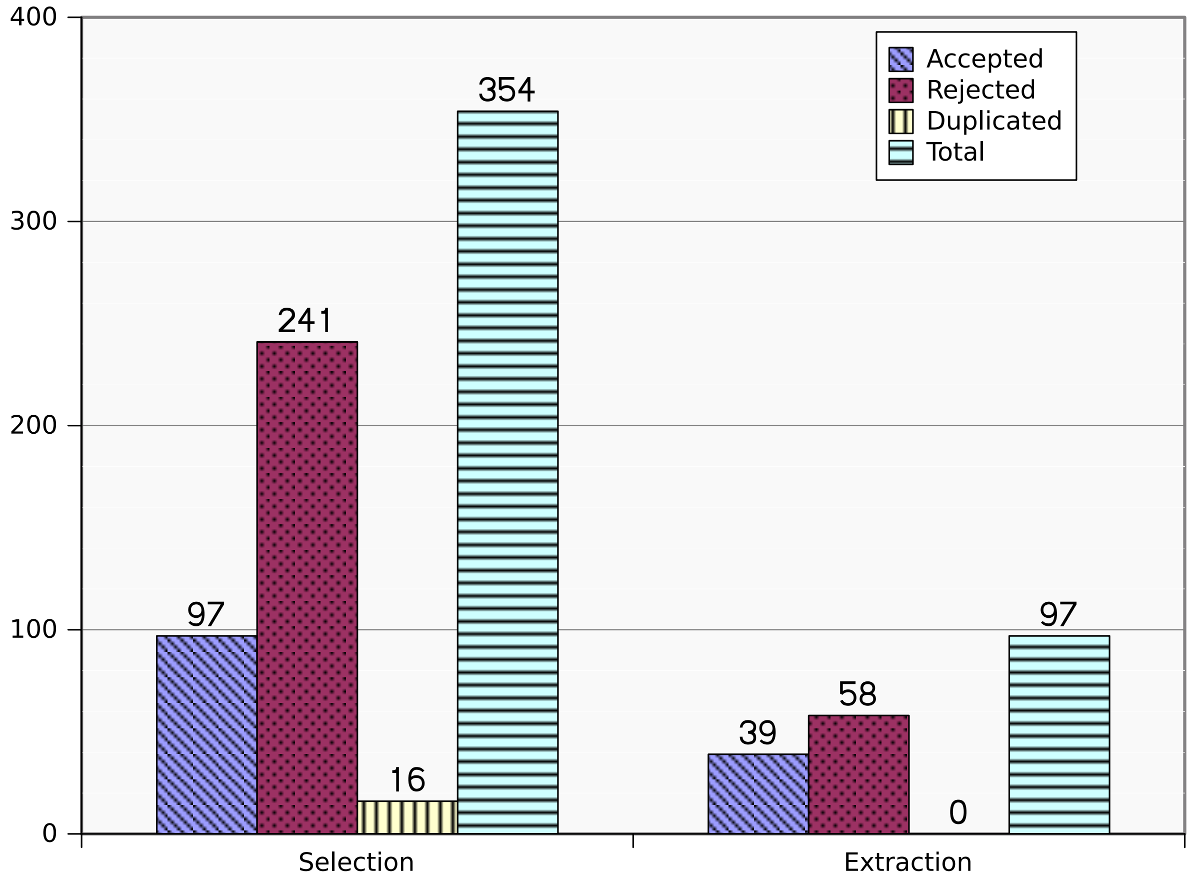
Number	Field Name	Field Type	Nominal Set
X_1	Uses Markov Models	Boolean	True
			False
X_2	Uses Petri-Nets	Boolean	True
			False
X_3	Uses Bayesian Models	Boolean	True
			False
X_4	Related to Software Models	Boolean	True
			False
X_5	Related to Business Models	Boolean	True
			False
X_6	Process Discovery Participation Type	Enumeration	Active
			Passive
X_7	Adaptative Process Discovery	Boolean	True
			False
X_8	Interactive Process Discovery	Boolean	True
			False
X_9	Incremental Process Discovery	Boolean	True
			False
X_{10}	Identified Algorithm Description	Text	Free Text

the discovery approach into active or passive. Finally, the last field of the extraction form was planned to allow the researchers to describe the identified algorithms.

After ensuring that there were no duplicated studies, the review was conducted by analyzing studies returned by each search engine. The search engine priority was set by reviewing first the engine that returned the least number of studies. In this manner, the selected order was, from first to last: Google Scholar, IEEE Explore, ACM Digital Library and Elsevier Scopus.

Study Quality Criteria are a set of rules that are planned in order to define which articles are carried into extraction and comparison. It has been established that the studies need to be accepted on at least an inclusion criterion and without any exclusion criterion in order to be carried to the summarizing (or reporting) phase. It was also expected that high quality studies would actually present an algorithm and/or a study related to the application of process discovery.

Figure 63 – Studies Reviewed During Each Phase



Source: Created by the author

A.3.3 *Reporting the Systematic Review*

The ‘Reporting’ is the last main phase. It is composed by a statistical analysis on the results, which is then summarized. This summarizing is then presented in this appendix chapter (Section A.4) and documented in the packing. These results are then discussed in ‘Discussion’ (presented in Section A.5) and finally, the ‘Conclusions’ are reported (Section A.7).

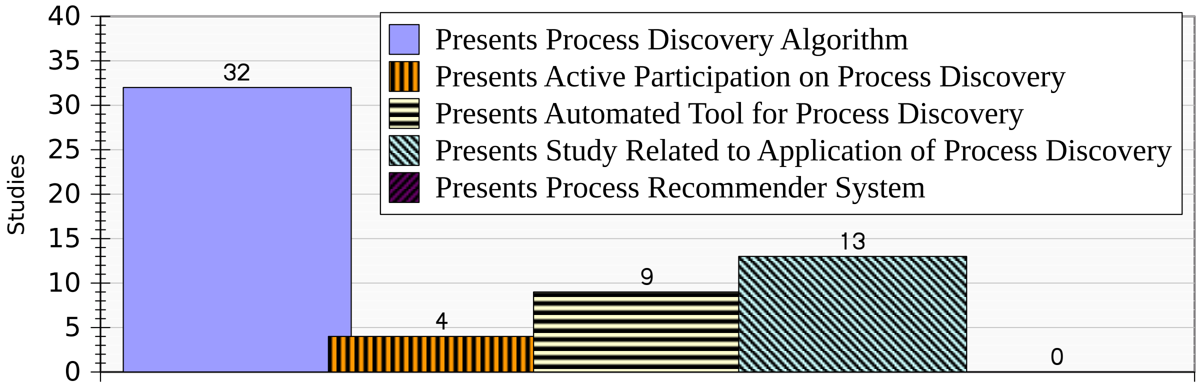
A.4 Systematic Review Results

This section contains the results gathered during the review conduction. It includes the search results and the extraction results.

Our planned search string was executed and returned 108, 81, 3 and 162 from ACM Digital Library, IEEE Xplore, Google Scholar and Elsevier Scopus, respectively.

There is a bar plot in Figure 63 that allows quick visualization of the quantity of studies reviewed during each phase and how many have been selected, rejected or marked as duplicated.

Figure 64 – Selected Studies by Inclusion Criterion



Source: Created by the author

It is important to mention that the studies selected during the selection phase are exactly the same as the total reviewed during extraction. The studies selected during extraction are the ones presented in summarizing, which are referenced as ESS (Extraction-Selected Studies).

Figure 64 contains a plot showing ESS distributed according to the inclusion criteria. It is important to mention that the sum of these categories surpasses the total number of studies because many of them fit into more than one category.

Figure 65 contains a plot which represents the quantity of studies selected during selection and extraction phases distributed according to the each search engine. This plot was not created considering the duplicated studies.

The plot in Figure 66 represents the distribution of studies and their acceptance or rejection according to each review phase. Since the extraction phase follows the selection phase, the studies accepted for selection are distributed between rejected for extraction and accepted for extraction. The plot also includes the amount of studies rejected for selection.

The ESS are listed on Table 64. This table contains three columns which contain data of published year, reference number and the extracted algorithm description for every one of the 39 ESS. This description was extracted as the last field of the extraction form.

The complete results of relating the studies to the inclusion criteria and the extraction fields numbered from one to nine are presented on Table 65. It is important to establish that these inclusion criterion were defined on Table 61 while the extraction fields were defined on Table 6. The boolean values are represented as "**T**" for "True", "**F**" for "False", "**P**" for "Passive" and "**A**" for "Active". It is important to remind that the specification of each inclusion and field definition can be found in Section A.3. As the last row of this table, we also present the total number of True and Active values.

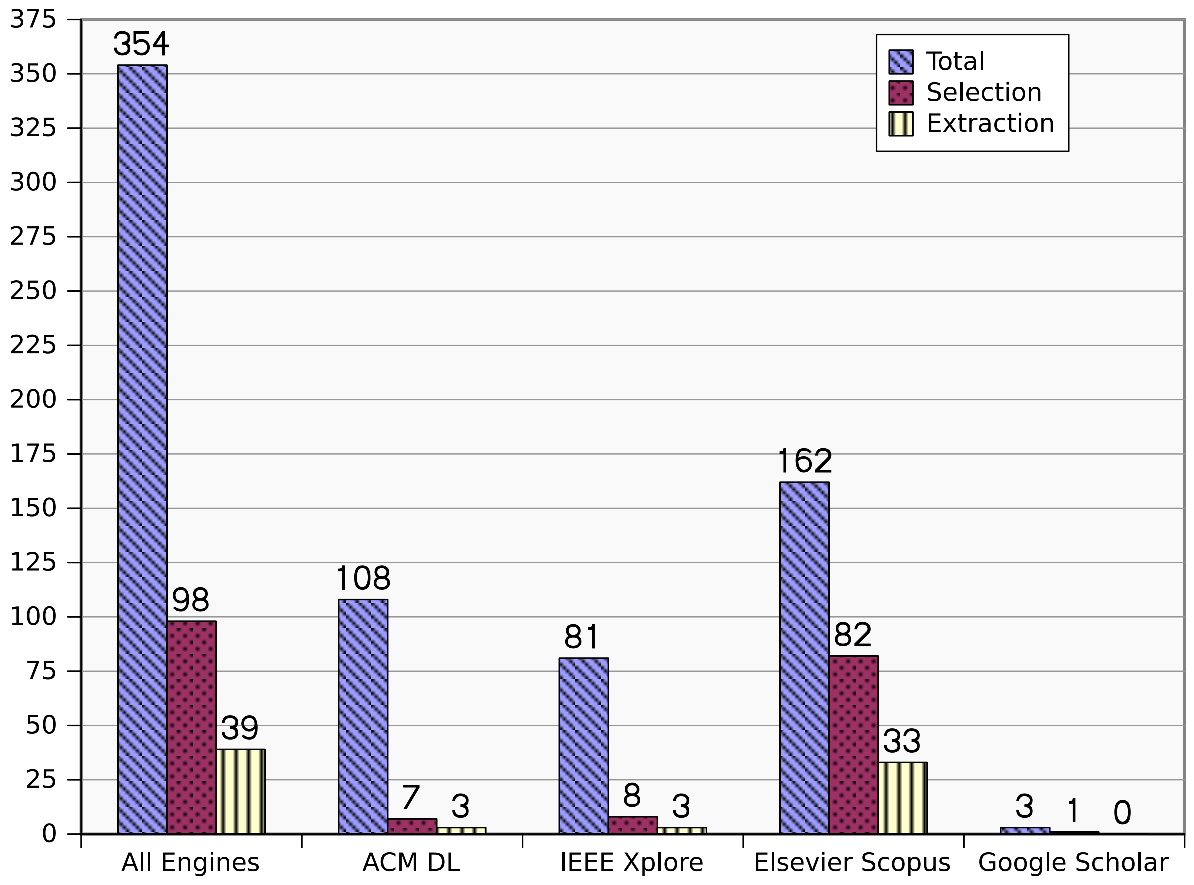
Table 64 – Studies Elected for Extraction

Year	Ref.	Description
2007	Bahsoon and Emmerich (2007)	Algorithm that focuses on financial factors.
2007	Duan and Babu (2007)	Presents an algorithm based on Bayesian networks. If involves node graph sequence inference by calculating the predicted numerical values.
2011	Jin and Yang (2011)	Algorithm that allows similarity detection among parts of a process graph.
2012	Buijs, Dongen and Aalst (2012b)	Presents a genetic algorithm for process mining employing Petri-nets which avoids deadlock. Displays quality criteria with four dimensions for process discovery.
2011	Maggi, Mooij and Aalst (2011)	Configurable language to perform a data/process mining.
2012	Schonig, Zeising and Jablonski (2012)	Algorithm to identify collaboration of people in business processes.
2013	Lakshmanan and Khalaf (2013)	Presents different types of algorithms and a decision tree to choose which is best for each case.
2012	Kim (2012)	Describes an algorithm for finding a validated workflow. Use of formal language. Conducts mining workflow system and unifies the process.
2012	Aalst (2012)	Describes a generic method to conduct process mining.
2012	Aalst *et al.* (2012)	Basic literature of process mining.
2012	Buijs, Dongen and Aalst (2012a)	Evaluates algorithms of process discovery. There is no definition of a particular algorithm.
2011	Lemos *et al.* (2011)	Presents a study of running mining process on real data implementing software process. Shows the use of tool and reliability.
2011	Pérez-Castillo *et al.* (2011)	Provides a statistical algorithm to identify the most suitable process for context.
2011	Wang *et al.* (2011)	Describes an algorithm for mining of business processes based on event types. Generates Petri nets. Seems applicable to software process despite not being officially described. Presents incremental operations model to optimize the result.
2011	Poncin, Serebrenik and Brand (2011a)	Shows the application of the discovery process in software projects of undergraduates. Offers a simple and usual metamodel for discovery.
2011	Aalst (2011)	Shows a distributed process discovery algorithm that takes execution log data captured from several subsystems distributed in the cloud.
2011	Walicki and Ferreira (2011)	Shows an algorithm that allows to detect whether specific untagged records are related to a process execution instance. This generic approach is applicable to several process model types.
2011	Duan and Shen (2011)	Displays discovery process from SVN code repository logs. Shows examples of using the tool for Java, C and C + +. Does not show the algorithm itself.
2011	Burattin and Vigo (2011)	Definition of an generic algorithm for process discovery. Supports exploiting preliminary known context data.
2011	Claes and Poels (2011)	Algorithm to merge logs to facilitate mining.
2011	Poncin, Serebrenik and Brand (2011b)	Describes process mining tool for source code repositories. Includes conceptual model and metamodel.
2011	Samalikova *et al.* (2011)	Shows studies on the application discovery process of real software systems.
2010	Kim *et al.* (2010)	Shows a process usage detection algorithm in a comprehensive manner.
2009	Akman and Demirörs (2009)	Evaluates four different algorithms in the context of software process. Related to RNet, Ktail, Markov (COOK; WOLF, 1998) and open repositories (JENSEN; SCACCHI, 2007) SCM (Aalst)
2009	Shang *et al.* (2009)	Shows the implementation of a distributed system for distributed mining of source code repositories system.
2008	Hassan (2008)	Displays various items to be searched in the mining of software repositories.
2007	Rubin *et al.* (2007)	Describes process mining from repository logs.
2007	Ferreira *et al.* (2007)	Describes the process of discovery algorithms that have been identified in bioinformatics and can be applied to various contexts.
2007	Li, Liu and Yang (2007)	Comparison of nodes to identify the same tasks.
2007	Ghose, Koliadis and Chueng (2007)	Displays mining business process specifically directed to the production of information systems algorithm. With validation and use of MDD/MDA.
2007	Jensen and Scacchi (2007)	Describes process models that are best discovered with the aid of a reference process model.
2007	Medeiros, Weijters and Aalst (2007)	Describes genetic mining algorithm, inputs are based on business process logs. Their definitions can be generic enough for other processes.
2006	Schofield *et al.* (2006)	Describes and compares four cases discovery code refactoring activities. It has as input the code repositories.
2006	Huo, Zhang and Jeffery (2006)	Shows a representation of a pattern of a process using Petri-nets. Statistical evaluation with Cohen's Kappa.
2006	Kindler, Rubin and Schäfer (2006)	Describes an algorithm for incremental discovery workflow that is applied from the first event record.
2006	Jensen and Scacchi (2006)	Evaluates discovering in various software processes open but only shows example in Netbeans. Shows use of PML to describe the process.
1998	Cook and Wolf (1998)	Mining Software processes from logs and employs Markov's model.
1995	Cook and Wolf (1995)	Mining Software processes from logs and employs Markov's model.
1991	Selby *et al.* (1991b)	Tree categorization. Script to capture events for calculation of metrics on demand.

Table 65 – Extraction Results

Year	**Ref.**	**I_1**	**I_2**	**I_3**	**I_4**	**I_5**	**X_1**	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**	**X_8**	**X_9**
2007	Bahsoon and Emmerich (2007)	T	F	F	F	F	F	F	F	F	T	P	T	F	F
2007	Duan and Babu (2007)	T	T	F	F	F	F	T	F	F	F	A	T	T	T
2011	Jin and Yang (2011)	T	F	F	F	F	F	F	F	F	F	P	F	F	F
2012	Buijs, Dongen and Aalst (2012b)	T	F	F	F	F	F	F	T	F	F	A	T	F	T
2011	Maggi, Mooij and Aalst (2011)	T	F	T	F	F	F	F	F	F	F	P	F	F	F
2012	Schonig, Zeising and Jablonski (2012)	T	F	F	F	F	F	F	F	F	T	A	F	F	T
2013	Lakshmanan and Khalaf (2013)	F	F	F	T	F	F	F	F	F	F	P	T	T	T
2012	Kim (2012)	T	F	F	F	F	F	F	F	F	T	P	T	T	T
2012	Aalst (2012)	F	F	F	T	F	F	F	F	F	F	P	F	F	F
2012	Aalst *et al.* (2012)	F	F	F	T	F	F	F	F	F	T	P	F	F	F
2012	Buijs, Dongen and Aalst (2012a)	T	F	T	F	F	F	F	F	F	F	P	T	F	F
2011	Lemos *et al.* (2011)	F	F	F	T	F	F	F	F	T	F	P	F	F	F
2011	Pérez-Castillo *et al.* (2011)	T	F	F	T	F	F	F	F	F	F	P	F	F	F
2011	Wang *et al.* (2011)	T	F	F	F	F	F	F	T	F	T	P	T	F	T
2011	Poncin, Serebrenik and Brand (2011a)	T	F	F	F	F	F	F	F	T	F	P	T	F	F
2011	Aalst (2011)	T	F	F	F	F	F	F	F	F	T	P	T	T	T
2011	Walicki and Ferreira (2011)	T	F	T	F	F	F	F	F	F	F	P	T	F	F
2011	Duan and Shen (2011)	T	F	F	F	F	F	F	F	T	F	P	F	F	F
2011	Burattin and Vigo (2011)	T	F	T	F	F	F	F	F	F	F	P	F	F	F
2011	Claes and Poels (2011)	T	F	F	F	F	F	F	F	F	F	P	F	F	F
2011	Poncin, Serebrenik and Brand (2011b)	T	F	T	F	F	F	F	F	T	F	P	T	T	T
2011	Samalikova *et al.* (2011)	T	T	F	T	F	F	F	F	T	F	P	T	F	F
2010	Kim *et al.* (2010)	T	F	F	F	F	F	F	F	F	T	P	F	F	T
2009	Akman and Demirörs (2009)	T	F	F	T	F	F	F	F	T	F	A	T	T	T
2009	Shang *et al.* (2009)	T	F	F	F	F	F	F	F	T	F	P	F	F	F
2008	Hassan (2008)	F	F	F	T	F	F	F	F	T	F	P	F	F	F
2007	Rubin *et al.* (2007)	T	F	F	T	F	F	F	F	T	F	A	T	T	T
2007	Ferreira *et al.* (2007)	T	F	F	T	F	F	F	F	F	F	P	T	F	F
2007	Li, Liu and Yang (2007)	T	F	F	F	F	F	F	F	F	F	P	F	F	F
2007	Ghose, Koliadis and Chueng (2007)	T	F	F	F	F	F	F	F	F	T	A	F	F	T
2007	Jensen and Scacchi (2007)	T	F	F	F	F	F	F	F	F	F	P	F	F	F
2007	Medeiros, Weijters and Aalst (2007)	T	F	T	F	F	F	F	F	F	T	P	F	F	F
2006	Schofield *et al.* (2006)	F	F	F	T	F	F	F	F	T	F	P	T	T	F
2006	Huo, Zhang and Jeffery (2006)	T	F	F	T	F	F	F	T	F	F	A	F	F	T
2006	Kindler, Rubin and Schäfer (2006)	T	T	F	F	F	F	F	F	F	T	A	T	T	T
2006	Jensen and Scacchi (2006)	F	F	F	T	F	F	F	F	T	F	P	F	F	F
1998	Cook and Wolf (1998)	T	F	T	F	F	T	F	F	T	F	A	T	T	T
1995	Cook and Wolf (1995)	T	F	T	F	F	T	F	F	T	F	A	T	T	T
1991	Selby *et al.* (1991b)	T	T	T	F	F	F	F	F	T	F	A	T	T	T
Total		32	4	9	13	0	2	1	3	14	10	11	20	12	17

Figure 65 – Sources for the Selected Studies



Source: Created by the author

A.5 Discussion

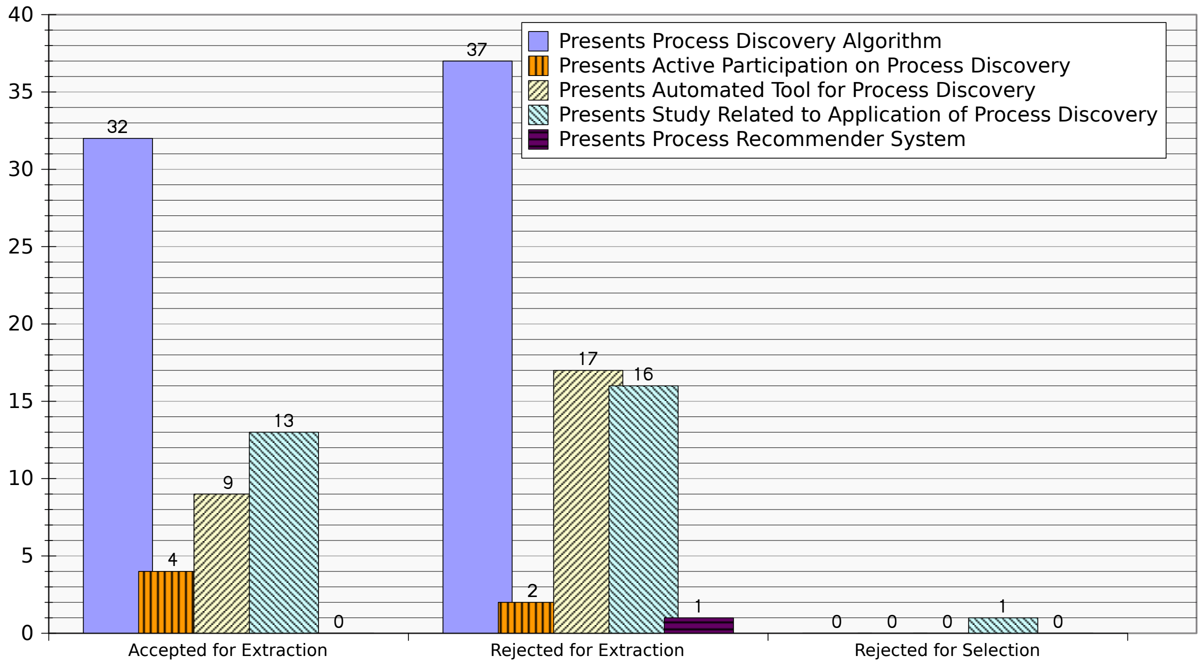
Despite originally planned, we do not present an in depth analysis of each identified algorithm, however, it was possible to identify the most common models and their challenges. Besides that, there were identified studies that provided the expected outcome (LAKSHMANAN; KHALAF, 2013; AKMAN; DEMIRöRS, 2009).

A.5.1 *Identified Algorithms*

During our searches, we have gathered different types of algorithms. For example, in the work by Akman and Demirörs four process discovery algorithms are described (AKMAN; DEMIRöRS, 2009), namely Markov Method, Heuristic Mining, Fuzzy Mining and Genetic Process Mining.

Besides these four categories, it was possible to identify algorithms based on RNets, K-Tail, Petri-Nets. Related approaches were also cited by Akman and Demirörs (AKMAN; DEMIRöRS, 2009), except for the usage of and other Bayesian Models that are not based on

Figure 66 – Studies Selected During each Phase According to Inclusion Criterion



Source: Created by the author

Markov Method, as well as tools generating higher abstraction models.

Therefore we advocate that a small update on their work would be beneficial, since they already provide the most important approaches that are still current according to our systematic review.

A.5.2 Adequate Implementations

ProM (VERBEEK *et al.*, 2010) is the most cited tool for process discovery. ProM is a toolkit for process mining that is highly configurable and extensible, allowing more algorithms to be plugged in.

However, according to our searches, there was no selected case of active discovery that employs ProM. This could be an evidence that this toolkit may not be optimized for this application. Besides that, more research is needed in order to provide conclusive evidences.

A.5.3 Active Process Discovery

Selby and Porter authored the first study that compromised a active discovery approach (SELBY *et al.*, 1991b). They have advocated that this would support the execution of an "empirical-based process".

"Empirical-Based Process" would allow developers to learn from successes and failures while guiding the process execution. Similarly a more recent work authored by Selby (SELBY, 2005), the idea is to collect project data in real-time while allowing the software engineers to constantly improve the process model.

Akman and Demirörs (AKMAN; DEMIRöRS, 2009) have identified the hypothesis that process discovery could help software industry to attain maturity levels that require process optimization, yet only 11 studies out of 39 are related to this specific case, which could indicate further research opportunities.

A.5.4 Adaptative Process Discovery

In the paper presenting ProM 6 (VERBEEK *et al.*, 2010), the authors advocate the adaptability of the toolkit. In 20 out of 39 studies, authors have discussed the benefits of adaptability. Different data sources and projects may require to be adapted in order to be successfully mined, which have become a constant concern of these authors in order to provide accurate outputs throughout projects.

A.5.5 Interactive and Incremental Process Discovery

Interactive process discoveries are approaches that allow the users conducting the discovery process to input configurations to better suit their needs. This characteristic was found in 12 out of 39 studies. Incremental process discoveries allow to continue an ongoing discovery process, accumulating more information to the output models, a characteristic found in 17 out of 39. These numbers were higher our expectations, which could require a more in depth validation of the analyzed studies in order to verify this categorization.

A.5.6 Model Category and Dichotomy

Several process discovery implementations are based on sequence graphs and formal methods, while others are based on higher level process model specifications, e.g., SPEM (Software & Systems Process Engineering Metamodel) (Object Management Group, 2008a).

Despite languages like SPEM also define graphs, a common concern that appears when discovering process is identifying when parts of the graph are isomorphic. Higher level modeling languages may have ambiguities that cause problems while trying to employ the isomorphism evaluation, making impossible the task of proving whether the process parts were repeated, which is desirable for creating a flexible process model (AKMAN; DEMIRöRS, 2009; BAHSOON; EMMERICH, 2007; JIN; YANG, 2011). subsectionsection

A.6 Works Related to this Systematic Review

We have not identified systematic reviews on the same topic of this one. Nevertheless, it is still worth mentioning related surveys and other secondary studies.

The work by Akman and Demirörs which describes the application of four process discovery algorithms (AKMAN; DEMIRöRS, 2009), namely Markov Method, Heuristic Mining, Fuzzy Mining and Genetic Process Mining.

Der Aalst *et al.*, as well as Lakshmanan and Khalaf, have written two studies that contain a secondary study focus, by presenting a set of different algorithms and their advantages (AALST *et al.*, 2012; LAKSHMANAN; KHALAF, 2013). These studies were considered by us as recommended literature for any software engineer to understand the basics of process discovery.

We believe that providing a systematic review including statistical analysis of the context could provide a more in depth secondary study for characterizing the state of art of the literature than non systematic surveys.

As a related systematic review, García-Borgoñon *et al.* have also written a work worth mentioning which presented a systematic review on software process modeling languages (GARCÍA-BORGOÑÓN *et al.*, 2014). Their work was considered related to ours since it compares different process model formats. These formats may be more or less suited when employing discovery algorithms, as discussed in Section A.5 and Chapter 4.

A.7 Conclusions and Final Remarks

In this appendix chapter, we have presented a systematic literature review on studies that present algorithms to discover software process models, presented the planned protocol, results per phases and discussion. The main contributions are: an overview of existing process discovery approaches, to identify the most common process models output formats and if they are appropriate and to provide software engineers a way to easily identify the present state of the art of process discovery, as well as related secondary studies.

We have also suggested updating the most adequate secondary studies that discuss process mining techniques and approaches after providing an state of art overview of this context.

As future work, we intend to work on the limitations of this study and carry out a more in depth analysis of each identified algorithm as well as employing it onto case studies. This work was created inside a project that is related to studying process tool support that includes process discovery and recommendation. The process discovery support must be flexible enough to allow developers to extend the discovered models to better suit their needs. Therefore, this explains our constant concern on making possible isomorphism detection. This is important because we intend to apply updated model-driven development methods by using model transformations on the discovered process models and devise new studies on their flexibility, adaptability and reuse. Further packing documents related to this secondary study are available[2].

[2] <http://tiny.cc/gottardi-doc>

APPENDIX

B

MOWS TOOLS REFERENCE

B.1 MOWS Tool-Chain

A set of cooperating tools have been developed for assisting developers who wish to build MOWS systems. These tools are referred as a tool-chain, since their usage is linked to each other and in sequential order.

A general overview of the tool-chain is shown in Figure 67. This figure contains a graph where arrows point from the input format towards output. The nodes are shaped differently depending on format Ellipses represent formats that are generic (not created for MOWS development). Rectangles represent formats created for MOWS development. Arrows are drawn differently to indicate whether the generator or dependency was created as part of this project. The letters over circles are used to refer to the description shown on Table 66 and Subsection B.1.1.

All referenced tools are available. Proposed tools have been implemented as part of this project and are functional.

B.1.1 Tool-Chain Graph Edge List

Further description for the transformers are written as paragraphs within this subsection.

A: MOWSDL to Ecore; Complete Metaclass Generation, ECore does not contain Interface Information.

B: Ecore to MOWSDL; Generates MOWSDL instance from ECore. Since ECore does not support interface information, the interface/port is added as a comment to assist the developer. The generator also saves existing interface data if the file already exists.

C: MOWSDL to XSD; Generates the XSD for the datatypes specified within the MOWSDL. Excludes method and interface declarations.

D: MOWSDL to WSDL; Generates a WSDL for the interface with port specified from

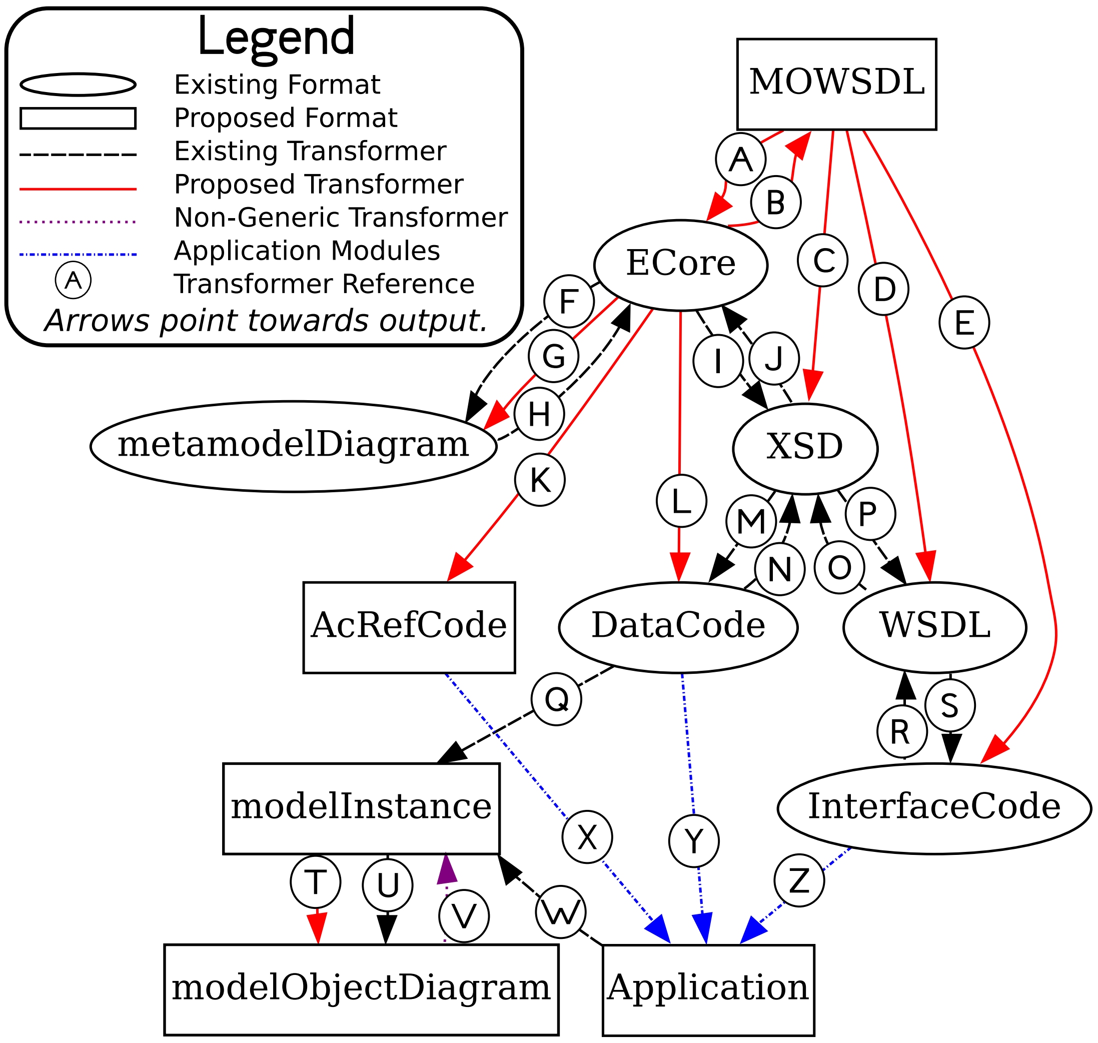


Figure 67 – Transformation Tool-chain

Source: Created by the author

the WSDL. Indirectly invokes the MOWSDL to XSD generator (C) for the datatypes when required.

E: MOWSDL to Interface Code; Generates interface code to be used by the clients and servers.

F: ECore to Metamodel Diagram; Generates a diagram model to allow developers to visualize the metamodel graphically. Part of EMF tools and depends on the Eclipse platform.

G: ECore to Metamodel Diagram; Generates a diagram model to allow developers to visualize the metamodel graphically. Provided by the author to export independent portable graphics.

H: Metamodel Diagram to ECore; Generates ecore metamodel from diagram specification. Part of EMF tools. Requires manual adaptation and compiling.

I: ECore to XSD; Generates XSD for the datatypes based on ECore. Opposite of J.

Table 66 – List of Edges of Tool-Chain Graph

Ref.	Input	Output	Origin
A	MOWSDL	ECore	Proposed Tool
B	ECore	MOWSDL	Proposed Tool
C	MOWSDL	XSD	Proposed Tool
D	MOWSDL	WSDL	Proposed Tool
E	MOWSDL	Interface Code	Proposed Tool
F	ECore	Metamodel Diagram	Part of EMF Tools
G	ECore	Metamodel Diagram	Proposed Tool
H	Metamodel Diagram	ECore	Part of EMF Tools
I	ECore	XSD	Part of EMF Tools
J	XSD	ECore	Part of EMF Tools
K	ECore	AcRefCode	Proposed Tool
L	ECore	DataCode	Proposed Tool
M	XSD	DataCode	Part of CXF Tools
N	DataCode	XSD	Part of CXF Tools
O	WSDL	XSD	Part of CXF Tools
P	XSD	WSDL	Part of CXF Tools
Q	DataCode	modelInstance	Proposed Tool
R	InterfaceCode	WSDL	Part of CXF Tools
S	WSDL	InterfaceCode	Part of CXF Tools
T	modelInstance	modelObjectDiagram	Proposed Tool
U	modelInstance	modelObjectDiagram	Part of EMF Tools
V	modelObjectDiagram	modelInstance	Part of EMF Tools
W	Application	modelInstance	Proposed Tool
X	AcRefCode	Application	Proposed Module Dependence
Y	DataCode	Application	Proposed Module Dependence
Z	InterfaceCode	Application	Proposed Module Dependence

J: XSD to ECore; Generates Ecore metamodel based on XSD, subject to compatibility verification. Opposite of I.

K: ECore to AcRefCode; Generates a code component that allows programmers to directly access model cross-references.

L: ECore to DataCode; Generates a code component that allows programmers to manipulate model objects as classes.

M: XSD to DataCode; Generates a code component that allows programmers to manipulate model objects as classes.

N: DataCode to XSD; Extracts data structures from annotated code to generate the representative XSD.

O: WSDL to XSD; Extracts the datatypes declarations inside the WSDL and exports them as an XSD.

P: XSD to WSDL; Creates an WSDL with the datatypes but without specific interface definition, since it is not covered by XSD.

Q: DataCode to modelInstance; Allows programmatic instantiation of models from the datacode.

R: InterfaceCode to WSDL; Extracts the interface from annotated code to export the representative WSDL. Opposite of S.

S: WSDL to InterfaceCode; Generates interface code for the specified WSDL. Opposite of R.

T: modelInstance to modelObjectDiagram; Generates an object diagram from an instance model that is generated by applications or editors. Part of EMF tools, requires manual adaptation and compiling for each type of metamodel.

U: modelInstance to modelObjectDiagram; Generates an object diagram from an instance model that is generated by applications or editors. Created by author to support dynamic detection of metamodel.

V: modelObjectDiagram and modelInstance; Diagram editor which generates model instances.

W: AcRefCode and Application; Proposed dependency. Handles references and dereferencing in case of models with cross-references.

X: DataCode and Application; Proposed dependency. Data structures to represent models at run-time for the application.

Z: InterfaceCode and Application; Proposed dependency. Header/interface code to implement the services provided by the web services system.

B.1.2 Proposed MOWSDL Tools

There are four tools that take MOWSDL as input, as represented by letters A, C, D and E on Table 66. These are compilers that take MOWSDL code as input and generate other artifacts. All versions use the Xtext grammar parser generator (Eclipse Foundation, 2016) (which also employs Antlr internally (PARR, 2013)).

B.1.2.1 MOWSDL to ECore

There are two editions of the ECore generator (Tool A), the first edition employs ATL (Eclipse Foundation, 2009) making it a declarative model to model transformation. The second edition employs Acceleo (Eclipse Foundation, 2015a) and generates the ECore XML via model to text transformation. Both versions are equivalent, however, the latter version is preferred as it was proven to be more stable when dealing with cross references (i.e., variable type usage). It is also intended to be integrated in an all-in-one MOWSDL compiler by generating all the resulting artifacts.

The resulting ECore artifact is created to represent the metamodel specified by the MOWSDL code. This metamodel is intended to be used as the data type and schema definition of MOWS systems (and also generic MOSS implementations). It is eventually converted to data structures in programming language and XSD to represent the XML structure managed by the MOSS.

B.1.2.2 MOWSDL to XSD

The MOWDSL to XSD (Tool C) tool is capable of generating XSD from a MOWSDL instance without using ECore as intermediate language. This generator was implemented using Acceleo (Eclipse Foundation, 2015a) and generates XSD in a model-to-text transformation.

The main advantage of using this generator is to avoid data type constraints and Eclipse Platform dependencies found in the existing ECore to XSD generator provided as part of EMF tools (Eclipse Foundation, 2015b).

The data type constraints are related to how the XSD generator exports the used data types. The EMF generator causes more dependency on the ECore data types, which could cause an overhead on decoding and does not allow the meta-metamodel to be omitted (this impact is discussed in Chapter 6).

The Eclipse Platform dependency is related to the file addressing rule used inside the platform. It is used to refer to files inside the development tool workspace and is not compatible to WS systems as required by MOWS.

B.1.2.3 MOWSDL to WSDL

The MOWSDL to WSDL generator (Tool D) transforms MOWSDL into standard WSDL (CHRISTENSEN *et al.*, 2001). This generator was implemented using Acceleo (Eclipse Foundation, 2015a) and generates WSDL via model-to-text transformation.

WSDL is used by SOAP compliant web service servers to represent their interface (ports). The generated WSDL depends on a XSD generated from the same MOWSDL, which can be done directly (via generator C) or through ECore metamodel instance. In this thesis, generated WSDL has been used to generate data structure code for Java (using Apache CXF tools (Apache Foundation, 2017)) or C++ (gSOAP tools (ENGELEN; GALLIVAN *et al.*, 2002)).

B.1.2.4 MOWSDL to Interface Code

The MOWSDL to Interface Code generator (Tool E) generates data structure code in Java. It is meant to inter-operate with Apache CXF code (Apache Foundation, 2017).

This generator was implemented using Acceleo (Eclipse Foundation, 2015a) and generates Java code via model-to-text transformation.

Initially, Apache CXF tools (Tool S) (Apache Foundation, 2017) were used to generate Java code from WSDL (CHRISTENSEN *et al.*, 2001). However, it has been noticed that this generation tool was not optimized for use with MOWS, generating data-types that were cumbersome to developers, i.e., heavy dependency on ECore and XSD classes, which was unnatural for them to code. The proposed generator, however, includes implicit conversion to fundamental Java types, allowing experienced Java developers to use the code without needing to learn ECore and XSD data types. This generator supports SOAP and REST-style interfaces: it was intended to provide interface support according to REST architectural recommendation besides SOAP, whereas the Apache CXF WSDL tool is focused on SOAP.

B.1.3 Proposed ECore Tools

There are four tools, as represented by letters B, G, K and L on Table 66. These tools take an ECore metamodel as input ECore is a meta-metamodel, i.e., allows to create metamodels. This metamodeling language is supported by the EMF tools, which were used to decode the metamodels for generating other artifacts.

B.1.3.1 ECore to MOWSDL

The ECore to MOWSDL Code generator (Tool B) generates MOWSDL code with the metamodel semantics from the metamodel in ECore language.

This tool was implemented by using Acceleo (Eclipse Foundation, 2015a) and generates MOWSDL concrete syntax text format in a model-to-text transformation.

Besides generating MOWSDL from metamodel, this tool suggests a basic interface for easy specification by the developers. In case of existing MOWSDL code used as output, this tool also attempts to save the existing definitions. As this tool is the opposite case from another proposed tool (Tool A) it allows a round-trip development strategy.

B.1.3.2 ECore to Metamodel Diagram

ECore to Metamodel Diagram generate class diagrams from the metaclasses. Create to support development whenever the EMF tool version cannot be used (Tool F).

It was implemented using a custom Java code that invokes Acceleo (Eclipse Foundation, 2015a) model-to-text transformation and transparently executes PlantUML (ROQUES, 2017) using the same Java Virtual Machine to output diagrams as image files.

This tool can also be used in conjunction with the round-trip strategy of metamodel creation using both ECore and MOWSDL (Tool A and Tool B) allowing quick visualization of the metamodel.

B.1.3.3 ECore to AcRefCode

ECore to AcRefCode (Tool K) generates Accessor and Referrer code pairs. This tool was provided in two versions, both implemented in Acceleo (Eclipse Foundation, 2015a), which generates code from models: The first version generates C++ code to be used with gSOAP2 (ENGELEN; GALLIVAN *et al.*, 2002); The second version generates Java code to be used with Java JAX and Apache CXF (Apache Foundation, 2017).

Accessor and Referrer code pairs are used in the context of the executing MOSS/MOWS application to access and create references to model objects that use cross-references.

B.1.3.4 ECore to Data Code

ECore to Data Code (Tool L) generates Data code in Java to be compatible with Apache CXF and Java JAX implementations (Apache Foundation, 2017).

It was implemented by using Acceleo (Eclipse Foundation, 2015a), which generates Java code from ECore models in a model-to-text transformation.

This tool is capable of generating data structures with methods that make the resulting application capable of reading and writing models, which is a basic requirement of MOSS (and MOWS) systems.

B.1.4 Other Proposed Tools

B.1.4.1 Data Code to Model Instance

The Data Code to Model Instance (Tool Q) is a tool that is actually generated by the ECore to Data Code (Tool L). Upon execution of the MOSS/MOWS application, the data can be written as models by using methods generated by the data code generation tool.

B.1.4.2 Model Instance to Model Object Diagram

Model Instance to Model Object Diagram (Tool T) is an advanced tool capable of dynamically decoding the metamodel at run-time and use model interpretation to read the model instance to generate a model object diagram.

This tool was entirely implemented in Java without EMF tool dependency. It reads the metamodels as XML and loads all definitions into memory to seek the metaclasses and their relationships that are referenced by the model instance dynamically. This was necessary to provide a tool that is completely metamodel independent, i.e., it can generate model object diagrams from any metamodel. This tool uses PlantUML (ROQUES, 2017) in the same virtual machine by sharing the run-time memory to output the object diagram into image files.

The objective of this tool is to allow developers and users to visualize the object model including their relationships and attribute values of any running MOSS/MOWS application.

B.1.4.3 Application to Model Instance

The Application to Model Instance (Tool W) depends on the Data Code to Model Instance (Tool Q). These tools are actually generated by the ECore to Data Code (Tool L). Upon execution of the MOSS/MOWS application, the data can be written as models by using methods generated by the data code generation tool.

B.1.5 Other Tools for MOWS

More tools for supporting MOWS development cycles are under construction. It is worth citing recent efforts on tools for MOWS testing. Since MOWS employ data as models and design as metamodels, it is possible to create test case generators that verify the design and compare inputs and outputs based on models. The test cases created for the experimental study presented in Chapter 7 (Section 7.3) employed an initial version of this tool, which is expected to be extended and released.

It is planned to update this list for the final version with new developments.

B.2 MOWSDL Grammar Specification

The reference for the MOWSDL grammar is also provided as part of this appendix. MOWSDL is available in XText grammar (Eclipse Foundation, 2016). It was necessary to break the grammar into two pages due to its length. These pages are shown as Figures 68 and 69, respectively.

Figure 68 – MOWSDL Grammar – Part 1

```
grammar mowsdl.MOWSDL with org.eclipse.xtext.common.Terminals
generate mOWSDL "http://mowsdl.labes.icmc.usp.br"

Declarations:
      (imports+=Import ';')*
      (
            (
                  metaDefinition = MetaDefinition
            )
            (interface= Interface)?
      )?
;

MetaDefinition:
      (Metamodel | Schema)
;

Metamodel:
      name= 'metamodel'
      rootPackage= RootPackage
;

RootPackage:
      name= ID
      namespace = Namespace
      (
            (
            '{'
                  (elements+=MetaElement (';')?)*
            '}'
            (';')?
            )
            |
            ';'
      )
;

Schema:
      'schema'
      name= ID
      namespace = Namespace
      (
            ';'
      )
;

Namespace:
      '(' name = STRING (',' schemaFile = STRING (',' metaFile = STRING)?)? ')'
;

Import:
      ('import'|'include'|'using')
      name= ID
      namespace = Namespace
;

MetaElement:
      (Package | Enum | MetaClass | MetaInterface)
;

Package:
      ('package')
      name= ID
      namespace = Namespace
      '{'
            (elements+=MetaElement (';')?)*
      '}'
;

Enum:
      ('enum')
      name= ID
      '{'
            (literals+= Literal)
            (',' literals+= Literal)*
      '}'
;

Literal:
      name= ID '(' literalName = STRING ':' literalValue = STRING ')'
;

MetaClass:
      ((abstract?='abstract')? (('metaclass') | ('class')))
      name= ID
      ('specializes' specialization+=QualifiedName (',' specialization+=QualifiedName)*)?
    ('extends' superType+=[MetaClass] (',' superType+=[MetaClass])*)?
    ('implements' superInterface+=[MetaInterface] (',' superInterface+=[MetaInterface])*)?
    '{'
      ((compositions+= Composition | references+=Reference | attributes+= Attribute) ';')*
    '}'
;

MetaInterface:
      (('metainterface') | ('interface'))
      name= ID
    ('extends' superInterface+=[MetaInterface] (',' superInterface+=[MetaInterface])*)?
    '{'
      ((compositions+= Composition | references+=Reference | attributes+= Attribute) ';')*
    '}'
;

QualifiedName: ID (('::'|'.') ID)*;
```

Source: Created by the author

Figure 69 – MOWSDL Grammar – Part 2

```
Composition:
      name=('composition'|'*') ('->')?
      element= ElementReference
      multiplicity= Multiplicity
;

Reference:
      name=('reference'|'@') ('->')?
      element= ElementReference
      multiplicity= Multiplicity
;

Attribute:
      name=('attribute'|'+') ('->')?
      element= ElementDeclaration
      multiplicity= Multiplicity
;

ElementDeclaration:
      name=ID ':'
      (
            (     (isClass?='class')
                  |
                  (isEnumeration?='enum')
                  |
                  (isType?='type')
            )?
            fullTypeName=QualifiedName
      )
;

ElementReference:
      name=ID ':'
      (
            reference=[MetaElement]
      )
;

Multiplicity:
      '[' (lowerBound=INT dots?='..')? (upperBound=INT | upperBoundStar?='*') ']'
;

Interface:
      'interface' name=ID
      namespace = Namespace
      '{'
            (('message') messages+= Message (';')?)*
            (('port') port= Port (';')?)
            (('message') messages+=Message (';')?)*
            (('service') service= Service ';')
            (('message') messages+= Message (';')?)*
      '}'
      (';')?
;

Message:
      name= ID
      '{'
            (
              ('part')? (elements+= (ElementDeclaration))?
              ((';'|',') (('part')? elements+= (ElementDeclaration))?)*
            )
      '}'
;

Port:
      name= ID
      '{'
            (operations+=Operation ';')*
      '}'
;

Operation:
            ('operation')? name=ID
                  '('
                        (
                              (
                                    ('message')? ((inputMessages+=[Message])  | (inputInnerMessages+=(Message)))
                              )
                              (','
                                    ('message')? ((inputMessages+=[Message])  | (inputInnerMessages+=(Message)))
                              )*
                        )?
                  ')'
                  (':'
                         (
                              ('message')? ((outputMessages+=[Message]) | (outputInnerMessages+=(Message)))
                         )
                         (
                              ','
                              (
                                    ('message')? ((outputMessages+=[Message]) | (outputInnerMessages+=(Message)))
                              )
                         )*
                  )?
;

Service:
      name= ID
      '('
       (
            (
                  bindingName= STRING
                  ','
                  actionName=  STRING
                  ','
            )?
            location=    STRING
       )?
      ')'
;
```

Source: Created by the author

APPENDIX

C

FEASIBILITY CASE STUDIES

C.1 Initial Remarks

We have developed Model-Oriented Web Service systems that we present as case studies within this section. We believe that they provide evidences about MOWS feasibility. At the same time, the simplest case study also serves as a concrete example of our proposals.

C.2 Simple Systems

The first case study shown in this appendix chapter is a simple Personal Information Management system named CWEB (Contact Web). CWEB contains contact list information. Therefore, there are entities, e.g., persons, and owned values associated to them, e.g., e-mail address.

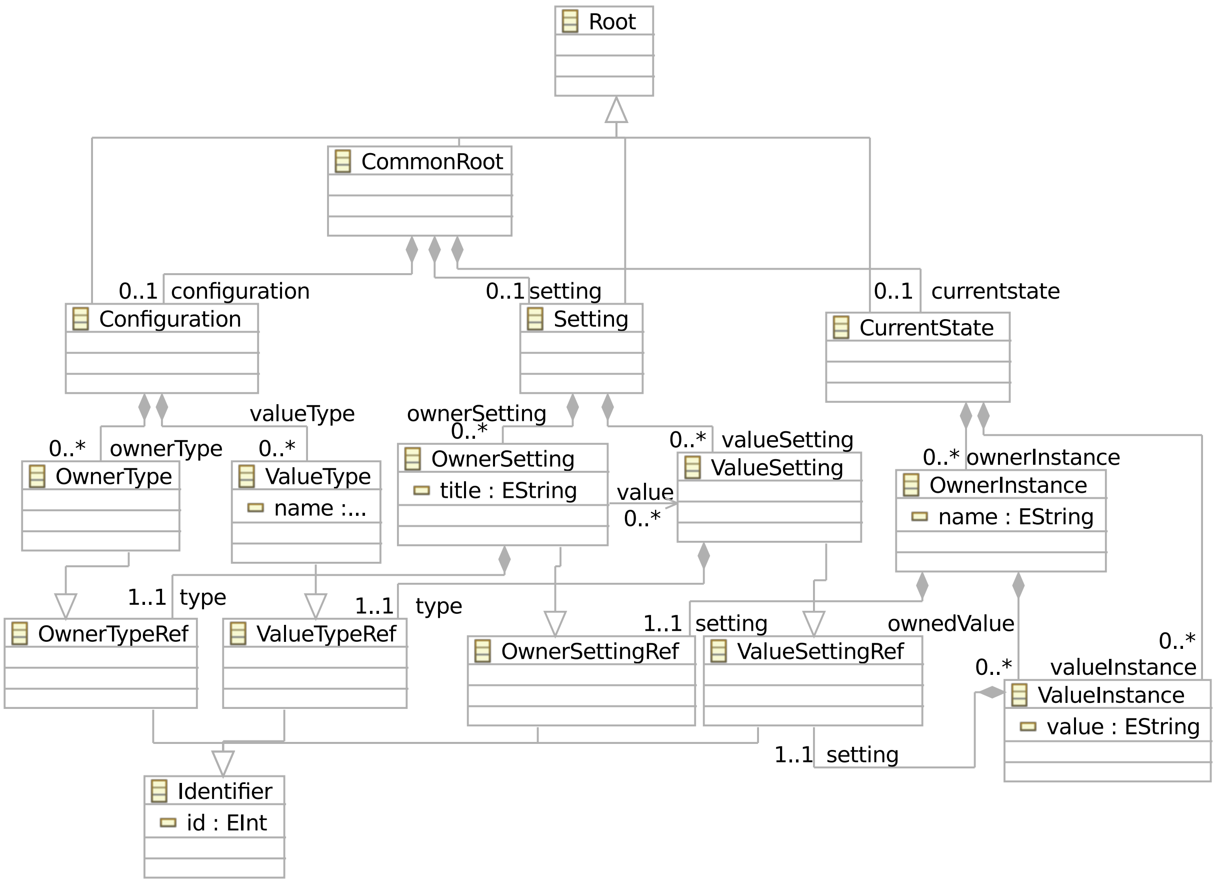
C.2.1 Extended CWEB System

This is the extended version of the example provided in Chapter 5, including a more complex metamodel to allow flexibility when defining the data stored in the server. The idea of the CWEB web service is to provide access and allow users to store and retrieve this contact list information. This case study should be simple enough to provide a clear view on how a MOWS is developed and how it behaves, providing evidences on the feasibility of the method as well as some advantages and disadvantages that should be considered upon deciding whether to use the method.

C.2.1.1 Metamodel Definition

The metamodel is the first artefact to be produced according to the established process. The metamodel for the CWEB MOWS system is shown in Figure 70. This metamodel employs

Figure 70 – Extended Metamodel for the CWEB DSL



Source: Created by the author

the three branches as root variation of CSCS (GOTTARDI; BRAGA, 2016), allowing a single branch to be transferred per WS message.

The *Configuration* metaclass aggregates the configuration elements named *OwnerType* and *ValueType*. *OwnerType* represents the type of an entity that owns values. *ValueType* represents the type of a value that contains an attribute important to the system and/or users. *ValueType* also allows to define the name of the value type. Since these metaclasses are under the configuration branch, they cannot be changed by the software during run-time, which means that the types and the value type names are fixed when the application is started. These Configuration Elements are referenced by using metaclasses with the suffix *"Ref"* (serving the role of *ConfigurationElementID*).

In turn, the *Setting* metaclass aggregates the setting elements named *OwnerSetting* and *ValueSetting*. The setting elements are used to enable referenced configuration elements. They can be also used to define specific settings that can be changed at run-time. In this example, *OwnerSetting* has the title property. Then, the title of value owners can be changed at any time by the server or by a client that has enough permission. Following the pattern, there are also *"Ref"* to allow branch cross-referencing.

For the *CurrentSetting* metaclass, there are the current setting elements named *OwnerInstance* and *ValueInstance*. These metaclasses are used to create the actual instance objects at the model level. They must reference *OwnerSetting* and *ValueSetting*, respectively, to receive the definitions specified both in setting and configuration levels. A concrete example is discussed within the following subsubsections.

C.2.1.2 Run-Time Model Instance

Considering a model instance of the PIM metamodel that is used to store persons with their names and e-mail addresses, a model is defined containing the following objects: Configuration branch: A *OwnerType* that is referenced by index 1. A *ValueType* that is referenced by index 2 and has name “e-mail”. Setting branch: A *OwnerSetting* that is referenced by index 1 and has title “Person”. A *ValueSetting* that is referenced by index 2.

Then, Current State branch would have the usual data of the PIM system: The first *OwnerInstance* has name “Alice” and contains an object *ValueInstance* which has value “alice@example.com”. The second *OwnerInstance* has name “Bob” and contains an object *ValueInstance* which has value “bob@example.com”.

Please note that the indexes do not clash even if they have the same number in this example. This is because the index is checked against each specific metaclass when referencing objects.

C.2.1.3 Model and Diagram Editor

First of all, since the MOWS transfers models serialized as XML, they also follow the XMI specification for standardization. For completeness, we also provide the full XMI/XML of the model in Figure 71. That is the actual payload in case a MOWS server transferred the complete model, which can be opened by model editors.

As part of the end-user visible characteristics of MOWS, modelling tools, including the the Eclipse Modelling Framework (EMF) (Eclipse Foundation, 2015b) originally designed for MDSE can be employed to visualize and edit models of state and data of the running system.

Models can be visualized as abstract syntax trees or as a concrete syntax representation (BRAMBILLA; CABOT; WIMMER, 2012). The abstract syntax allows to view the complete structure of the model as a tree. Concrete syntax may vary according to their objective, for instance, there could be partial graphical views as well as human readable textual representations. The described model abstract syntax tree editor is shown in Figure 72. The editor was made using the Graphical Modeling Framework for EMF (Eclipse Foundation, 2015b).

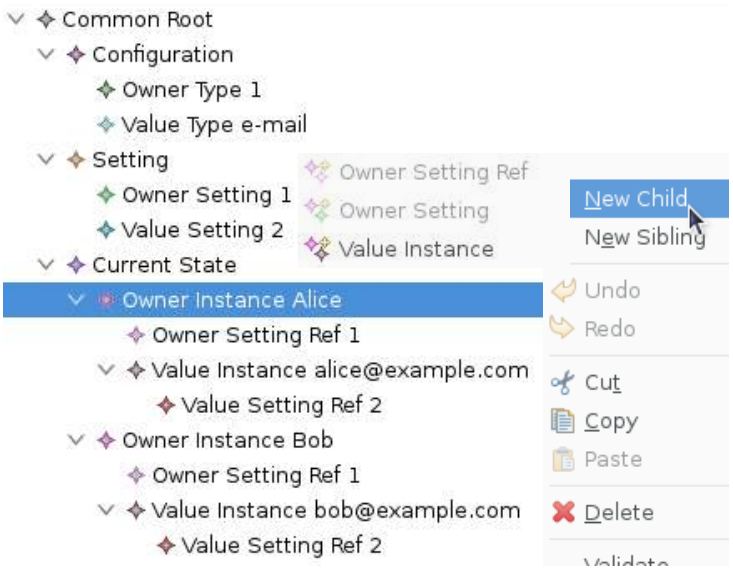
It is possible to visualize every object in the abstract syntax tree, which includes every branch, *OwnerTypes*, *ValueTypes*, *OwnerSettings*, *ValueSettings*, *OwnerInstances* and *Value-*

Figure 71 – CWEB Model as an XMI/XML

```
<?xml version=“1.0" encoding=“UTF-8"?>
<cweb:CommonRoot xmi:version=“2.0"
  xmlns:xmi=“http://www.omg.org/XMI"
  xmlns:cweb=“http://example/cweb">
 <configuration>
  <ownerType id=“1"/>
  <valueType id=“2" name=“e-mail"/>
 </configuration>
 <setting>
  <ownerSetting id=“1" title=“Person"
    value=“//@setting/@valueSetting.0"/>
  <valueSetting id=“2"/>
 </setting>
 <currentState>
  <ownerInstance name=“Alice">
   <setting id=“1"/>
   <ownedValue value=“alice@example.com">
    <setting id=“2"/>
   </ownedValue>
  </ownerInstance>
  <ownerInstance name=“Bob">
   <setting id=“1"/>
   <ownedValue value=“bob@example.com">
    <setting id=“2"/>
   </ownedValue>
  </ownerInstance>
 </currentState>
</cweb:CommonRoot>
```

Source: Created by the author

Figure 72 – CWEB Model in EMF model edit tool

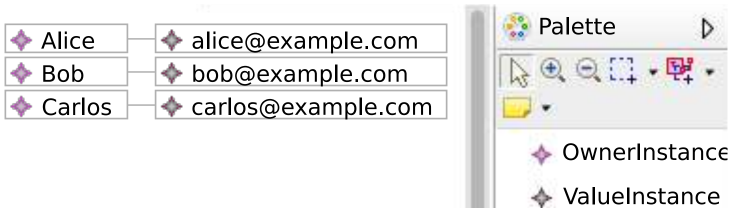


```
Common Root
  Configuration
    Owner Type 1
    Value Type e-mail
  Setting
    Owner Setting 1
    Value Setting 2
  Current State
    Owner Instance Alice
      Owner Setting Ref 1
      Value Instance alice@example.com
        Value Setting Ref 2
    Owner Instance Bob
      Owner Setting Ref 1
      Value Instance bob@example.com
        Value Setting Ref 2
```

Source: Created by the author

Instances. The editor also shows the first property of the object, the rest of the properties are accessed by a specific panel.

Figure 73 – CWEB Diagram Editor using GMF

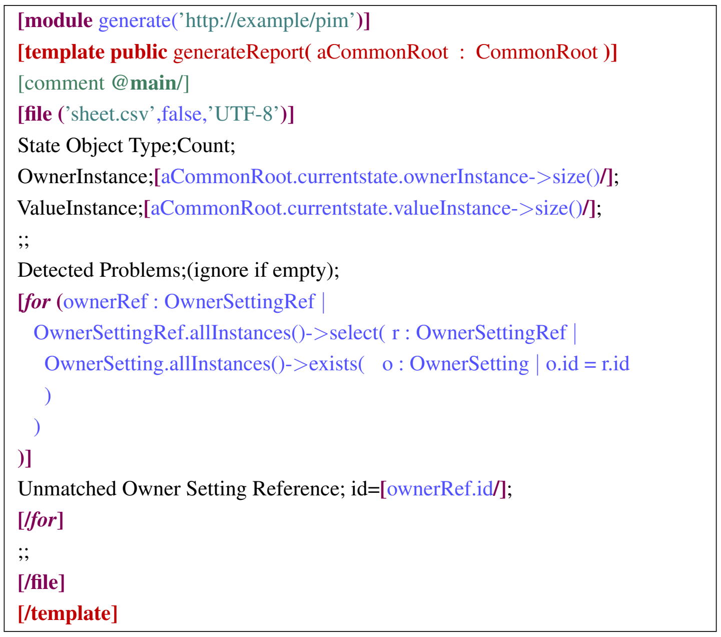


Source: Created by the author

An example of a graphical concrete syntax representation of the same model is presented in Figure 73. In this diagram editor, it is possible to view and edit the current state branch graphically. Please note that this figure contains a new entry for the *InstanceOwner* "Carlos", which was added by using the graphical editor.

C.2.1.4 Statistics Generation

Figure 74 – Code for Spreadsheet Generator



```
[module generate('http://example/pim')]
[template public generateReport( aCommonRoot : CommonRoot )]
[comment @main/]
[file ('sheet.csv',false,'UTF-8')]
State Object Type;Count;
OwnerInstance;[aCommonRoot.currentstate.ownerInstance->size()/];
ValueInstance;[aCommonRoot.currentstate.valueInstance->size()/];
;;
Detected Problems;(ignore if empty);
[for (ownerRef : OwnerSettingRef |
   OwnerSettingRef.allInstances()->select( r : OwnerSettingRef |
     OwnerSetting.allInstances()->exists(   o : OwnerSetting | o.id = r.id
     )
   )
)]
Unmatched Owner Setting Reference; id=[ownerRef.id/];
[/for]
;;
[/file]
[/template]
```

Source: Created by the author

Apart from model editors, model transformation can also be employed to extract information from the models. If the models contain data of the application configuration and state, it is possible to extract numbers from the models in order to allow statistical analysis. Code generators and code validators can also be used to validate the model. For instance, in the example present in Figure 74, there is a code of model-to-text transformation in MTL (MOFM2T) (Object Management Group, 2008b) that allows to extract a spreadsheet from the model. In

Table 67 – Generated Spreadsheet for Statistics

State Object Type	Count
OwnerInstance	3
ValueInstance	3
Detected Problems	(ignore if empty)

the spreadsheet, there are instance counts per type and it is also checked if a provided object reference is correctly matched. The results of this code generator are provided on Table 67.

C.3 Experimental Systems

This section contains the systems that were built according to MOWS requirements. They are functional and were used for the experiment described in Chapter 7 (Section 7.3.

They are instances of sales systems which follow the same requirements and present the same complexity. These systems are based on client-server architecture where the server aggregates the sales history and available products for sale. The client has freedom to create its own shopping list before submitting it to the server, which processes the transaction in case of acceptance.

C.3.1 Shop Management System

Shop Management System is a MOWS system used during the training for the experimental study presented in Chapter 7 (Section 7.3). More details on the training instrumentation are provided within Appendix E. Its design is shown in Figure 75. The marked class and relationships represent the requested maintenance carried during the experiment. The server component has a pair of methods specified as shown in Figure 76. The client design is shown in Figure 77.

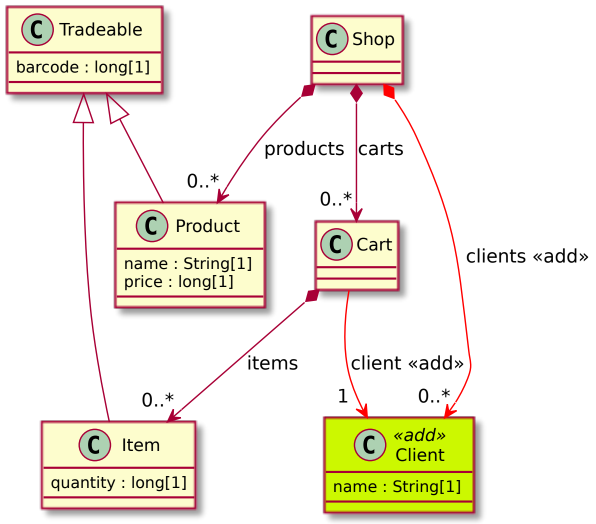
C.3.2 Clinic Management System

Clinic Management System is a MOWS system used in the experimental study presented in Chapter 7 (Section 7.3). Its design is shown in Figure 78. The marked class and relationships represent the requested maintenance carried during the experiment. This system is a derivative of the Shop Management System.

C.3.3 Delivery Tracking System

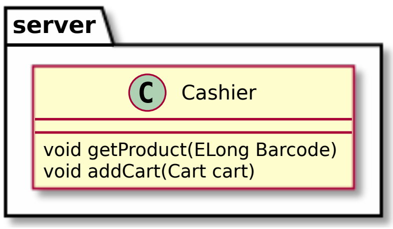
Delivery Tracking System is a MOWS system used in the experimental study presented in Chapter 7 (Section 7.3). Its design is shown in Figure 79. The marked class and relationships

Figure 75 – Class Diagram for Shop Management System



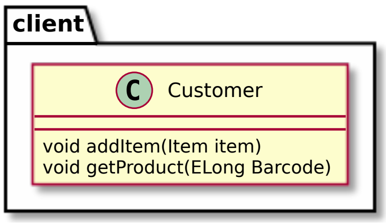
Source: Created by the author

Figure 76 – Class Diagram for Server Component



Source: Created by the author

Figure 77 – Class Diagram for Client Component



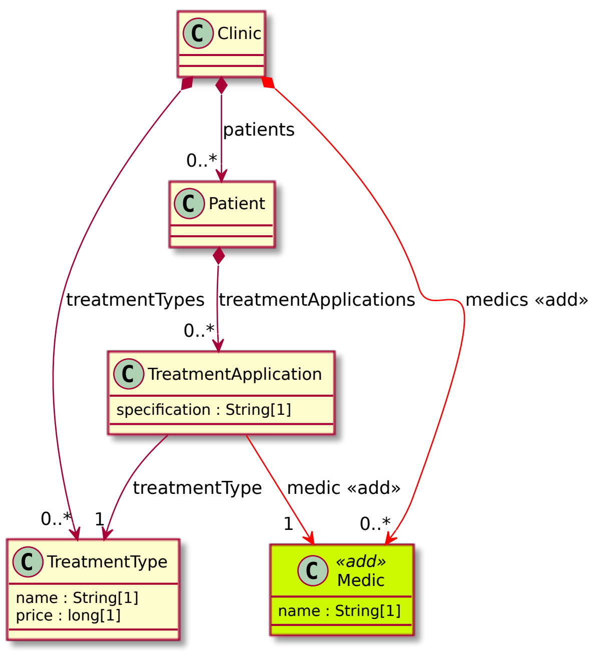
Source: Created by the author

represent the requested maintenance carried during the experiment. This system is a derivative of the Shop Management System.

C.3.4 Flight Booking System

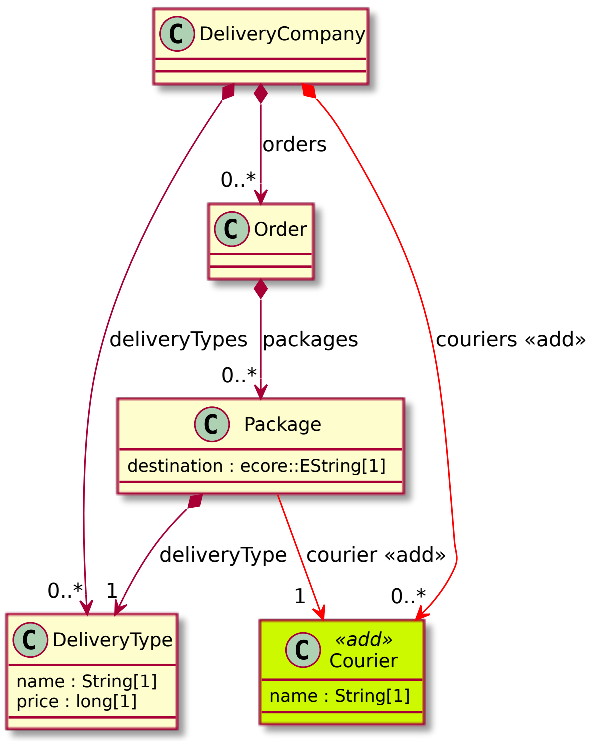
Flight Booking System is a MOWS system used in the experimental study presented in Chapter 7 (Section 7.3). Its design is shown in Figure 80. The marked class and relationships

Figure 78 – Class Diagram for Clinic Management System



Source: Created by the author

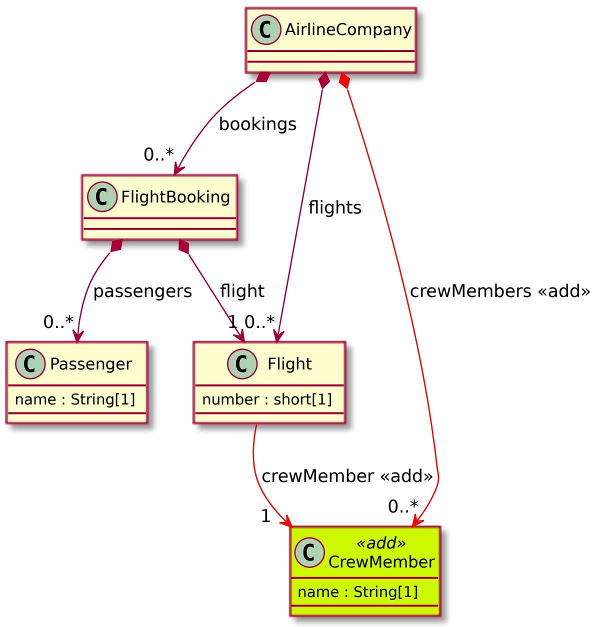
Figure 79 – Class Diagram for Flight Booking System



Source: Created by the author

represent the requested maintenance carried during the experiment. This system is a derivative of the Shop Management System.

Figure 80 – Class Diagram for Flight Booking System



Source: Created by the author

C.3.5 Restaurant Orders System

Restaurant Orders System is a MOWS system used in the experimental study presented in Chapter 7 (Section 7.3). Its design is shown in Figure 81. The marked class and relationships represent the requested maintenance carried during the experiment. This system is a derivative of the Shop Management System.

C.4 Advanced Systems

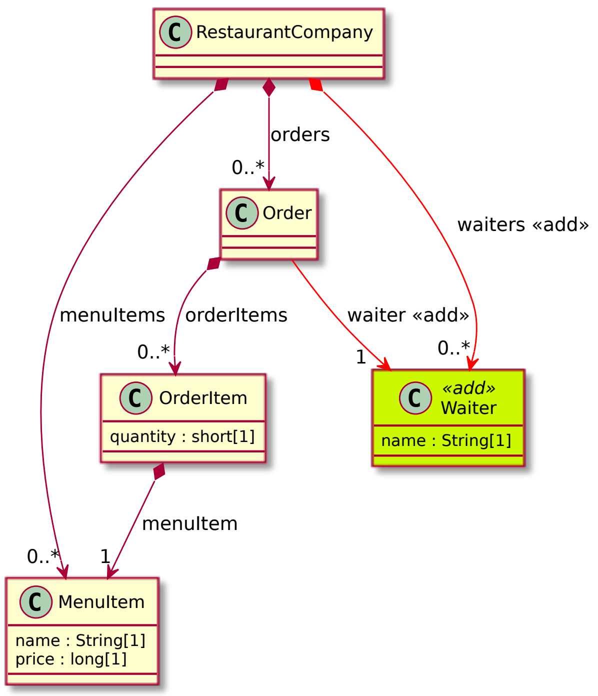
C.4.1 Related Real World Project: Software Reuse Tool

The author of this thesis is involved in a software engineering tool project which is a software itself, intended to help developers to find components to be reused. This project, albeit in early development, is targeted for production state, being used freely by real world users.

This tool is basically a smart distributed repository for reusable artifacts. It is smart because of recommendation capabilities and distributed because there can be more than one server. The combination of these servers allows the execution of cascaded queries.

In this project, the flexible metamodel pattern is not applied, because there are metamodels specifically for each concern, that is, the configuration is not performed by creating models of the same metamodel. However, the state of the server is still represented as a model. This multiple metamodel definition is also used to increase modularity. Each module only needs to decode models related to their concern, which can be smaller than a flexible metamodel.

Figure 81 – Class Diagram for Restaurant Orders System



Source: Created by the author

As part of this definition, the tool uses models to represent reusable artefact definition and indexing. Therefore, the state of the server would be a model that contains every indexed element within its database. Since this model could become too large for transmission, the indexing model allows partial loading of models while the cascaded queries allow the models to be merged to provide a single all-inclusive response.

In this manner, a request to search for a specific artifact may return a model with no index reference or a selection of indexes that match the request of the clients, that could be present in any of the connected servers requested for the cascaded query.

The clients are also allowed to send requests to change the state of the server if they have enough permission. This is done by an append request. The client sends a model that contains a new element to be appended while providing a set of credentials. The server must validate the credentials to confirm the permission of the client, then, validate if the model can be in fact appended to the state of the server.

In summary, we must highlight the following features of this project that are related to the Model-Oriented Web Service method:

- Software Engineering Application;
- Partial model transmission;
- Model merging for cascaded queries;

- Planned model processing for statistics, which are necessary for recommendation;
- High modularity design.

C.4.2 Related Real World Project: Turn Based Strategy Computer Game

A multi-player strategy computer game under development employs the Model-Oriented Web Service method while adhering completely to the CSCS pattern (GOTTARDI; BRAGA, 2016).

It is a territorial conquest turn-based game, which is also the largest case study that employs this pattern. The configuration branch is employed for game modifications, while the setting branch contains fixed situation definitions that do not change throughout the course of the turns, e.g as the maximum number of players and the fixed parts of the world in which the game is set on.

Despite the flexible configuration advantages, the most significant drawback of using flexibility adherence for this large project is that the metamodel is the most complex among the projects presented herein. The current state branch is used to define the current turn situation. The history branch variation was employed in this project to allow accumulating the previous states as well, which serve as a history for what happens inside the game situation.

According to the client-server architecture, the server hosts a game for a number of clients that represent the players. The clients send requests containing the actions that the players wish to be carried for the turn to be ended. The server processes the actions for ending the turns and responds to the clients with the new states after the turn has been changed.

The server has to validate the permission of the clients for the action requests as well as hiding the data from opponent players. This data hiding is also a specific case of partial model transmission.

Therefore, for this project, we must highlight the following Model-Oriented Web Service method related features:

- Partial model transmission;
- Call synchronisation;
- Planned model processing for statistics, which are necessary for recommendation;
- Full CSCS adherence.

C.5 Planned Systems

New proposals for MOWS systems have been suggested. The ability of quickly building systems that allows asynchronous transmission of partial models are being studied in order to develop software for education, traffic control and smart homes.

Partial model transmission allows the clients to manage a branch of the model and then re-synchronize the model at the server. This is beneficial in cases of network interruption, since the client would still keep a partial copy of the model in order to maintain its functionality without the server.

APPENDIX D

DATA STRUCTURE VERIFICATION EXPERIMENT

This section contains the verbatim copy of the tasks performed by the participants of the experimental study presented in Section 7.2 (Chapter 7). This section follows the same structure as the survey used for the study: Consent Agreement Form, Profile Characterization Form, Class Identification Task, List Identification Task, Relationship Identification Task, Diagram Matching Task, and final feedback form.

D.1 Consent Agreement Form

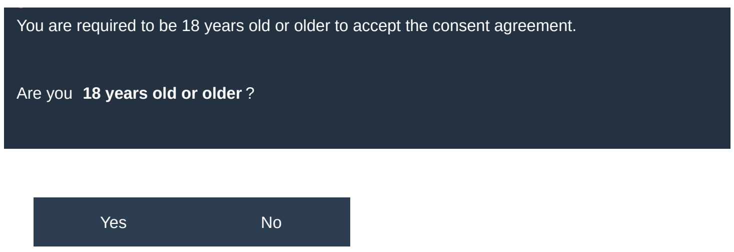
Prior to starting the experiment, the participants had to answer a consent form, which was only valid if they were 18 years old or older, as visible in Figure 82. Therefore, the form asked the participants to confirm their age and to answer the consent agreement whether they wished to participate. They also had freedom to quit at anytime, as specified in the consent agreement. This agreement was only shown to participants who answered “yes” to the age question, which implies they can only accept it if their age is 18 or older. A verbatim copy of the agreement is shown in Figure 83. Answering “no” to any of these questions would forward the participant to the last page of the survey, without effectively participating on any task.

D.2 Profile Characterization Form

The profile characterization was planned to verify the experience of participants. This profile was asked to any participant who accepted the consent agreement. The first question was related to the school attendance level of the participants, as present in Figure 84.

After answering the school attendance question, if the participants answered they were undergraduate students or completed graduation, a new question on the graduation course would

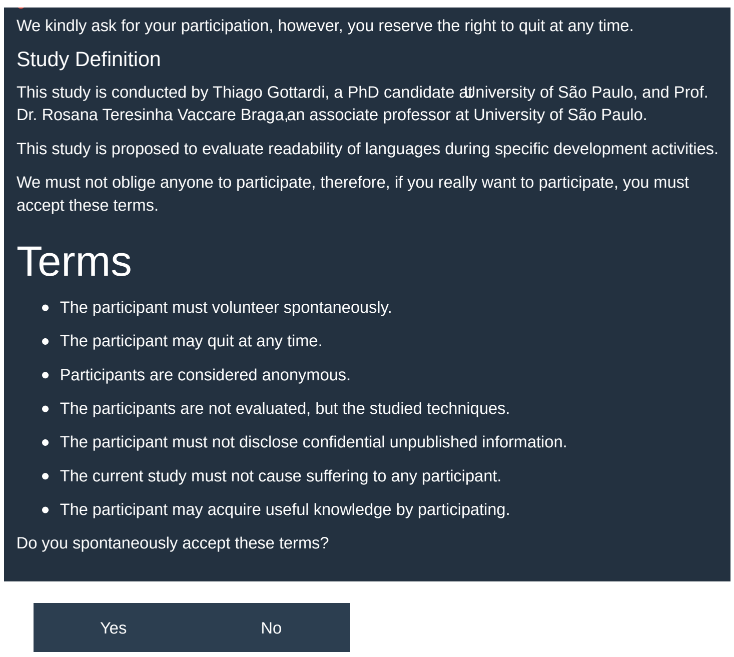
Figure 82 – Participant Age Confirmation


You are required to be 18 years old or older to accept the consent agreement.

Are you **18 years old or older** ?

Source: Created by the author

Figure 83 – Consent Agreement


We kindly ask for your participation, however, you reserve the right to quit at any time.

Study Definition

This study is conducted by Thiago Gottardi, a PhD candidate at University of São Paulo, and Prof. Dr. Rosana Teresinha Vaccare Braga,an associate professor at University of São Paulo.

This study is proposed to evaluate readability of languages during specific development activities.

We must not oblige anyone to participate, therefore, if you really want to participate, you must accept these terms.

Terms

- The participant must volunteer spontaneously.
- The participant may quit at any time.
- Participants are considered anonymous.
- The participants are not evaluated, but the studied techniques.
- The participant must not disclose confidential unpublished information.
- The current study must not cause suffering to any participant.
- The participant may acquire useful knowledge by participating.

Do you spontaneously accept these terms?

Source: Created by the author

appear, as visible in Figure 85.

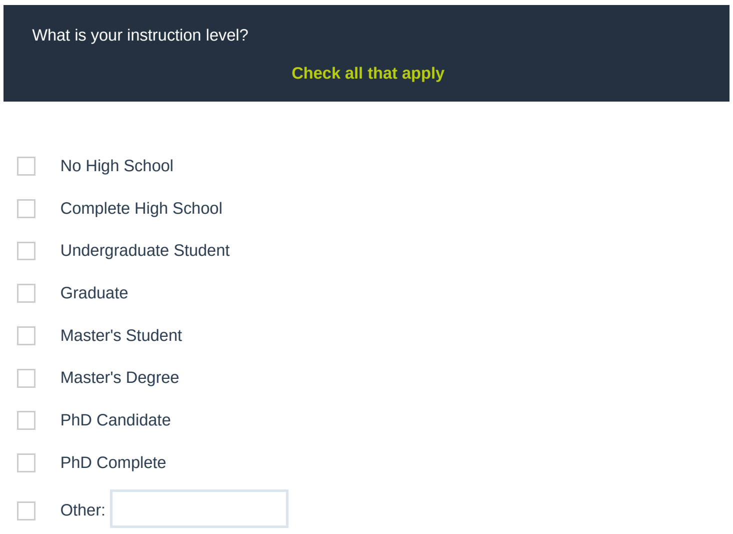
Regardless of the graduation of the participant, the form would also include a question on their occupation, as shown in Figure 86. Past occupation experiences are also asked to be included.

Afterwards, the programming experience level is asked to the participants as present in Figure 87. This question is very important to the study and has been used to select the participants, as defined in Section 7.2.2 (Chapter 7).

The experience on Java programming language is also required since it was the programming language used in the study. This question form is shown in Figure 88.

Experience on UML class and object diagrams is also asked, as visible in Figure 89, but it was not required for selection. Class diagrams are used during the diagram matching task in

Figure 84 – School Attendance Question



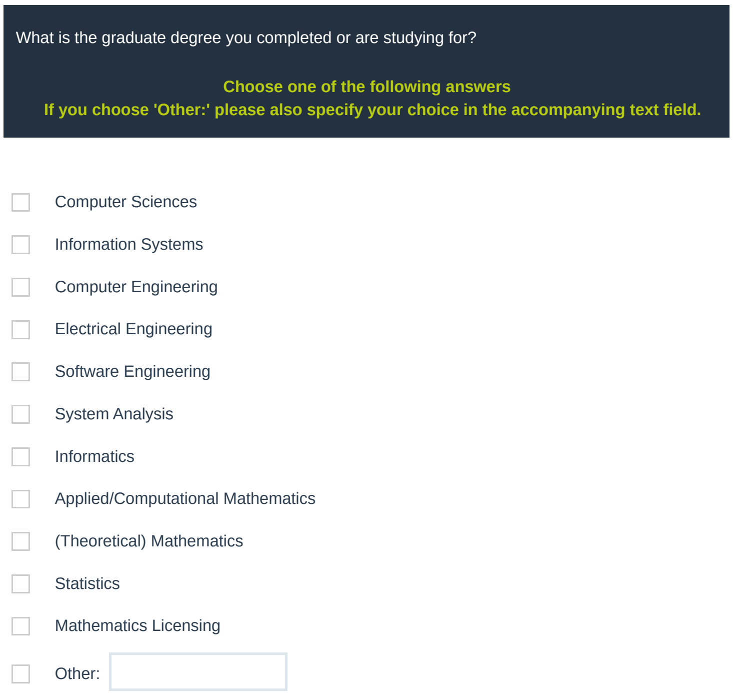
What is your instruction level?

Check all that apply

- ☐ No High School
- ☐ Complete High School
- ☐ Undergraduate Student
- ☐ Graduate
- ☐ Master's Student
- ☐ Master's Degree
- ☐ PhD Candidate
- ☐ PhD Complete
- ☐ Other:

Source: Created by the author

Figure 85 – Graduation Question



What is the graduate degree you completed or are studying for?

Choose one of the following answers
If you choose 'Other:' please also specify your choice in the accompanying text field.

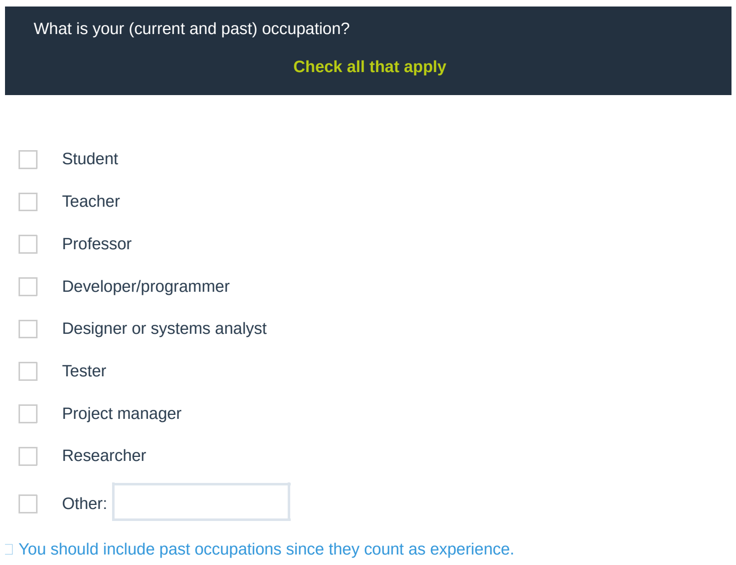
- ☐ Computer Sciences
- ☐ Information Systems
- ☐ Computer Engineering
- ☐ Electrical Engineering
- ☐ Software Engineering
- ☐ System Analysis
- ☐ Informatics
- ☐ Applied/Computational Mathematics
- ☐ (Theoretical) Mathematics
- ☐ Statistics
- ☐ Mathematics Licensing
- ☐ Other:

Source: Created by the author

the experiment, therefore, their experience could affect its results.

The source code present in the tasks throughout the study were generated using the code syntax highlight from Eclipse IDE, i.e. colors and text format. Therefore, it was also asked if the participants had experience on Eclipse IDE, which is shown in Figure 90, since it could affect their ability to read the code.

Figure 86 – Current and Past Occupation Question



What is your (current and past) occupation?

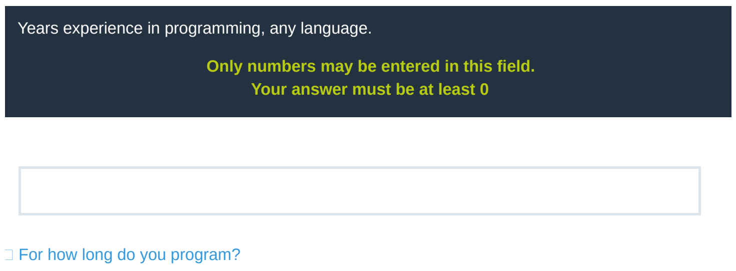
Check all that apply

- ☐ Student
- ☐ Teacher
- ☐ Professor
- ☐ Developer/programmer
- ☐ Designer or systems analyst
- ☐ Tester
- ☐ Project manager
- ☐ Researcher
- ☐ Other:

You should include past occupations since they count as experience.

Source: Created by the author

Figure 87 – Programming Experience Question



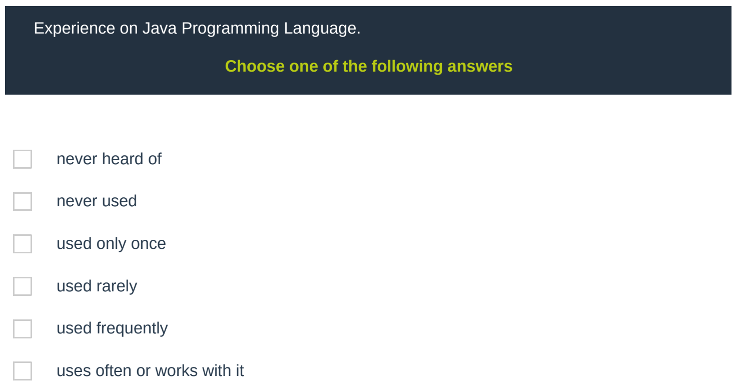
Years experience in programming, any language.

Only numbers may be entered in this field.
Your answer must be at least 0

For how long do you program?

Source: Created by the author

Figure 88 – Java Experience Question



Experience on Java Programming Language.

Choose one of the following answers

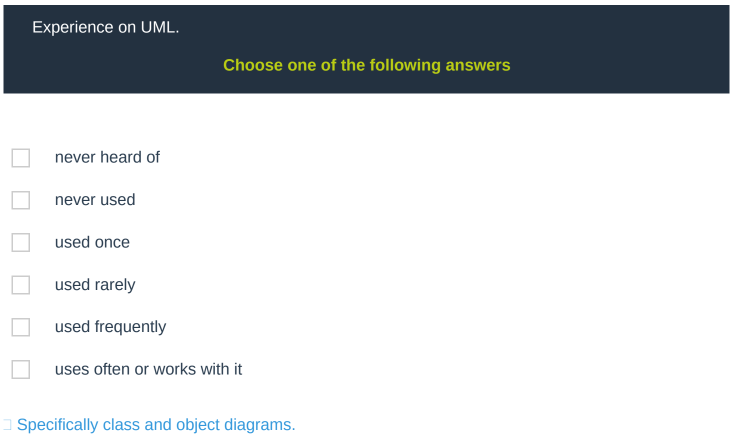
- ☐ never heard of
- ☐ never used
- ☐ used only once
- ☐ used rarely
- ☐ used frequently
- ☐ uses often or works with it

Source: Created by the author

AS the MOWSDL language was created for metamodeling, a question on metamodeling experience was also included in the characterization form, as shown in Figure 91.

There was a concern that ontology experience would also affect the metamodeling tasks,

Figure 89 – UML Experience Question



Experience on UML.

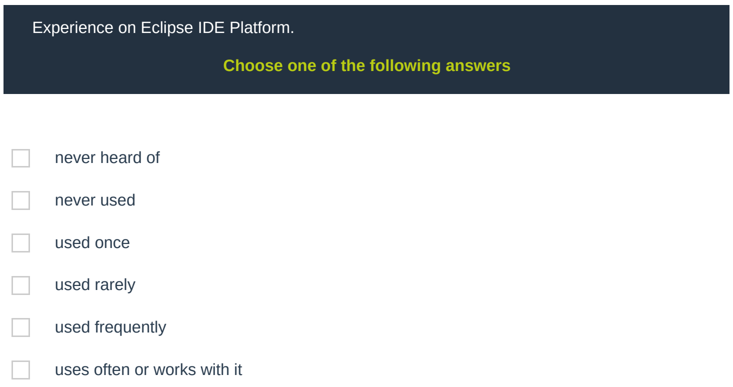
Choose one of the following answers

- ☐ never heard of
- ☐ never used
- ☐ used once
- ☐ used rarely
- ☐ used frequently
- ☐ uses often or works with it

Specifically class and object diagrams.

Source: Created by the author

Figure 90 – Eclipse IDE Experience Question



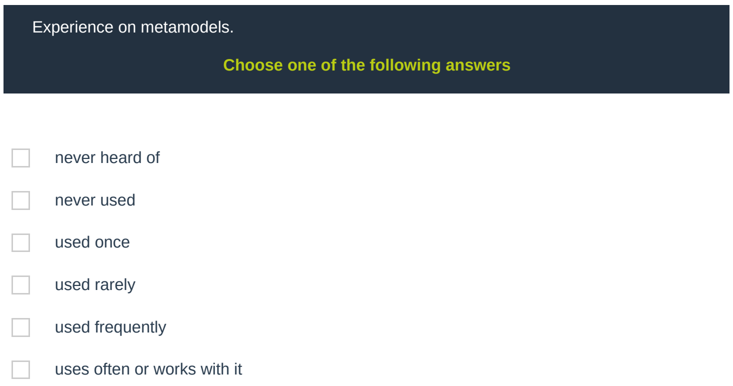
Experience on Eclipse IDE Platform.

Choose one of the following answers

- ☐ never heard of
- ☐ never used
- ☐ used once
- ☐ used rarely
- ☐ used frequently
- ☐ uses often or works with it

Source: Created by the author

Figure 91 – Metamodeling Experience Question



Experience on metamodels.

Choose one of the following answers

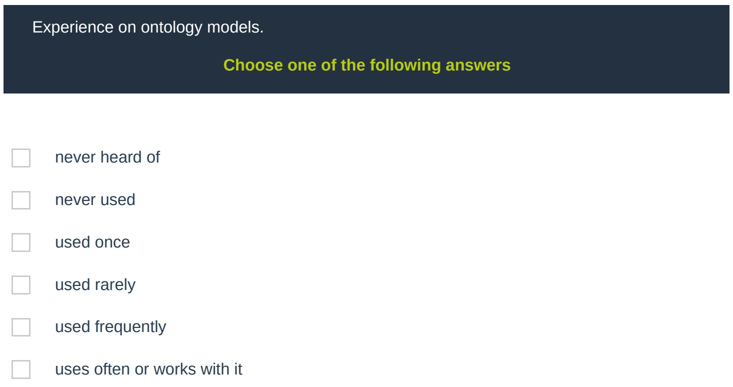
- ☐ never heard of
- ☐ never used
- ☐ used once
- ☐ used rarely
- ☐ used frequently
- ☐ uses often or works with it

Source: Created by the author

due to similarity on concept modeling. Therefore, question on metamodeling experience was also added, as visible in Figure 92.

Another concern that was considered during study planning was that experience on code

Figure 92 – Ontology Experience Question


Experience on ontology models.

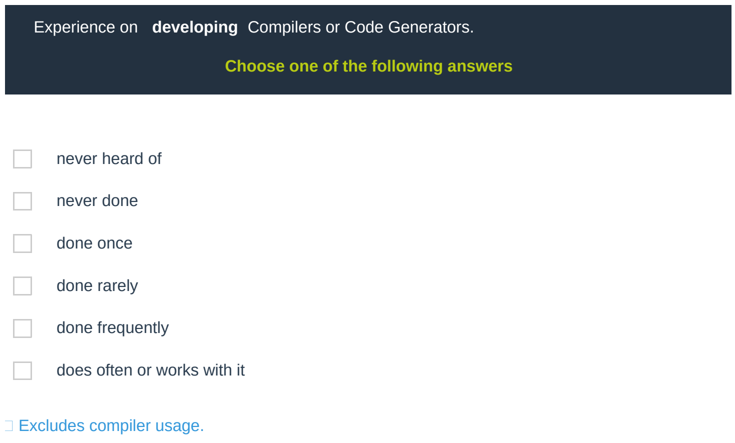
Choose one of the following answers

- never heard of
- never used
- used once
- used rarely
- used frequently
- uses often or works with it

Source: Created by the author

generation and compilers would also affect the usage of MOWSDL as it could be linked to language creation tasks that are present in projects that involve code generation or compilers. For this reason, this question on experience was added as shown in Figure 93.

Figure 93 – Experience on Developing Code Generation and Compilers Question


Experience on **developing** Compilers or Code Generators.

Choose one of the following answers

- never heard of
- never done
- done once
- done rarely
- done frequently
- does often or works with it

Excludes compiler usage.

Source: Created by the author

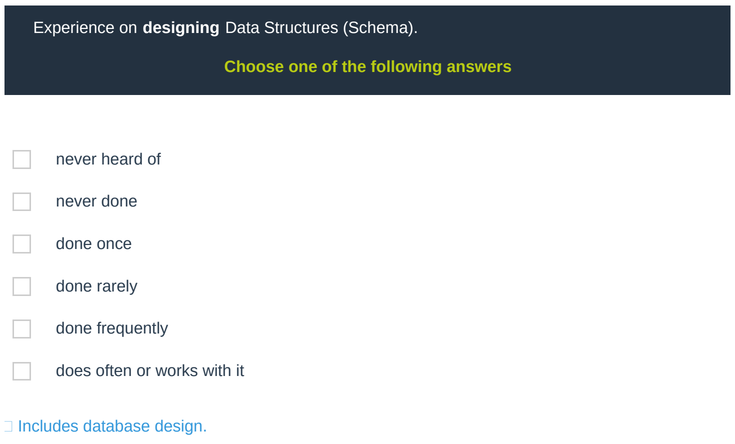
The usage of Java and MOWSDL to declare data structures could also be related to the efforts of design database schemas. Therefore, a question on database design experience was added, as visible in Figure 94.

D.3 Class Identification

Class identification is presented as the first task of the study. Prior to this task, a specific training session is presented, as described in Section 7.2.3 (Chapter 7).

This task is divided into subtasks. Each subtask asks the participant to count classes using either MOWSDL or Java code. The order of subtasks is completely random, therefore, the participants only know that there are 4 Java subtasks and 4 MOWSDL subtasks, i.e. 8 subtasks.

Figure 94 – Database Design Experience Question


Experience on **designing** Data Structures (Schema).

Choose one of the following answers

- ☐ never heard of
- ☐ never done
- ☐ done once
- ☐ done rarely
- ☐ done frequently
- ☐ does often or works with it

Includes database design.

Source: Created by the author

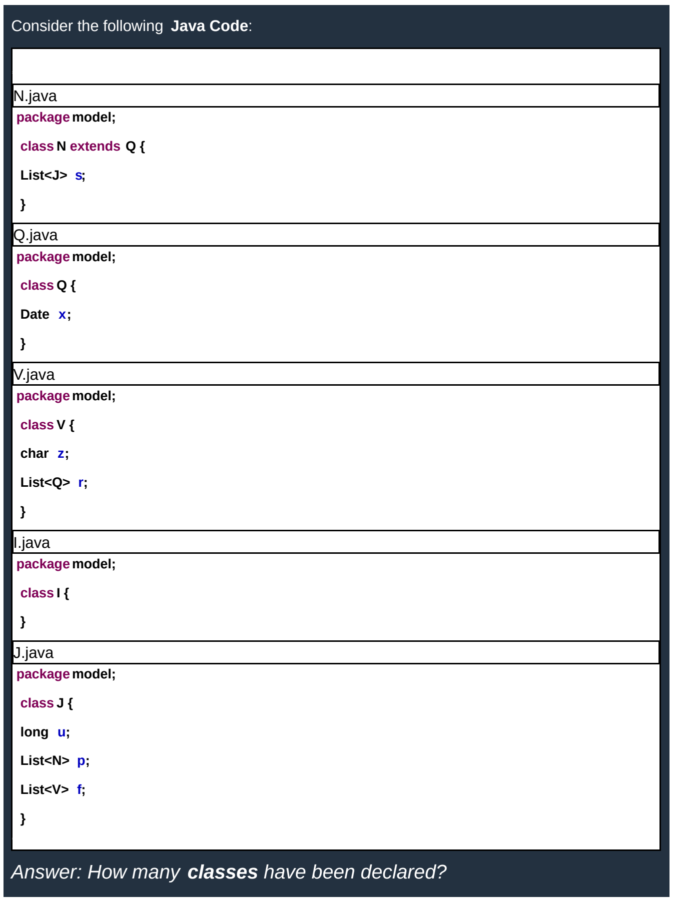
Each one of the 8 subtasks may appear in any order, therefore, it is not possible to know which language is going to be used before the subtask appears. The subtasks were also paired but the declaration order of classes was changed. Participants who completed the experiment said they were not able to identify that there were paired subtasks. The language and valid answers for the subtasks are represented on Table 68. This table is composed by four columns: The first column represents the paired number of the subtask, i.e. there are two subtasks with each number; The second column indicates which is the language used for the subtask; The third column indicates the valid answers that the participants could enter, they could be null for "no answer" or a decimal number from 0 to 10; The fourth column indicates the expected answer for this subtask to be considered correct, i.e., it must be equal to the correct answer. Null answers were also considered incorrect.

Table 68 – Class Identification Subtasks and Answers

Subtask Number	Language	Valid Answers	Correct Answer
1	Java	Null or 0 to 10	5
1	MOWSDL	Null or 0 to 10	5
2	Java	Null or 0 to 10	6
2	MOWSDL	Null or 0 to 10	6
3	Java	Null or 0 to 10	9
3	MOWSDL	Null or 0 to 10	9
4	Java	Null or 0 to 10	8
4	MOWSDL	Null or 0 to 10	8

The questions showing the code for the participants to identify the number of lists are visible in Figures 95 to 102. As the questions from the first three tasks were answered by counting keywords on text, a detail that is hidden inside the questions is a random set of keywords, e.g. "class". These keywords were hidden inside these questions to incapacitate the participants from using automatic tools (e.g., web browser search function) from counting these keywords.

Figure 95 – Class Counting Subtask 1 Java



Consider the following **Java Code**:

N.java
```
package model;
 class N extends Q {
 List<J>  s;
 }
```
Q.java
```
package model;
 class Q {
 Date  x;
 }
```
V.java
```
package model;
 class V {
 char  z;
 List<Q>  r;
 }
```
I.java
```
package model;
 class I {
 }
```
J.java
```
package model;
 class J {
 long  u;
 List<N>  p;
 List<V>  f;
 }
```
*Answer: How many **classes** have been declared?*

Source: Created by the author

It is important to remind that after this task was finished, the participants had access to an opinion form on which language was preferred for the given task, as described in Section 7.2.3 (Chapter 7).

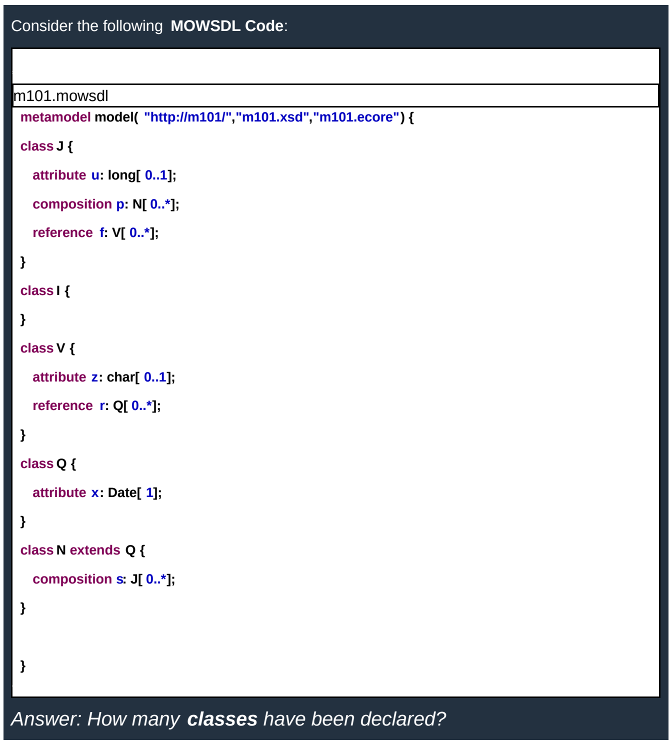
D.4 List Identification

The list identification task follows the same planning as the class identification task. The major difference is the task type, which is focused on counting lists instead of classes. The answers table, presented on Table 69, follows the same structure as the previous table. It is important to remind that prior to the execution of this task, there is a training session as described in Section 7.2.3 (Chapter 7).

The questions showing the code for the participants to identify the number of lists are visible in Figures 103 to 110.

It is important to remind that after this task was finished, the participants had access to an opinion form on which language was preferred for the given task, as described in Section 7.2.3 (Chapter 7).

Figure 96 – Class Counting Subtask 1 MOWSDL


Consider the following **MOWSDL Code**:

m101.mowsdl

```
metamodel model( "http://m101/","m101.xsd","m101.ecore") {
class J {
  attribute u: long[ 0..1];
  composition p: N[ 0..*];
  reference f: V[ 0..*];
}
class I {
}
class V {
  attribute z: char[ 0..1];
  reference r: Q[ 0..*];
}
class Q {
  attribute x: Date[ 1];
}
class N extends Q {
  composition s: J[ 0..*];
}


}
```

*Answer: How many **classes** have been declared?*

Source: Created by the author

Table 69 – List Identification Subtasks and Answers

Subtask Number	Language	Valid Answers	Correct Answer
1	Java	Null or 0 to 10	6
1	MOWSDL	Null or 0 to 10	6
2	Java	Null or 0 to 10	5
2	MOWSDL	Null or 0 to 10	5
3	Java	Null or 0 to 10	3
3	MOWSDL	Null or 0 to 10	3
4	Java	Null or 0 to 10	4
4	MOWSDL	Null or 0 to 10	4

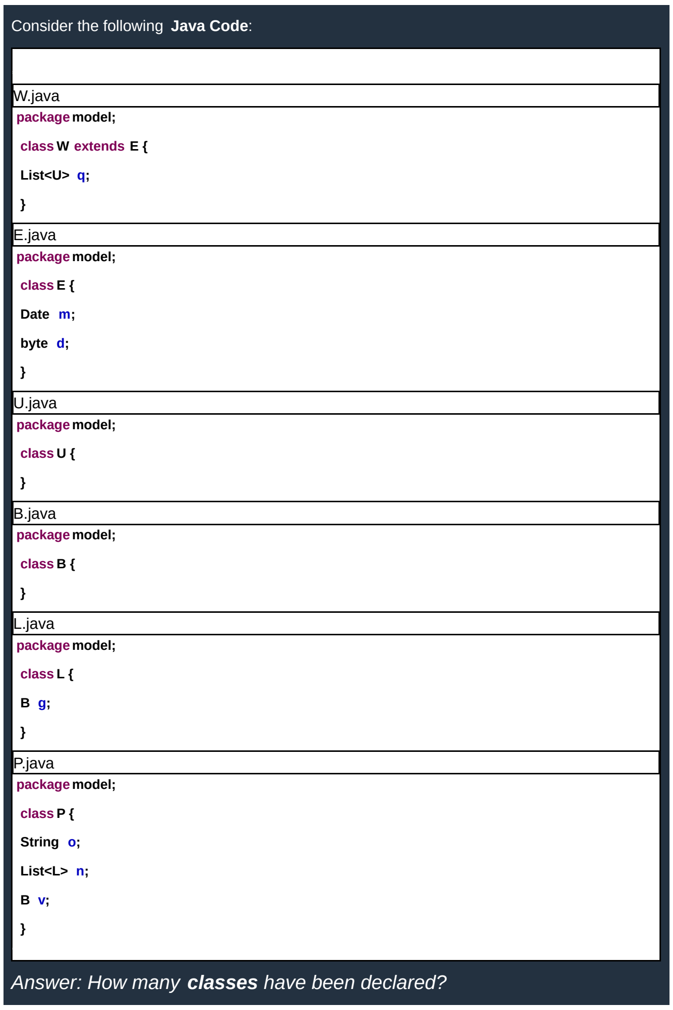
D.5 Relationship Identification

The relationship identification task follows the same planning as the class and list identification task. The major difference is the task type, which is focused on counting lists instead of lists. The answers table, presented on Table 70, follows the same structure as the previous table. It is important to remind that prior to the execution of this task, there is a training session as described in Section 7.2.3 (Chapter 7).

The questions showing the code for the participants to identify the number of relationships are visible in Figures 111 to 118.

It is important to remind that after this task was finished, the participants had access to an

Figure 97 – Class Counting Subtask 2 Java


Consider the following **Java Code**:

W.java
```
package model;
 class W extends E {
 List<U> q;
 }
```
E.java
```
package model;
 class E {
 Date m;
 byte d;
 }
```
U.java
```
package model;
 class U {
 }
```
B.java
```
package model;
 class B {
 }
```
L.java
```
package model;
 class L {
 B g;
 }
```
P.java
```
package model;
 class P {
 String o;
 List<L> n;
 B v;
 }
```
*Answer: How many **classes** have been declared?*

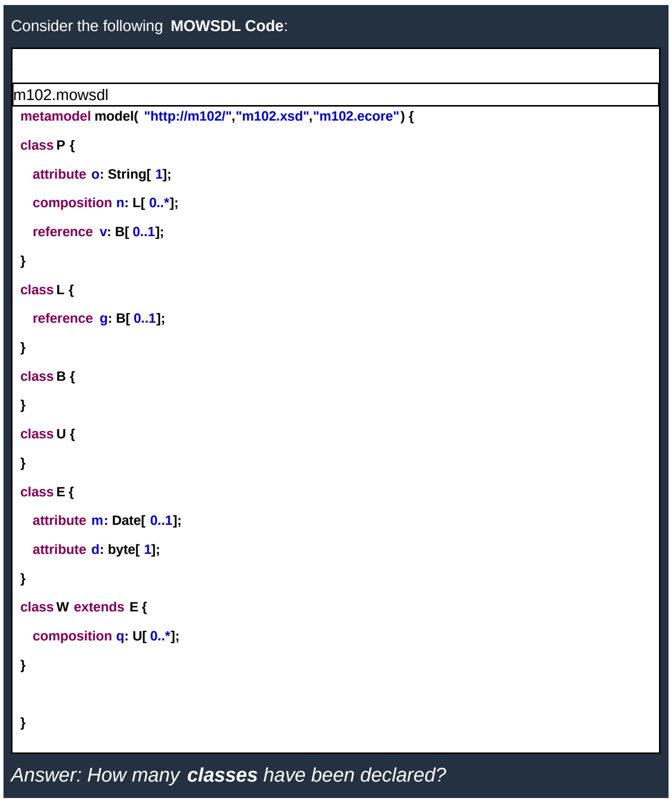
Source: Created by the author

Table 70 – Relationship Identification Subtasks and Answers

Subtask Number	Language	Valid Answers	Correct Answer
1	Java	Null or 0 to 10	5
1	MOWSDL	Null or 0 to 10	5
2	Java	Null or 0 to 10	6
2	MOWSDL	Null or 0 to 10	6
3	Java	Null or 0 to 10	4
3	MOWSDL	Null or 0 to 10	4
4	Java	Null or 0 to 10	3
4	MOWSDL	Null or 0 to 10	3

opinion form on which language was preferred for the given task, as described in Section 7.2.3 (Chapter 7).

Figure 98 – Class Counting Subtask 2 MOWSDL


Consider the following **MOWSDL Code**:

m102.mowsdl

```
metamodel model( "http://m102/","m102.xsd","m102.ecore") {
class P {
   attribute o: String[ 1];
   composition n: L[ 0..*];
   reference v: B[ 0..1];
}
class L {
   reference g: B[ 0..1];
}
class B {
}
class U {
}
class E {
   attribute m: Date[ 0..1];
   attribute d: byte[ 1];
}
class W extends E {
   composition q: U[ 0..*];
}


}
```

*Answer: How many **classes** have been declared?*

Source: Created by the author

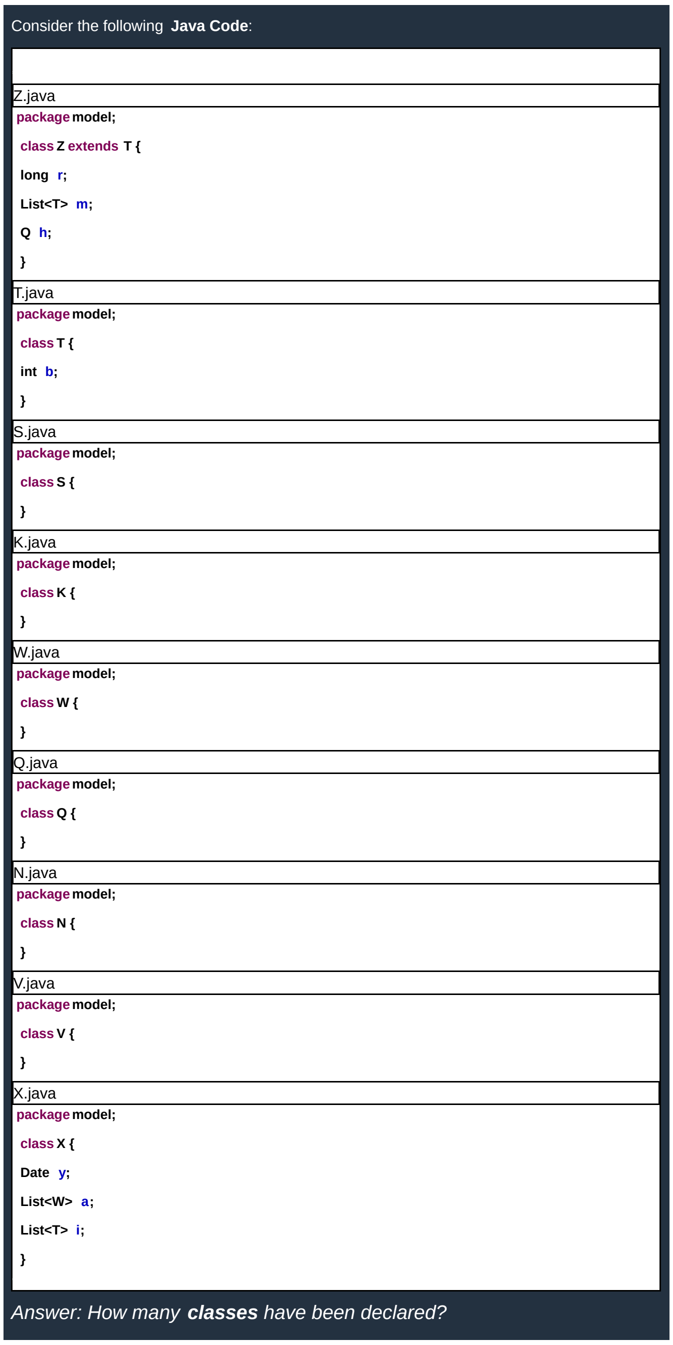
D.6 Diagram matching

Diagram matching task is also composed by 4 pairs of subtasks, i.e., 4 Java subtasks and 4 MOWSDL subtasks. Instead of accepting a 0 to 10 decimal number, this task was composed by a three diagrams per question. Only one diagram matched the presented code. The order of subtasks and the order of diagrams presented to the participant are completely random.

There are eight possible diagrams. Their names are based on their numeric seed used to generate random diagrams. Diagram A17 is shown in Figure 119; Diagram A26 is shown in Figure 120; Diagram A20 is shown in Figure 121; Diagram A34 is shown in Figure 122; Diagram A23 is shown in Figure 123; Diagram A47 is shown in Figure 124; Diagram A4 is shown in Figure 125; Diagram A29 is shown in Figure 126.

The valid answers are presented on Table 71. This table follows the same structure of previous answers tables. While the correct answers shown on table are always the first option, it is important to remind that the answer options are shown as diagrams in random order to the participants, i.e., the correct answer position varies. Also, the names of diagrams was not visible to the participants. The MOWSDL version indeed did show the seed number as part of the metamodel name, but it was not possible to match it to any other information. It is important to remind that prior to the execution of this task, there is a training session as described in

Figure 99 – Class Counting Subtask 3 Java


Consider the following **Java Code**:

Z.java
```
package model;
 class Z extends T {
 long r;
 List<T> m;
 Q h;
 }
```
T.java
```
package model;
 class T {
 int b;
 }
```
S.java
```
package model;
 class S {
 }
```
K.java
```
package model;
 class K {
 }
```
W.java
```
package model;
 class W {
 }
```
Q.java
```
package model;
 class Q {
 }
```
N.java
```
package model;
 class N {
 }
```
V.java
```
package model;
 class V {
 }
```
X.java
```
package model;
 class X {
 Date y;
 List<W> a;
 List<T> i;
 }
```
*Answer: How many **classes** have been declared?*

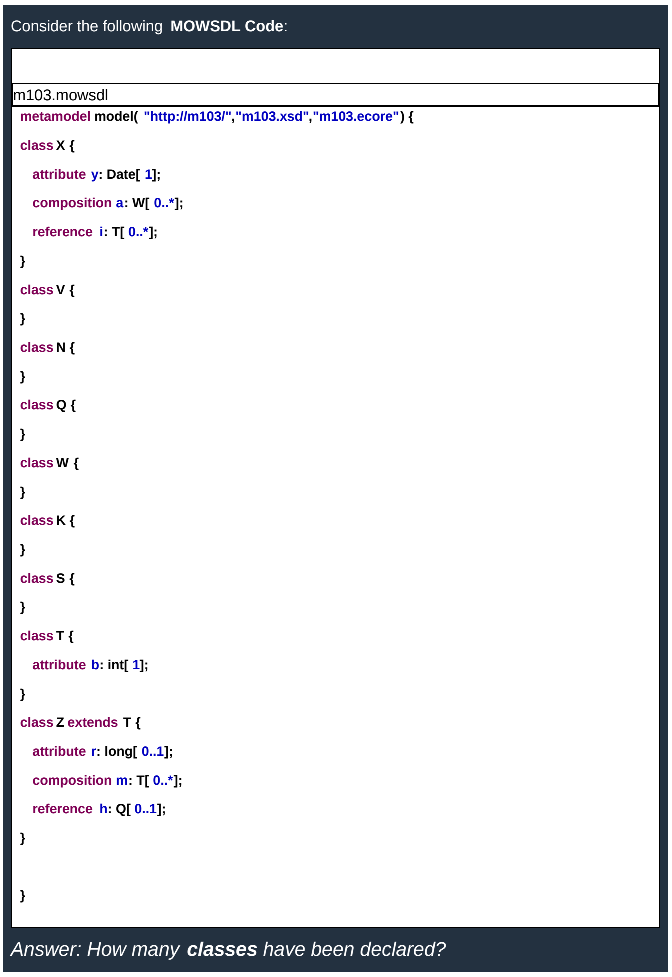
Source: Created by the author

Section 7.2.3 (Chapter 7).

The questions showing the code for the participants to find the matching diagram are visible in Figures 127 to 134.

It is important to remind that after this task was finished, the participants had access to an opinion form on which language was preferred for the given task, as described in Section 7.2.3 (Chapter 7).

Figure 100 – Class Counting Subtask 3 MOWSDL


Consider the following **MOWSDL Code**:

m103.mowsdl

```
metamodel model( "http://m103/","m103.xsd","m103.ecore") {
class X {
  attribute y: Date[ 1];
  composition a: W[ 0..*];
  reference  i: T[ 0..*];
}
class V {
}
class N {
}
class Q {
}
class W {
}
class K {
}
class S {
}
class T {
  attribute b: int[ 1];
}
class Z extends  T {
  attribute r: long[ 0..1];
  composition m: T[ 0..*];
  reference  h: Q[ 0..1];
}

}
```

Answer: How many ***classes*** *have been declared?*

Source: Created by the author

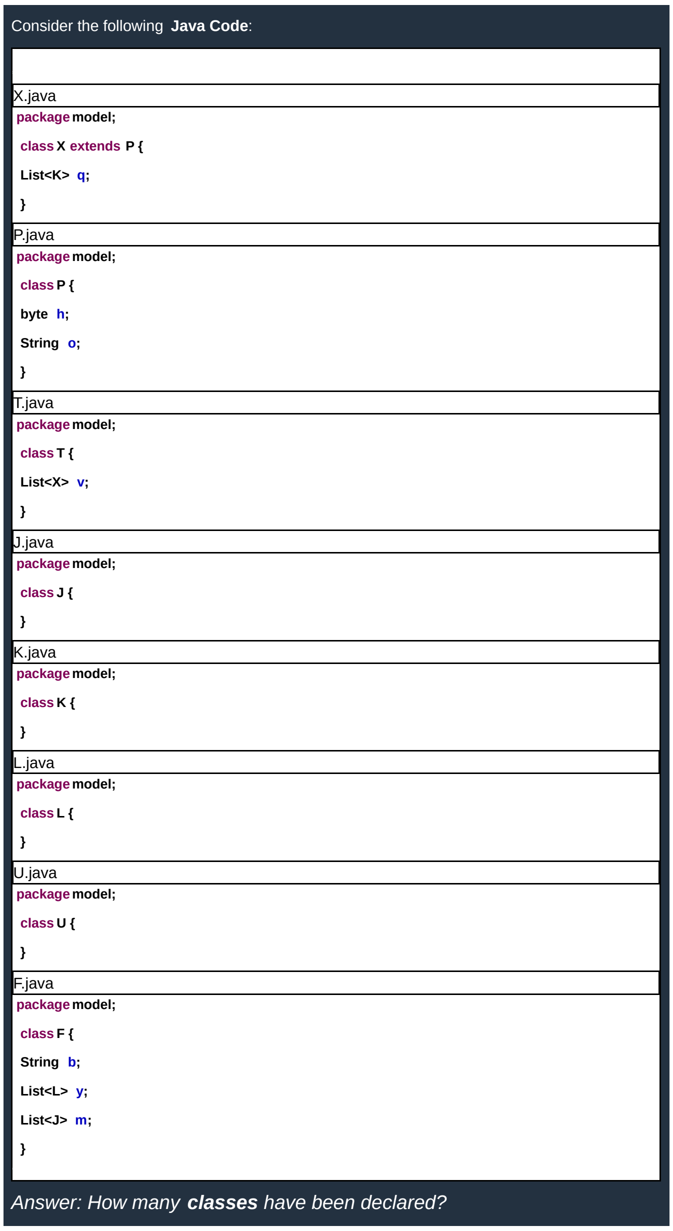
Table 71 – Diagram Matching Subtasks and Answers

Subtask Number	Language	Valid Answers	Correct Answer
1	Java	Null, A17, A26 or A20	A17
1	MOWSDL	Null, A17, A26 or A20	A17
2	Java	Null, A26, A34 or A23	A26
2	MOWSDL	Null, A26, A34 or A23	A26
3	Java	Null, A34, A4 or A47	A34
3	MOWSDL	Null, A34, A4 or A47	A34
4	Java	Null, A4, A17 or A29	A4
4	MOWSDL	Null, A4, A17 or A29	A4

D.7 Feedback Form

After the task was finished, the participants had access to a feedback form. This page is completely optional. The first feedback question was a range of -2,-1,0,+1,+2 answers labelled as “Not at All”, “No”, “So-So”, “Yes”, “A lot”, respectively, for several attributes, which are

Figure 101 – Class Counting Subtask 4 Java


Consider the following **Java Code**:

X.java
```
package model;
class X extends P {
List<K> q;
}
```
P.java
```
package model;
class P {
byte h;
String o;
}
```
T.java
```
package model;
class T {
List<X> v;
}
```
J.java
```
package model;
class J {
}
```
K.java
```
package model;
class K {
}
```
L.java
```
package model;
class L {
}
```
U.java
```
package model;
class U {
}
```
F.java
```
package model;
class F {
String b;
List<L> y;
List<J> m;
}
```
*Answer: How many **classes** have been declared?*

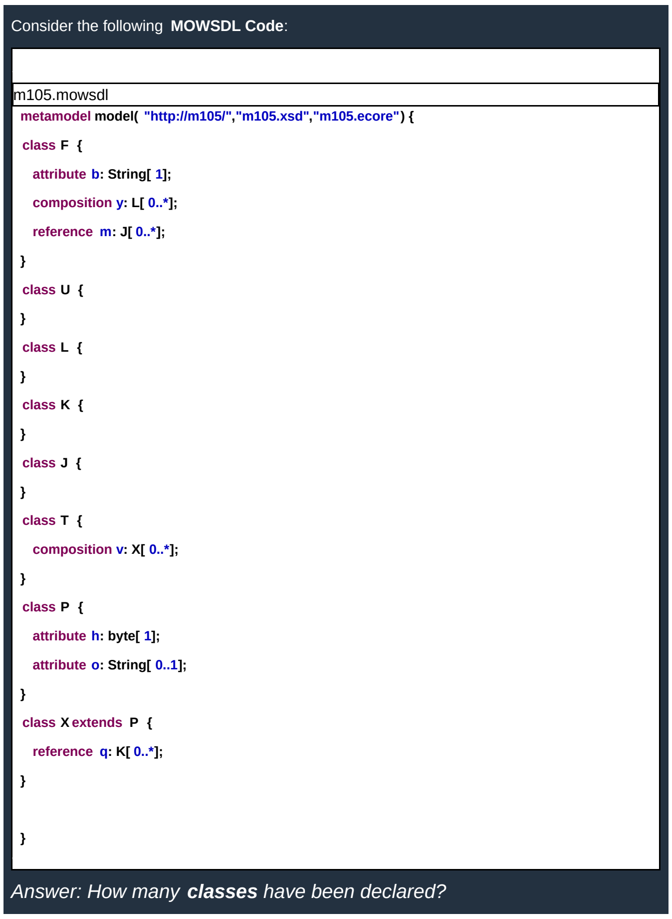
Source: Created by the author

presented on Table 72. This table is composed by four columns: The first column represents the number of the attribute; The second column indicates the attribute text; The third column indicates the valid answers that the participants could enter, they could be null for “no answer” or an integer number from -2 to +2; The fourth column indicates the expected answer for this subtask to be considered correct, i.e., any answer is valid since this is an opinion form.

It is important to clarify that any answer is considered correct since this is an opinion form. The addition of contradictory attributes was planned to cross-examine the answers for contradictions.

This form also included free text feedback and allowed the participants to provide contact

Figure 102 – Class Counting Subtask 4 MOWSDL



Consider the following **MOWSDL Code**:

m105.mowsdl

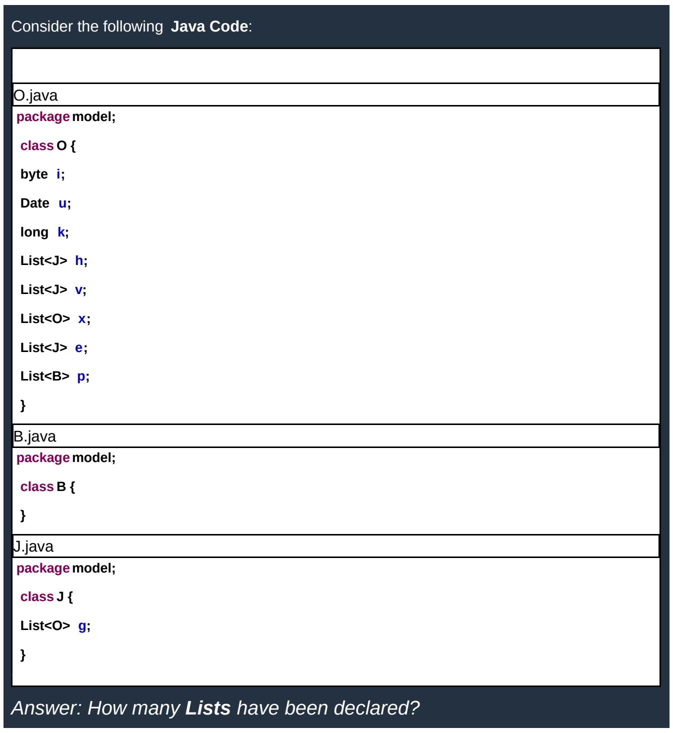
```
metamodel model( "http://m105/","m105.xsd","m105.ecore") {
 class F  {
   attribute b: String[ 1];
   composition y: L[ 0..*];
   reference  m: J[ 0..*];
}
 class U  {
}
 class L  {
}
 class K  {
}
 class J  {
}
 class T  {
   composition v: X[ 0..*];
}
 class P  {
   attribute h: byte[ 1];
   attribute o: String[ 0..1];
}
 class X extends  P  {
   reference  q: K[ 0..*];
}


}
```

*Answer: How many **classes** have been declared?*

Source: Created by the author

if they really wished to. Due to possible identifying information in this section of the form, this information will not be published since we intended to keep the participants anonymous.

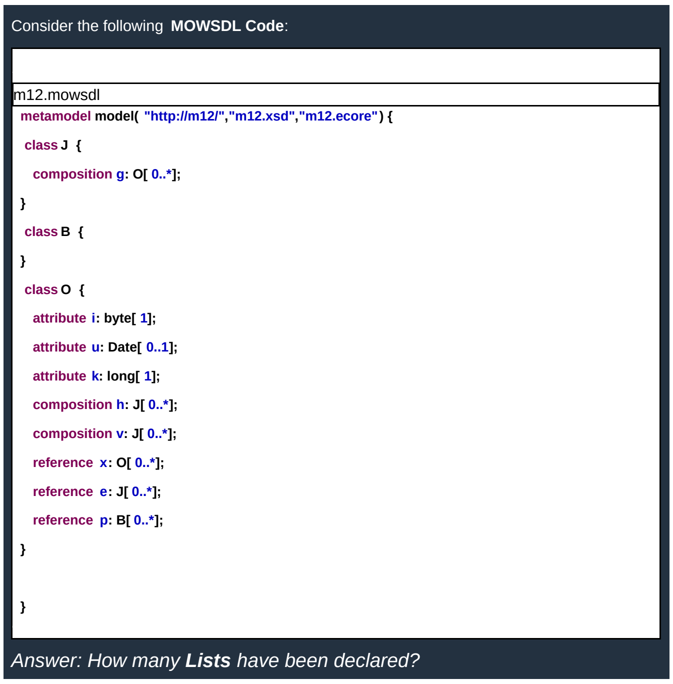
Figure 103 – List Counting Subtask 1 Java


Consider the following **Java Code**:

O.java
```
package model;
 class O {
 byte  i;
 Date  u;
 long  k;
 List<J>  h;
 List<J>  v;
 List<O>  x;
 List<J>  e;
 List<B>  p;
 }
```
B.java
```
package model;
 class B {
 }
```
J.java
```
package model;
 class J {
 List<O>  g;
 }
```
*Answer: How many **Lists** have been declared?*

Source: Created by the author

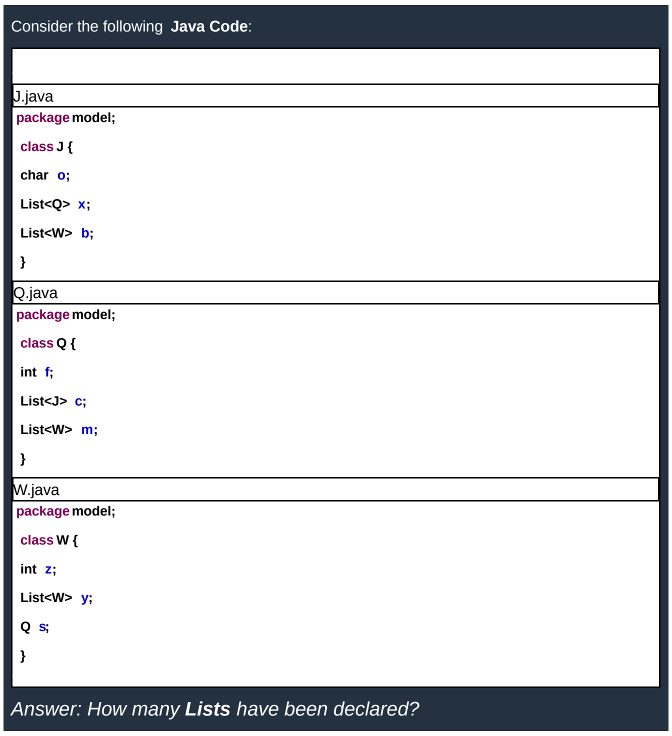
Figure 104 – List Counting Subtask 1 MOWSDL


Consider the following **MOWSDL Code**:

m12.mowsdl
```
 metamodel model( "http://m12/","m12.xsd","m12.ecore") {
  class J  {
    composition g: O[ 0..*];
 }
  class B  {
 }
  class O  {
    attribute  i: byte[ 1];
    attribute  u: Date[ 0..1];
    attribute  k: long[ 1];
    composition h: J[ 0..*];
    composition v: J[ 0..*];
    reference  x: O[ 0..*];
    reference  e: J[ 0..*];
    reference  p: B[ 0..*];
 }


 }
```
*Answer: How many **Lists** have been declared?*

Source: Created by the author

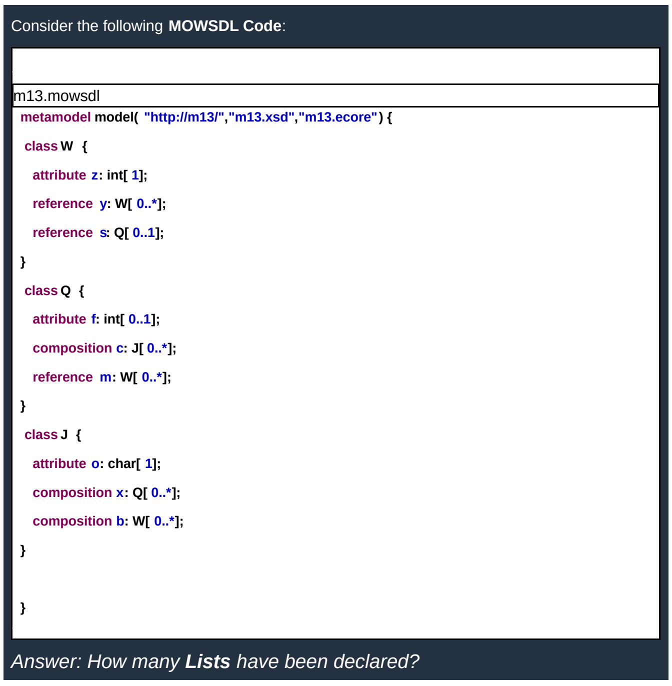
Figure 105 – List Counting Subtask 2 Java


Consider the following **Java Code**:

J.java
```
package model;
 class J {
 char  o;
 List<Q>  x;
 List<W>  b;
 }
```
Q.java
```
package model;
 class Q {
 int  f;
 List<J>  c;
 List<W>  m;
 }
```
W.java
```
package model;
 class W {
 int  z;
 List<W>  y;
 Q  s;
 }
```
*Answer: How many **Lists** have been declared?*

Source: Created by the author

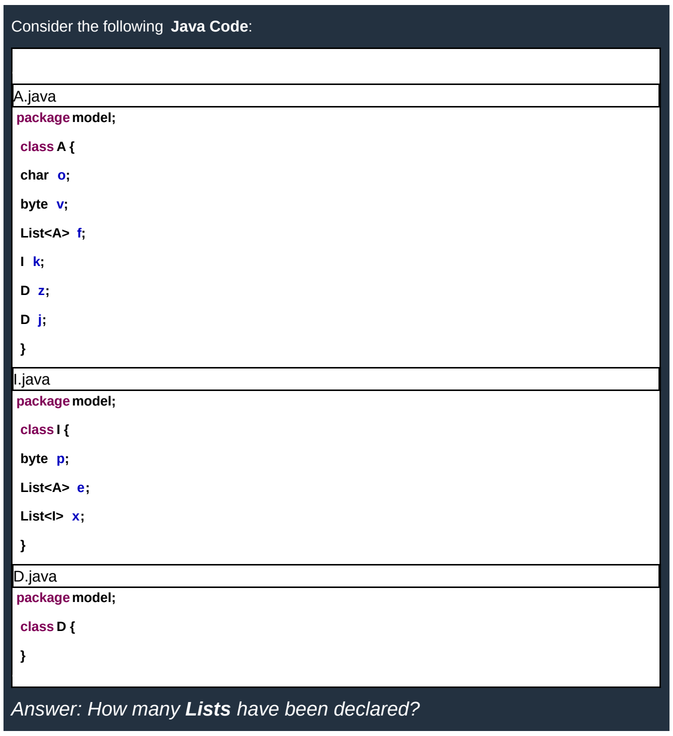
Figure 106 – List Counting Subtask 2 MOWSDL


Consider the following **MOWSDL Code**:

m13.mowsdl
```
 metamodel model( "http://m13/","m13.xsd","m13.ecore") {
  class W  {
    attribute z: int[ 1];
    reference  y: W[ 0..*];
    reference  s: Q[ 0..1];
 }
  class Q  {
    attribute f: int[ 0..1];
    composition c: J[ 0..*];
    reference  m: W[ 0..*];
 }
  class J  {
    attribute o: char[ 1];
    composition x: Q[ 0..*];
    composition b: W[ 0..*];
 }


 }
```
*Answer: How many **Lists** have been declared?*

Source: Created by the author

Figure 107 – List Counting Subtask 3 Java



Consider the following **Java Code**:

A.java
```
package model;
class A {
char  o;
byte  v;
List<A>  f;
I  k;
D  z;
D  j;
}
```

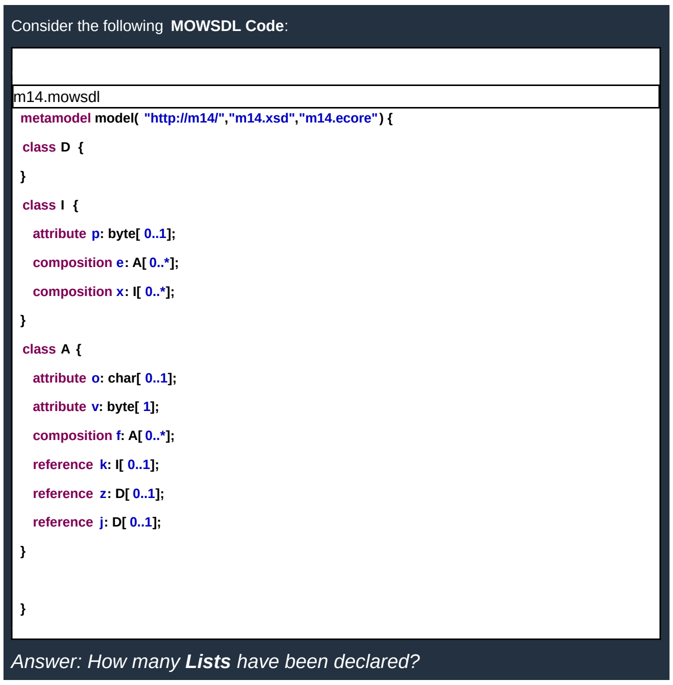
I.java
```
package model;
class I {
byte  p;
List<A>  e;
List<I>  x;
}
```

D.java
```
package model;
class D {
}
```

*Answer: How many **Lists** have been declared?*

Source: Created by the author

Figure 108 – List Counting Subtask 3 MOWSDL



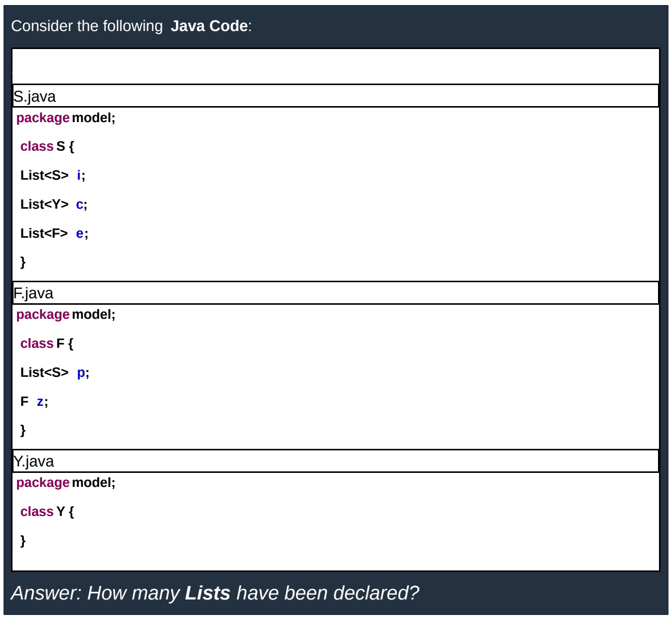
Consider the following **MOWSDL Code**:

m14.mowsdl
```
metamodel model( "http://m14/","m14.xsd","m14.ecore") {
 class D  {
}
 class I  {
   attribute  p: byte[ 0..1];
   composition e: A[ 0..*];
   composition x: I[ 0..*];
}
 class A  {
   attribute  o: char[ 0..1];
   attribute  v: byte[ 1];
   composition f: A[ 0..*];
   reference  k: I[ 0..1];
   reference  z: D[ 0..1];
   reference  j: D[ 0..1];
}


}
```

*Answer: How many **Lists** have been declared?*

Source: Created by the author

Figure 109 – List Counting Subtask 4 Java


Consider the following **Java Code**:

S.java

```
package model;
 class S {
 List<S>  i;
 List<Y>  c;
 List<F>  e;
 }
```

F.java

```
package model;
 class F {
 List<S>  p;
 F  z;
 }
```

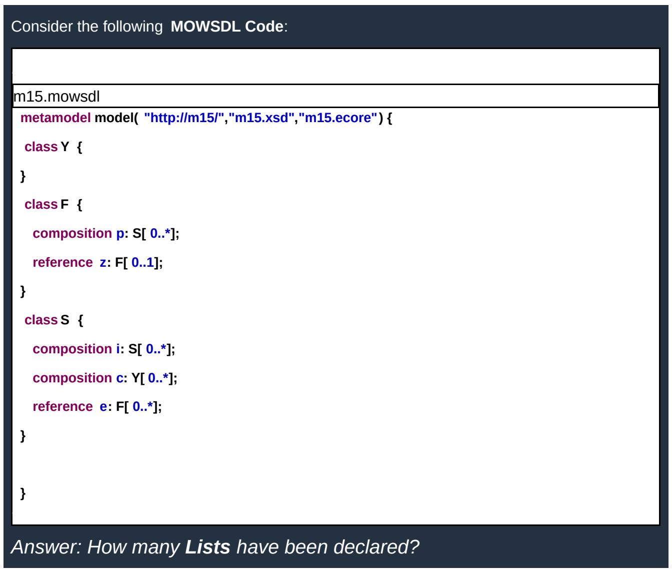
Y.java

```
package model;
 class Y {
 }
```

*Answer: How many **Lists** have been declared?*

Source: Created by the author

Figure 110 – List Counting Subtask 4 MOWSDL


Consider the following **MOWSDL Code**:

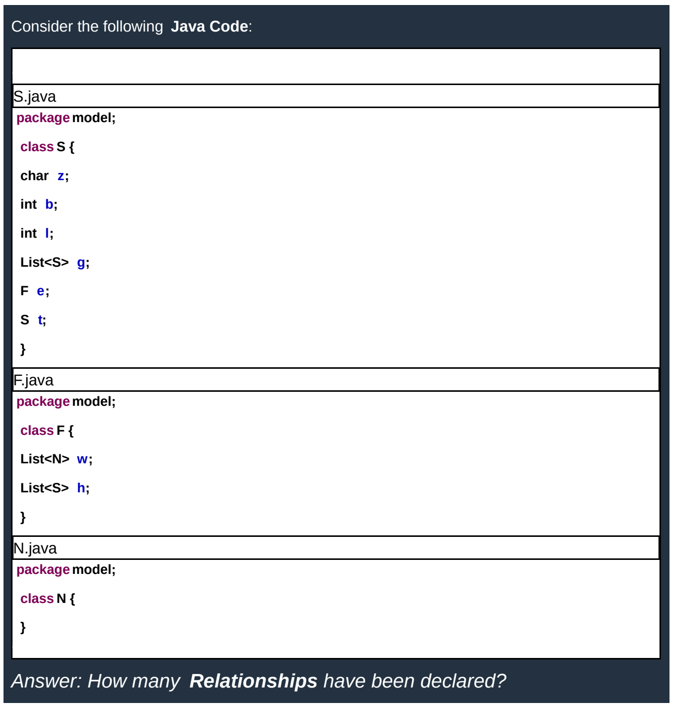
m15.mowsdl

```
 metamodel model( "http://m15/","m15.xsd","m15.ecore") {
  class Y  {
 }
  class F  {
    composition p: S[ 0..*];
    reference  z: F[ 0..1];
 }
  class S  {
    composition i: S[ 0..*];
    composition c: Y[ 0..*];
    reference  e: F[ 0..*];
 }


 }
```

*Answer: How many **Lists** have been declared?*

Source: Created by the author

Figure 111 – Relationship Counting Subtask 1 Java



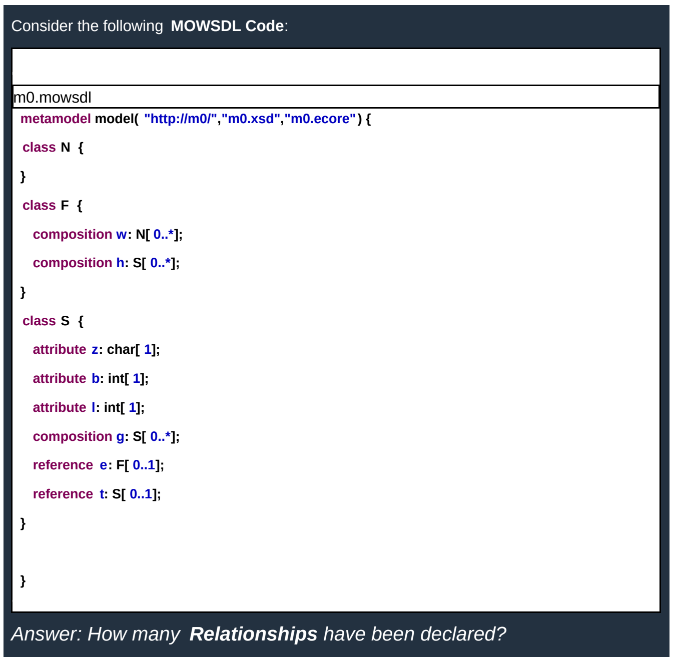
Consider the following **Java Code**:

```
S.java
package model;
 class S {
 char  z;
 int  b;
 int  l;
 List<S>  g;
 F  e;
 S  t;
 }
F.java
package model;
 class F {
 List<N>  w;
 List<S>  h;
 }
N.java
package model;
 class N {
 }
```

Answer: How many ***Relationships*** *have been declared?*

Source: Created by the author

Figure 112 – Relationship Counting Subtask 1 MOWSDL



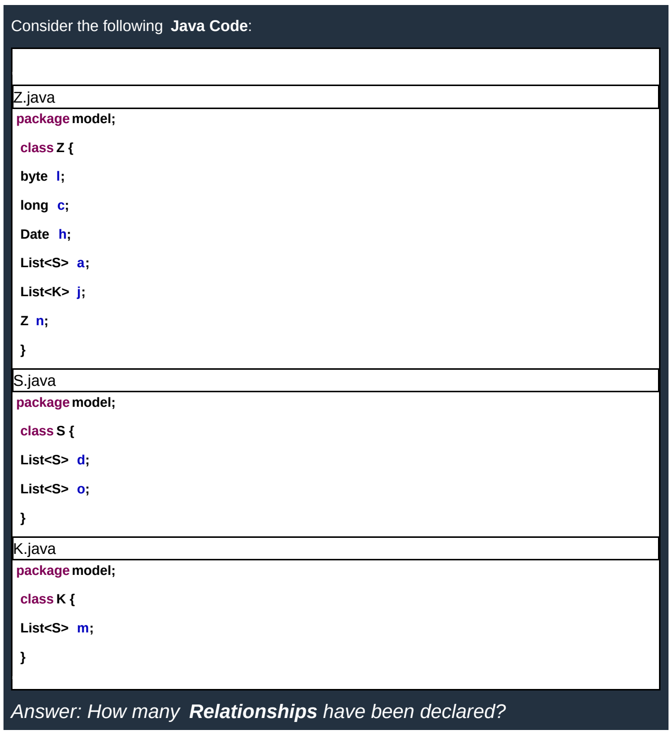
Consider the following **MOWSDL Code**:

```
m0.mowsdl
 metamodel model( "http://m0/","m0.xsd","m0.ecore") {
  class N  {
 }
  class F  {
    composition w: N[ 0..*];
    composition h: S[ 0..*];
 }
  class S  {
    attribute  z: char[ 1];
    attribute  b: int[ 1];
    attribute  l: int[ 1];
    composition g: S[ 0..*];
    reference  e: F[ 0..1];
    reference  t: S[ 0..1];
 }


 }
```

Answer: How many ***Relationships*** *have been declared?*

Source: Created by the author

Figure 113 – Relationship Counting Subtask 2 Java



Consider the following **Java Code**:

Z.java

```
package model;
 class Z {
 byte  l;
 long  c;
 Date  h;
 List<S>  a;
 List<K>  j;
 Z  n;
 }
```

S.java

```
package model;
 class S {
 List<S>  d;
 List<S>  o;
 }
```

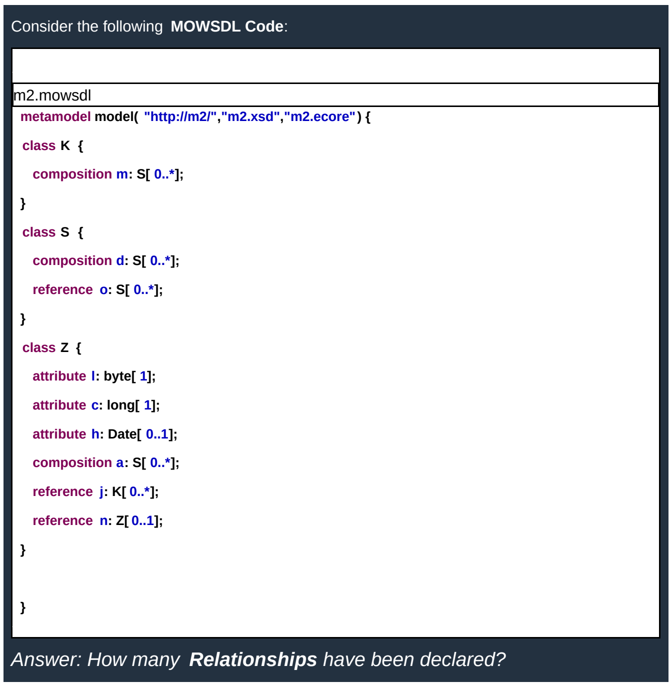
K.java

```
package model;
 class K {
 List<S>  m;
 }
```

*Answer: How many **Relationships** have been declared?*

Source: Created by the author

Figure 114 – Relationship Counting Subtask 2 MOWSDL



Consider the following **MOWSDL Code**:

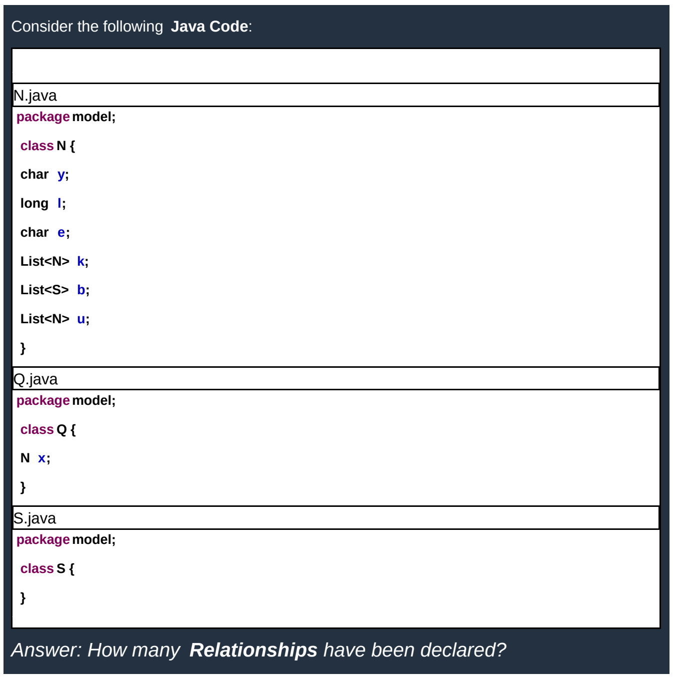
m2.mowsdl

```
 metamodel model( "http://m2/","m2.xsd","m2.ecore") {
  class K  {
    composition m: S[ 0..*];
 }
  class S  {
    composition d: S[ 0..*];
    reference  o: S[ 0..*];
 }
  class Z  {
    attribute  l: byte[ 1];
    attribute  c: long[ 1];
    attribute  h: Date[ 0..1];
    composition a: S[ 0..*];
    reference  j: K[ 0..*];
    reference  n: Z[ 0..1];
 }


 }
```

*Answer: How many **Relationships** have been declared?*

Source: Created by the author

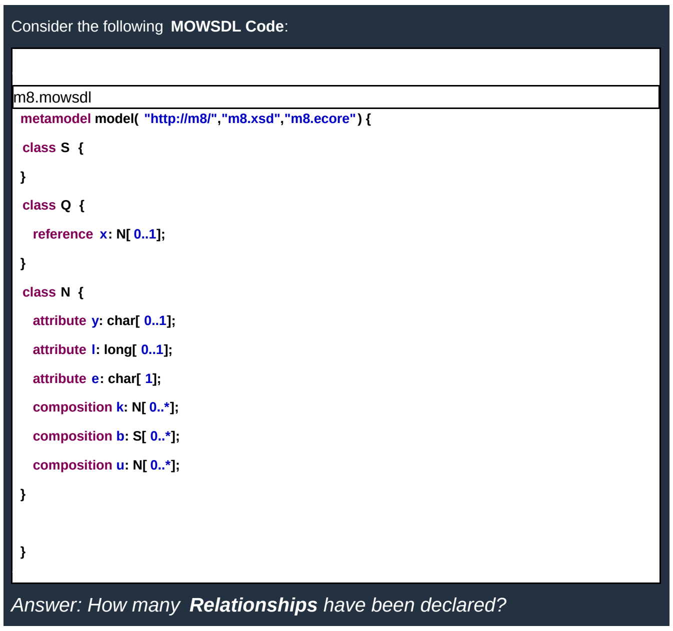
Figure 115 – Relationship Counting Subtask 3 Java


Consider the following **Java Code**:

N.java
```
package model;
 class N {
 char  y;
 long  l;
 char  e;
 List<N>  k;
 List<S>  b;
 List<N>  u;
 }
```
Q.java
```
package model;
 class Q {
 N  x;
 }
```
S.java
```
package model;
 class S {
 }
```
*Answer: How many **Relationships** have been declared?*

Source: Created by the author

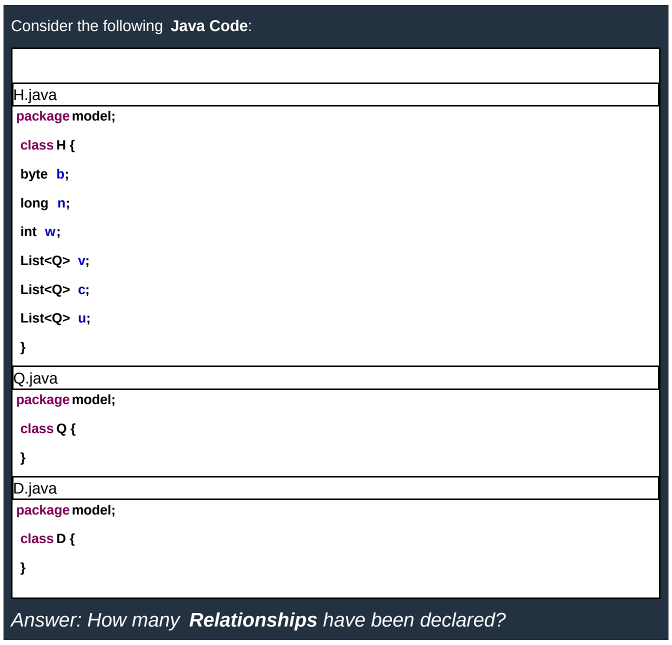
Figure 116 – Relationship Counting Subtask 3 MOWSDL


Consider the following **MOWSDL Code**:

m8.mowsdl
```
 metamodel model( "http://m8/","m8.xsd","m8.ecore") {
 class S  {
 }
 class Q  {
   reference  x: N[ 0..1];
 }
 class N  {
   attribute  y: char[ 0..1];
   attribute  l: long[ 0..1];
   attribute  e: char[ 1];
   composition k: N[ 0..*];
   composition b: S[ 0..*];
   composition u: N[ 0..*];
 }


 }
```
*Answer: How many **Relationships** have been declared?*

Source: Created by the author

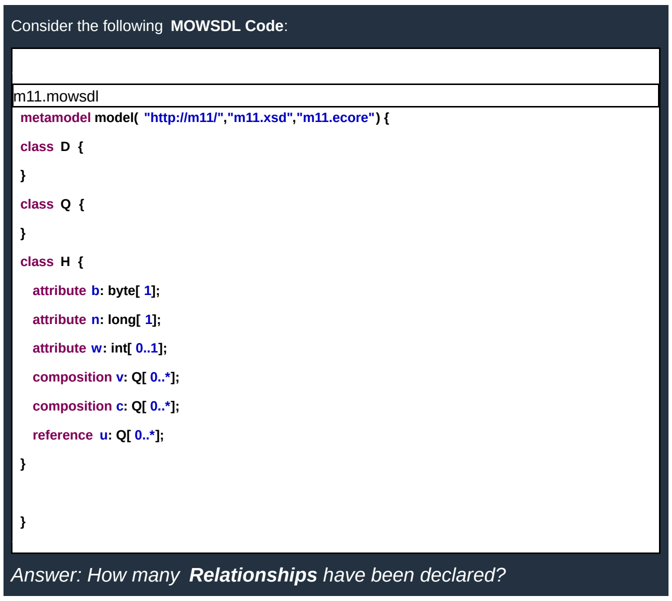
Figure 117 – Relationship Counting Subtask 4 Java


Consider the following **Java Code**:

H.java
```
package model;
 class H {
 byte  b;
 long  n;
 int  w;
 List<Q>  v;
 List<Q>  c;
 List<Q>  u;
 }
```
Q.java
```
package model;
 class Q {
 }
```
D.java
```
package model;
 class D {
 }
```
*Answer: How many **Relationships** have been declared?*

Source: Created by the author

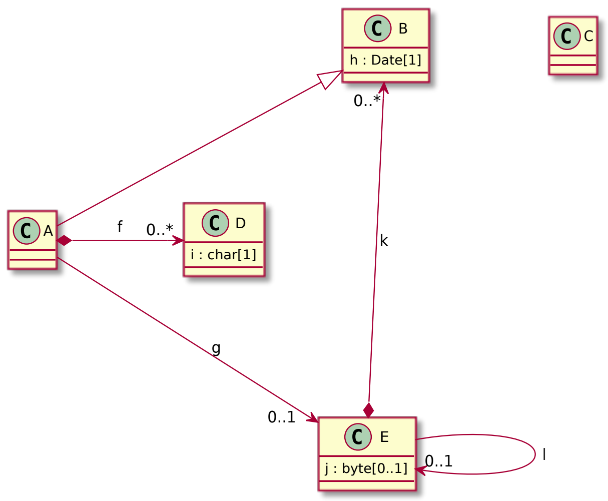
Figure 118 – Relationship Counting Subtask 4 MOWSDL


Consider the following **MOWSDL Code**:

m11.mowsdl
```
 metamodel model( "http://m11/","m11.xsd","m11.ecore") {
 class  D  {
 }
 class  Q  {
 }
 class  H  {
    attribute  b: byte[ 1];
    attribute  n: long[ 1];
    attribute  w: int[ 0..1];
    composition  v: Q[ 0..*];
    composition  c: Q[ 0..*];
    reference  u: Q[ 0..*];
 }


 }
```
*Answer: How many **Relationships** have been declared?*

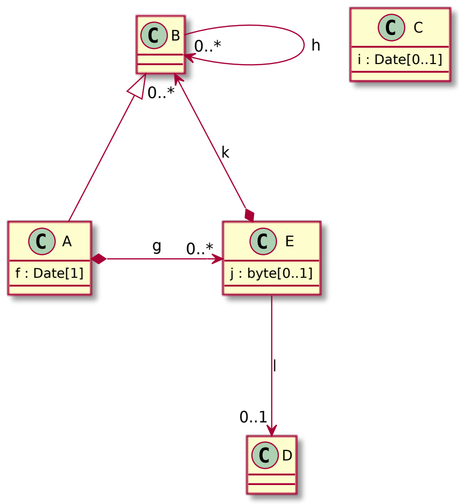
Source: Created by the author

Figure 119 – A17 Class Diagram



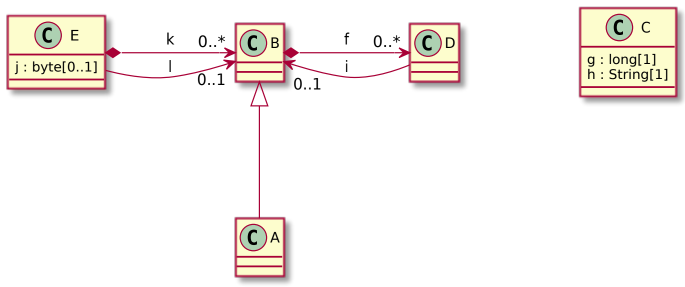
Source: Created by the author

Figure 120 – A26 Class Diagram



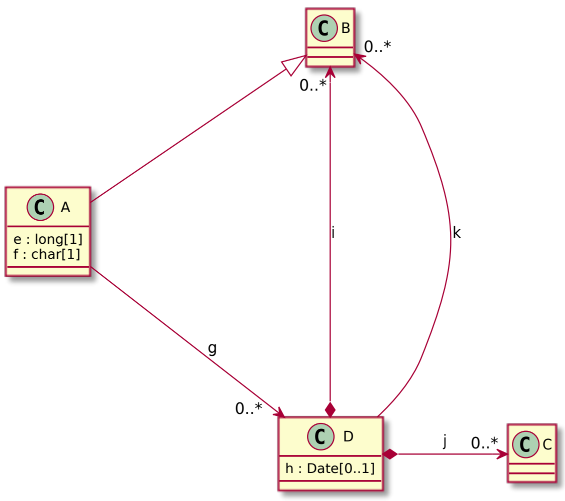
Source: Created by the author

Figure 121 – A20 Class Diagram



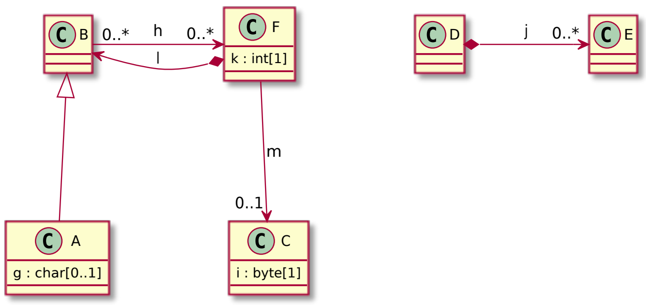
Source: Created by the author

Figure 122 – A34 Class Diagram



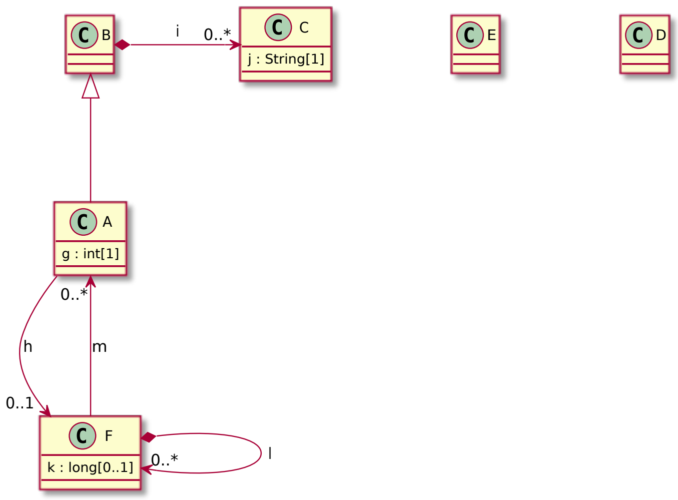
Source: Created by the author

Figure 123 – A23 Class Diagram



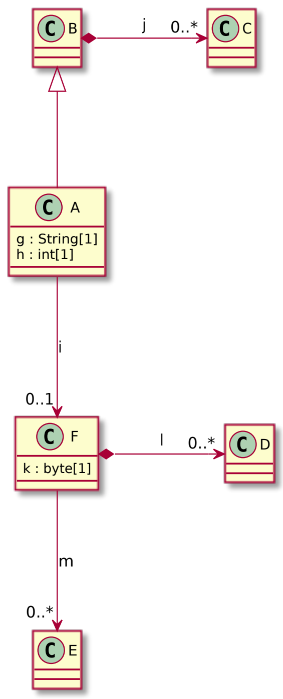
Source: Created by the author

Figure 124 – A47 Class Diagram



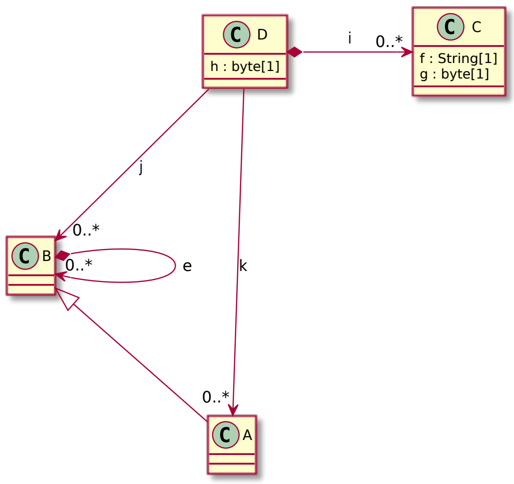
Source: Created by the author

Figure 125 – A4 Class Diagram



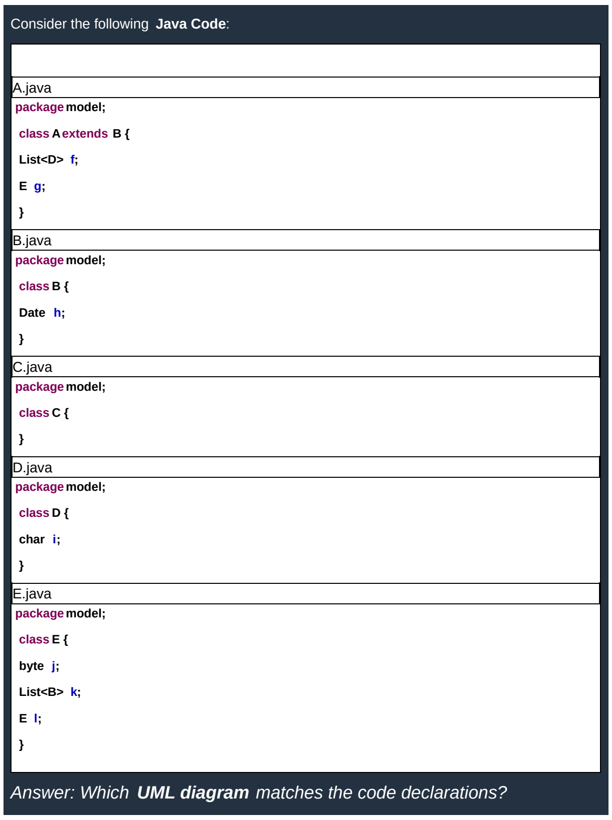
Source: Created by the author

Figure 126 – A29 Class Diagram



Source: Created by the author

Figure 127 – Diagram Matching Subtask 1 Java

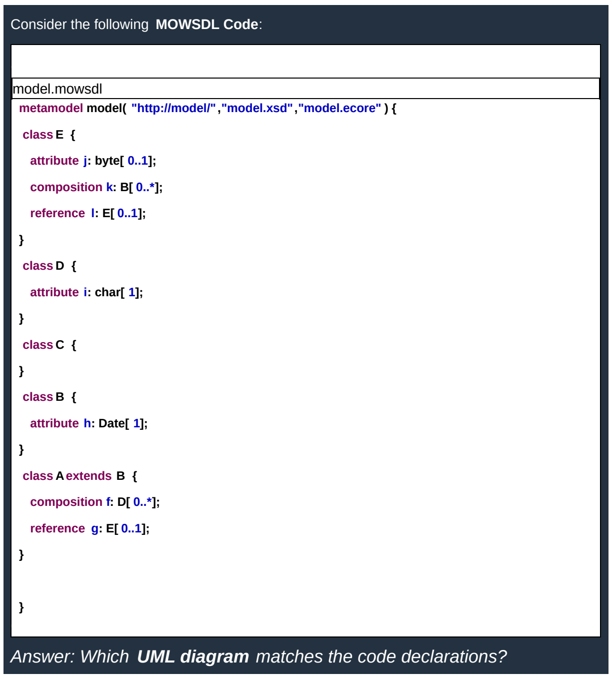


Consider the following **Java Code**:

A.java
```
package model;
 class A extends B {
 List<D> f;
 E g;
 }
```
B.java
```
package model;
 class B {
 Date h;
 }
```
C.java
```
package model;
 class C {
 }
```
D.java
```
package model;
 class D {
 char i;
 }
```
E.java
```
package model;
 class E {
 byte j;
 List<B> k;
 E l;
 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

Figure 128 – Diagram Matching Subtask 1 MOWSDL

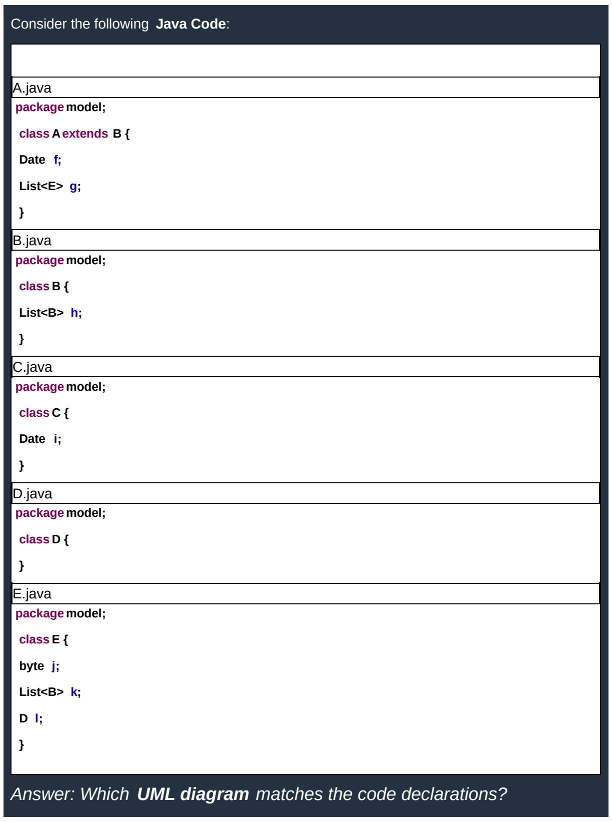


Consider the following **MOWSDL Code**:

model.mowsdl
```
 metamodel model( "http://model/","model.xsd","model.ecore" ) {
  class E {
    attribute j: byte[ 0..1];
    composition k: B[ 0..*];
    reference l: E[ 0..1];
 }
  class D {
    attribute i: char[ 1];
 }
  class C {
 }
  class B {
    attribute h: Date[ 1];
 }
  class A extends B {
    composition f: D[ 0..*];
    reference g: E[ 0..1];
 }


 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

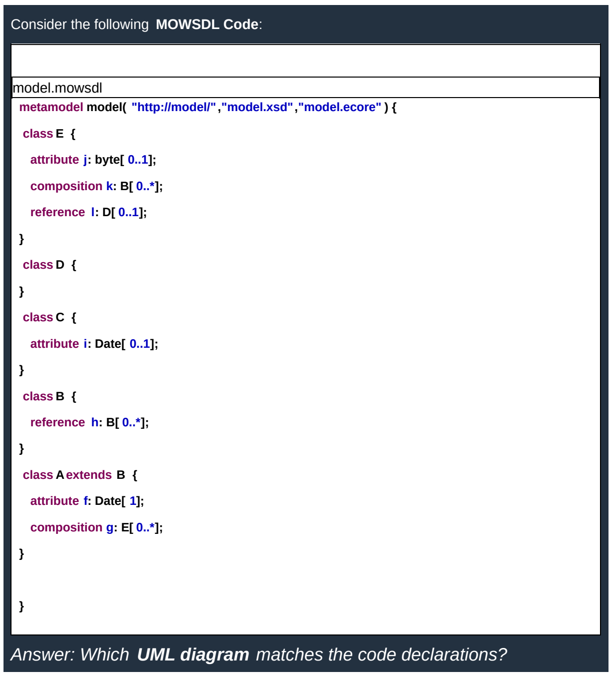
Figure 129 – Diagram Matching Subtask 2 Java


Consider the following **Java Code**:

A.java
```
package model;
 class A extends B {
 Date  f;
 List<E>  g;
 }
```
B.java
```
package model;
 class B {
 List<B>  h;
 }
```
C.java
```
package model;
 class C {
 Date  i;
 }
```
D.java
```
package model;
 class D {
 }
```
E.java
```
package model;
 class E {
 byte  j;
 List<B>  k;
 D  l;
 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

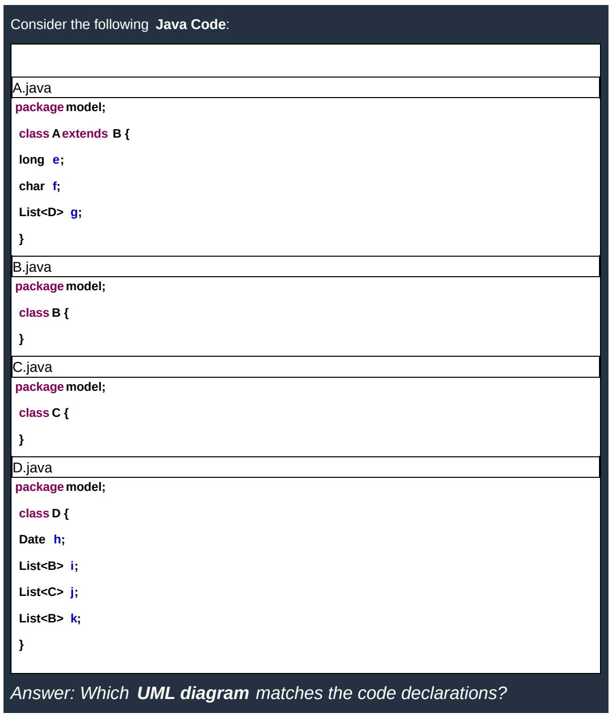
Figure 130 – Diagram Matching Subtask 2 MOWSDL


Consider the following **MOWSDL Code**:

model.mowsdl
```
 metamodel model( "http://model/","model.xsd","model.ecore" ) {
  class E  {
    attribute j: byte[ 0..1];
    composition k: B[ 0..*];
    reference  l: D[ 0..1];
 }
  class D  {
 }
  class C  {
    attribute i: Date[ 0..1];
 }
  class B  {
    reference  h: B[ 0..*];
 }
  class A extends B  {
    attribute f: Date[ 1];
    composition g: E[ 0..*];
 }


 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

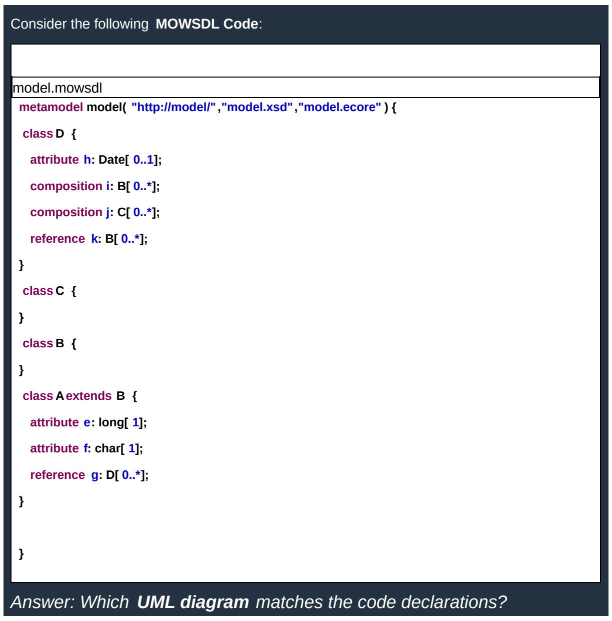
Figure 131 – Diagram Matching Subtask 3 Java


Consider the following **Java Code**:

A.java
```
package model;
 class A extends B {
 long e;
 char f;
 List<D> g;
 }
```
B.java
```
package model;
 class B {
 }
```
C.java
```
package model;
 class C {
 }
```
D.java
```
package model;
 class D {
 Date h;
 List<B> i;
 List<C> j;
 List<B> k;
 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

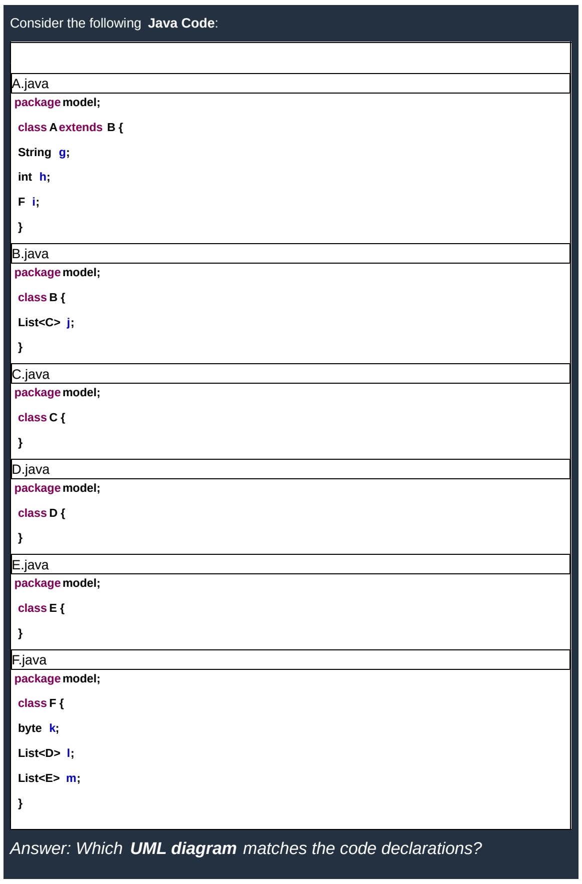
Figure 132 – Diagram Matching Subtask 3 MOWSDL


Consider the following **MOWSDL Code**:

model.mowsdl
```
 metamodel model( "http://model/","model.xsd","model.ecore" ) {
  class D {
    attribute h: Date[ 0..1];
    composition i: B[ 0..*];
    composition j: C[ 0..*];
    reference k: B[ 0..*];
 }
  class C {
 }
  class B {
 }
  class A extends B {
    attribute e: long[ 1];
    attribute f: char[ 1];
    reference g: D[ 0..*];
 }


 }
```
*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

Figure 133 – Diagram Matching Subtask 4 Java



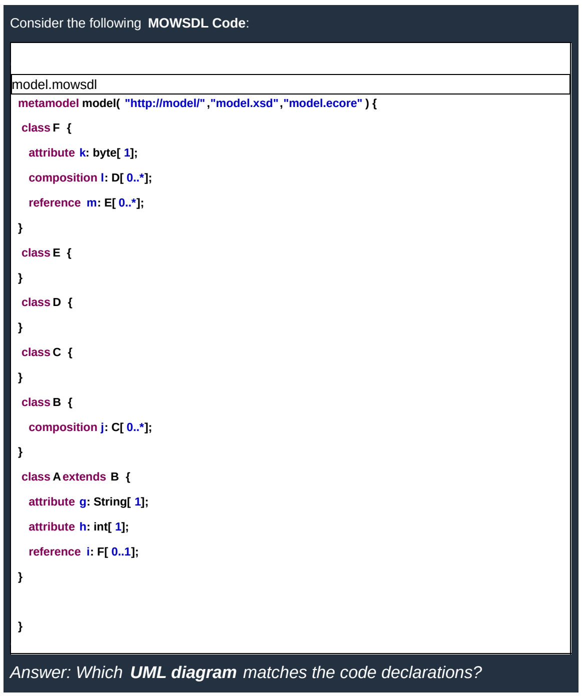
Consider the following **Java Code**:

A.java
```
package model;
class A extends B {
String g;
int h;
F i;
}
```
B.java
```
package model;
class B {
List<C> j;
}
```
C.java
```
package model;
class C {
}
```
D.java
```
package model;
class D {
}
```
E.java
```
package model;
class E {
}
```
F.java
```
package model;
class F {
byte k;
List<D> l;
List<E> m;
}
```

*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

Figure 134 – Diagram Matching Subtask 4 MOWSDL



Consider the following **MOWSDL Code**:

model.mowsdl
```
metamodel model( "http://model/","model.xsd","model.ecore" ) {
 class F {
   attribute k: byte[ 1];
   composition l: D[ 0..*];
   reference m: E[ 0..*];
}
 class E {
}
 class D {
}
 class C {
}
 class B {
   composition j: C[ 0..*];
}
 class A extends B {
   attribute g: String[ 1];
   attribute h: int[ 1];
   reference i: F[ 0..1];
}


}
```

*Answer: Which **UML diagram** matches the code declarations?*

Source: Created by the author

Table 72 – Feedback Attribute List and Possible Answers

Attribute Number	Attribute Text	Answer Range	Correct Answer
1	I like MOWSDL	Null or -2 to +2	Any
2	I like Java	Null or -2 to +2	Any
3	it was too long	Null or -2 to +2	Any
4	the survey looks good	Null or -2 to +2	Any
5	it was hard	Null or -2 to +2	Any
6	it was boring	Null or -2 to +2	Any
7	Let's collaborate on research!	Null or -2 to +2	Any
8	I like your survey	Null or -2 to +2	Any
9	It made me sleepy	Null or -2 to +2	Any
10	it was easy	Null or -2 to +2	Any
11	the survey is ugly	Null or -2 to +2	Any
12	it was inspiring	Null or -2 to +2	Any
13	I like UML	Null or -2 to +2	Any
14	it was useful for me	Null or -2 to +2	Any
15	it was short	Null or -2 to +2	Any
16	it was fun	Null or -2 to +2	Any

APPENDIX

E

IMPLEMENTATION STUDY INSTRUMENTATION DOCUMENTS

E.1 Initial Remarks

This appendix chapter includes the manuals used by participants during training and operation of the Data Structure Implementation Experiment presented in Chapter 7 (Section 7.3).

This chapter is divided into two manuals, the first manual is used for MOWSDL (Section E.3) while the second is used for Java (JAX) implementation (Section E.4).

E.2 Retail System Example

Throughout this manual, a shop sales software system is used as an example. This software is structured as in a client-server architecture. The client is responsible to manage the cart of the customer while the server stores the product information, collects the final cart managed by the client and executes the final checkout, as illustrated in the use case diagram shown in Figure 135.

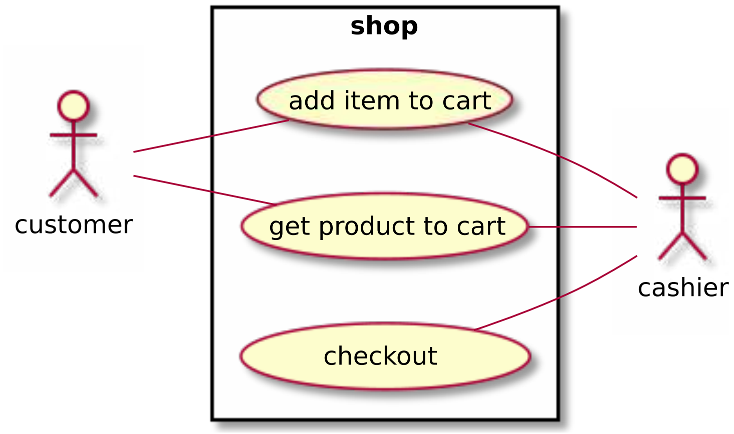
E.3 MOWSDL Data Structure Declaration Manual

E.3.1 Manual Introduction

MOWSDL (Model-Oriented Web Service Definition Language) is a language defined within this project. The objective of this language is to provide a language definition similar to WSDL for describing MOWS systems.

Similarly to WSDL and WADL, MOWSDL has an XML structure. It also follows modeling principles by complying to the XMI standard. On top of that, an alternate textual

Figure 135 – Shop System Use Case Diagram

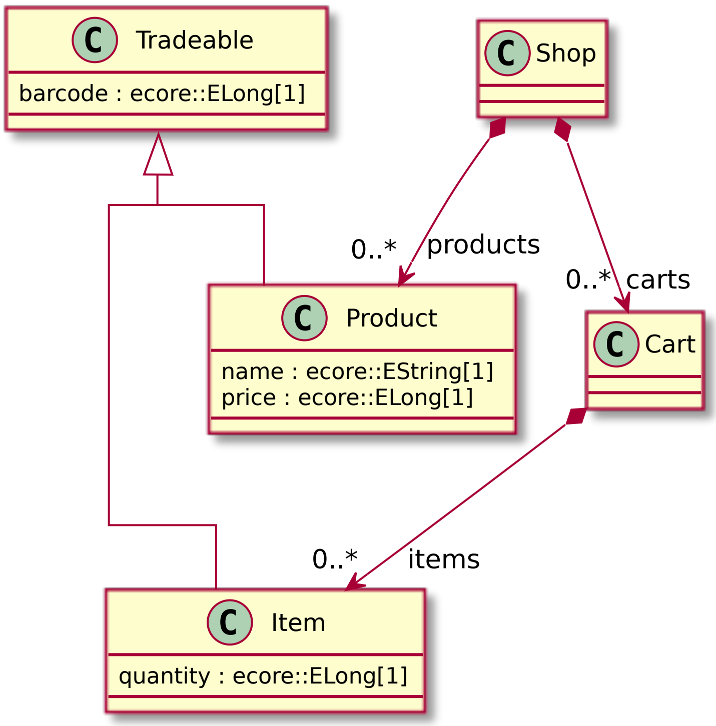


Source: Created by the author

representation (concrete syntax) has been defined, allowing developers to write MOWSDL in a text without XML idiosyncrasies. This concrete syntax is described herein as the recommended syntax for defining MOWSDL instances.

MOWSDL can be used to replace WSDL and WADL documents for MOWS systems while using SOAP as well as providing a common SOAP and REST document interface since it can also be used to generate REST WS systems without changes.

Figure 136 – Designed Data Classes (MOWSDL Manual)



Source: Created by the author

The data class diagram is visible in Figure 136. The data classes are divided into Shop (which stores the cart data and product data); Carts (which stores the items selected by customers, declaration present in Figure 146); Products (which are the registry of products sold by the shop); and Items (which are the quantifier objects of products, present in Figure 147). *Tradeable* is the generalization for the "barcode" attribute.

The server and client operations are also designed, however, their implementation is out of the scope of this manual.

E.3.2 MOWSDL Declarations

As MOWDL is a declarative language, each document is structured as declarations. A document may have up to 3 (three) kinds of declarations, which are completely optional. This implies that an empty document is also valid, however, this represents a system which has no interfaces or data types for communication.

The first declaration is used to import referenced metamodels and XSD files that are used to define the data types to be employed by the MOWS system.

E.3.3 Imports

The import syntax of MOWSDL is illustrated in Figure 137. Its import syntax starts either by the "uses" or "import" reserved word. This was planned to differentiate the XSD and Metamodel imports, however, they are treated as synonyms in the language. following the reserved word, an internal name to reference the imported file is added, which is then followed by path specifications inside parenthesis. These paths should declare the target namespace, optional source XSD file and optional source metamodel file. The optional paths are to be provided in order.

Therefore, when specifying a metamodel, its related XSD file must be provided. This is not a limitation since XSD generators for metamodels have been provided, both the author and the EMF project have equally adequate tools for this concern.

Figure 137 – Imports Declaration

```
using  xsd("http://www.w3.org/2001/XMLSchema");

using  xmi("http://www.omg.org/XMI");

import  ecore("http://www.eclipse.org/emf/2002/Ecore","Ecore.ecore","Ecore.xsd");
```

Source: Created by the author

E.3.4 Metamodel

The second declaration is used to declare the main metamodel to be employed by the system. It is illustrated by Figure 138. This metamodel can be either imported or completely declared within the document, including metapackages, metaenumerations, metainterfaces and metaclasses. Metaclasses may have generalizations directed towards other metaclasses. Metaclasses may also have attributes and relationships, both with multiplicity support.

Relationships are data types declared in external metamodels or inside the same MOWSDL document. These are defined as one of two categories: associations (object references) and compositions (objects are part of the owner). Attributes are declared similarly to compositions, however, they are only used when declaring a fundamental data type which must be specified by an external XSD, e.g. Strings and Integers.

E.3.4.1 Interface

The last declaration is used to define the MOWS system interfaces. An example is present in Figure 139. The interface declaration represents the port to connect clients to servers of a MOWS system. This part of MOWSDL is not evaluated in this study.

Within the interface declaration, the developer should define message types, which represent sets of parameters. These message types are then used as input and/or outputs for the operations (which represent interface methods).

The MOWSDL is provided with an editor and compilers developed by the author.

E.3.5 Classes and Inheritance

Classes are declared as metaclasses in MOWSDL since they are compiled as part of a metamodel. The metaclass declaration syntax is similar to Java programming language, but by using a metaclass reserved word, an example is provided in Figure 140.

Metaclasses may be abstract by prepending an “abstract” reserved word prior to the metaclass. Metaclasses may extend other metaclasses. Multiple inheritance is supported by using a comma separated list.

E.3.6 Class Owned Elements

MOWSDL was also designed to support metaoperations. These metaoperations allows the developer to declare method signatures that are added to the data structure objects of a MOWS system.

MetaInterfaces can only include metaoperations. MetaEnumerations are lists of literals that can be used as types.

Figure 138 – Metamodel Declaration

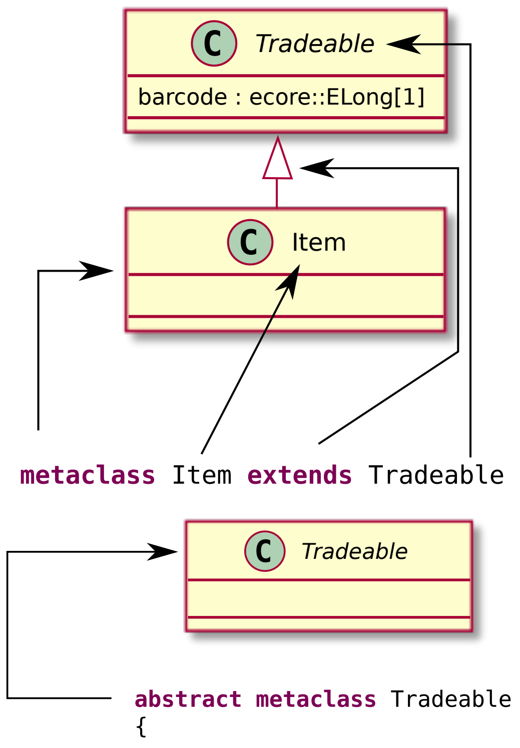
```
metamodel  store("Store","http://store/","store.xsd")

{

 metaclass  Shop

  {

  composition  carts : Cart[*];

  composition  products : Product[*];

  }


 abstract  metaclass  Tradeable

  {

  attribute  barcode  :  ecore::ELong[1];

  }


 metaclass  Cart

 {

  composition  items  : Item[1];

 }


 metaclass  Product extendsTradeable

 {

  attribute  name  :  ecore::EString[1];

  attribute  price  :  ecore::ELong[1];

 }


 metaclass  Item extends  Tradeable

 {

  attribute quantity : ecore::ELong[1];

  reference items1 : Item[0..*];

  composition items2 : Item[0..*];

 }

}
```

Source: Created by the author

Figure 139 – MOWSDL Interface Declaration

```
interface storews( “http://store/wsdl/” )

 {


  // implied import store(“Store”,“http://store/”,“store.xsd”);


  message BarcodeMessage

 {

  part barcode : xsd::integer;

 }


  message ProductMessage

 {

  part product : store::product;

 }


  message CartMessage

 {

  part cart : store::cart;

 }


  message ResponseSuccess

 {

  part response : xsd::string;

 }


  port storewsp

 {

  operation getProduct(BarcodeMessage):ProductMessage;

  operation addCart(CartMessage):ResponseSuccess;

 }


  service StoreService( “store_bind” , “action” , “http://localhost:9000/” );

}
```

Figure 140 – MOWSDL Class Declaration


Tradeable
barcode : ecore::ELong[1]
Item
metaclass Item extends Tradeable
Tradeable
abstract metaclass Tradeable
{

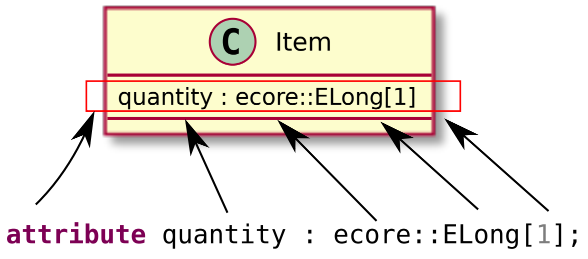
Source: Created by the author

E.3.7 Class Attributes

As illustrated in Figure 141, class attributes are declared in MOWSDL similarly to UML notation, except that it requires the reserved word “attribute” followed by the name of the attribute, a colon, the name of the package of the data type, a pair of colons, the name of the data type itself and the multiplicity between braces.

The valid multiplicities are: [0..1]; [0..*] or [*]; [1..1] or [1]; [1..*].

Figure 141 – MOWSDL Class Attributes

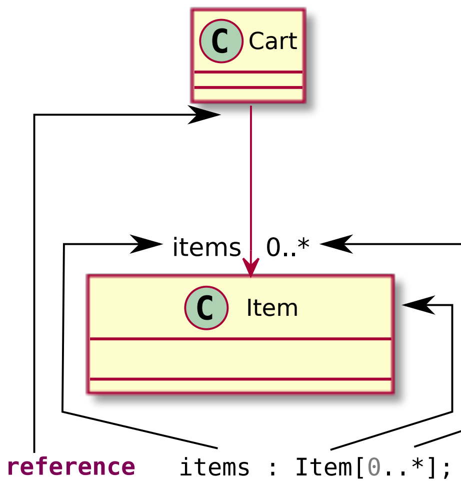

Item
quantity : ecore::ELong[1]
attribute quantity : ecore::ELong[1];

Source: Created by the author

E.3.7.1 Class Associations

As visible in Figure 142, class compositions are declared similarly to references, however the reserved word is “reference”. The associations must refer to classes, i.e. fundamental data types are not valid.

Figure 142 – MOWSDL Class References



Source: Created by the author

E.3.8 Class Compositions

Class compositions are declared similarly to associations, however the reserved word is “composition”. This is illustrated in Figure 143.

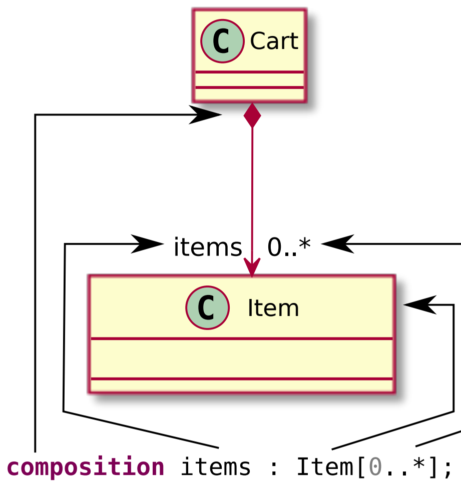
E.3.9 Other Features

MOWSDL was also designed to support inline method declaration for classes and interfaces. An editor and a set of compilers is also available. The editor supports automatic syntax validation and code completion. Compilers are available for round-trip engineering when using XSD, metamodels and CXF source code. The compilers are also capable of saving the manual modifications performed by the developer during round-trip engineering.

E.3.10 Compiling MOWSDL

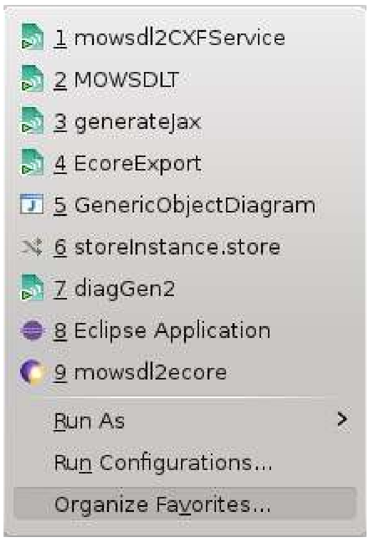
In order to complile MOWSDL to metamodels and CXF code, compilers have been provided. During the experiment, compilers can be accessed by using the menu visible in Figure 144. They include validation and data-type matching to optimize the output without forcing the programmers to deal with unnecessary data formats libraries, e.g. it is not required to import XSD data types into source code.

Figure 143 – MOWSDL Class Composition



Source: Created by the author

Figure 144 – Compiler Access



Source: Created by the author

E.3.11 Testing the Result

The compilers used throughout the study include test case support to verify whether the generated code complies to the requirements.

E.3.12 MOWSDL Execution Workflow

It is recommended to follow this execution while working on the provided exercises.

1. Run “Record starting time”;

2. Run exercise test case (halt if test case succeeds);
3. Edit MOWSDL code according to diagram;
4. Run MOWSDLT (parser);
5. Run JAX Code Generator;
6. Return to item 2.

E.4 JAX Data Structure Declaration Manual

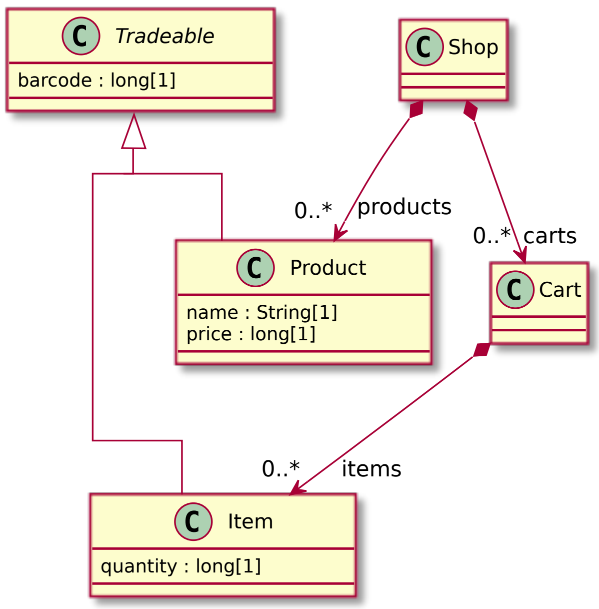
E.4.1 Manual Introduction

JAX (Java API for XML) is an application programming interface specified by Oracle Corporation (2017). It is composed by annotations to bound Java classes and their owned fields (properties) to XML and JSON formats.

JAX allows the developers to specify XML data structures without relying on XSD (XML Schema Definition) to define the format.

This API can be employed for XML and JSON, supporting generation and parsing. It can be used for transferring the formatted data in web services. For instance, it can be employed by both SOAP and REST protocols/architectures without modifying the data structure classes.

Figure 145 – Designed Data Classes (JAX Manual)



Source: Created by the author

The data class diagram is visible in Figure 145. The data classes are divided into Shop (which stores the cart data and product data); Carts (which stores the items selected by customers, declaration present in Figure 146); Products (which are the registry of products sold by the shop); and Items (which are the quantifier objects of products, present in Figure 147). *Tradeable* is the generalization for the “barcode” attribute.

The server and client operations are also designed, however, their implementation is out of the scope of this manual.

Figure 146 – Cart JAX Class Declaration

```
package store;


import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

import javax.xml.bind.annotation.XmlType;


@ XmlRootElement

@ XmlAccessorType (XmlAccessType. FIELD )

@ XmlType (name =  “Cart” , propOrder = { “items” })

public  class Cart {


 protected Item  items ;


 /**

*  field getters and setters

*/

 public Item getItems() {

 return  this . items ;

}


 public  void setItems(Item element) {

 this . items = element;

}


}
```

Source: Created by the author

Figure 147 – Item JAX Class Declaration

```
package store;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlType;


@ XmlAccessorType (XmlAccessType. FIELD )

@ XmlType (name =  "Item" , propOrder = { "quantity" })

public  class Item  extends Tradeable {


 protected  long  quantity ;


 /**

*  field getters and setters

*/

 public  long getQuantity() {

 return  this . quantity ;

}

 public  void setQuantity( long element) {


 this . quantity = element;

}


}
```

Source: Created by the author

E.4.2 Classes and Inheritance

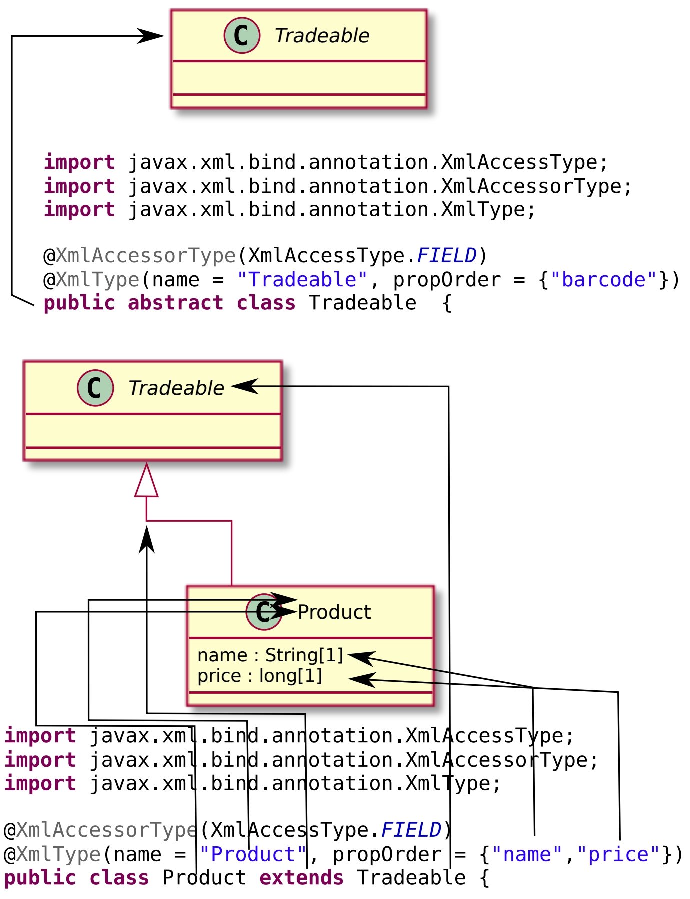
In JAX, Java classes are declared with annotations that bind the class to XML structures, for example, as shown in Figure 148.

A Java class must be declared for each data class. They must be annotated with "@Xml-Type". It is recommended to also provide their name in XML and property order to avoid compatibility problems.

The "@XmlAccessorType" is used to specify how to bind the class attributes to XML node attributes. In this study, the recommended setting is "XmlAccessType.FIELD".

Besides these annotations, XmlRootElement is used to annotate an XmlType that can be used as the root of the XML file, i.e. the first node that is composed by every other inner node. Every XML instance is a tree with exactly one root, regardless if it is written to file or transferred by a WS. Therefore, it is required to have at least one possible root element type per XML specification. In most cases, it is possible to simply add this setting to every XmlType.

Figure 148 – JAX Class Declaration



```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Tradeable", propOrder = {"barcode"})
public abstract class Tradeable  {
```

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Product", propOrder = {"name","price"})
public class Product extends Tradeable {
```

Source: Created by the author

E.4.3 Class Owned Elements

E.4.3.1 Class Attributes

JAX was based on Java Beans specifications. An example of attribute declaration is visible in Figure 149. The attributes are actually managed by getter and setter methods. The properties/fields actual must be accessed by these methods.

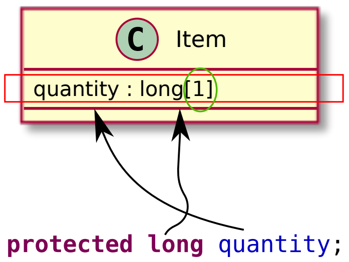
Therefore, getters and setters are mandatory and they must affect the data of the intended field.

For **multiplicities 0..1 and 1..1**, the field must match the counterpart java type presented in the design diagram.

The getter method must start with "get" concatenated to the name of the field with its first letter capitalized. The getter must have no input parameters and must return the exact same type of the intended field.

The setter method must start with “set” concatenated to the name of the field with its first letter capitalized. The getter must have one input parameter with the same type of the intended field. It must be a void method.

Figure 149 – JAX Class Attributes (zero or one)



```java
protected long quantity;

/**
 * field getters and setters
 */
public long getQuantity() {
    return this.quantity;
}
public void setQuantity(long element) {

    this.quantity = element;
}
```

Source: Created by the author

For **multiplicities 0..* and 1..***, the field must match be a java list, an example of list declaration is present in Figure 150.

The getter method must start with "get" concatenated to the name of the field with its first letter capitalized. The getter must have no input parameters and must return the list.

The setter method must start with “set” concatenated to the name of the field with its first letter capitalized. The getter must have one input list parameter of the intended field. It must be a void method.

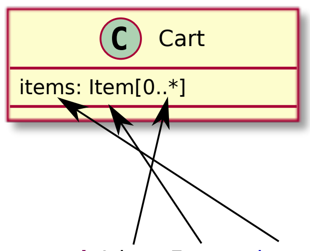
It is also recommended to add list management operations, however they are optional.

E.4.3.2 Class Associations

Classes associations are declared exactly as attributes, it is important to identify the notation correctly to find the names, multiplicity and types.

Similarly to class attributes, they must follow the multiplicity rule (whether to use lists or not). Examples of class associations are visible in Figure 151.

Figure 150 – JAX Class Attributes (Lists)



```
protected List<Item> items = null;

  /**
   * field getters and setters
   */
  public List<Item> getItems() {
      if (this.items == null) {
          this.items = new List<Item>();
      }
      return this.items;
  }

  public void setItems(List<Item> element) {
      this.items = element;
  }
```

Source: Created by the author

E.4.3.3 Class Compositions

Examples of class compositions are shown in Figure 152 In Java, there is no significant difference in declaring compositions and associations.

During this study diamond representing the composition may be considered irrelevant and the developer should simply follow the instruction for associations.

E.4.4 Other Features

JAX is implemented by different frameworks. In the case of CXF, JAX classes can be used to completely generate the associated XSD specification. Compilers capable of generating JAX classes from XSD is also available, allowing the developers to perform a round-trip engineering process.

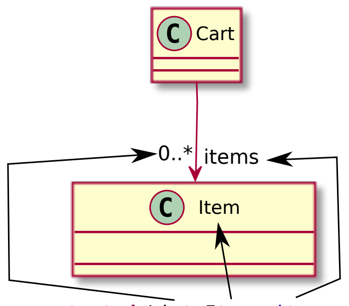
E.4.5 Testing the Result

The compilers used throughout the study include test case support to verify whether the generated code complies to the requirements.

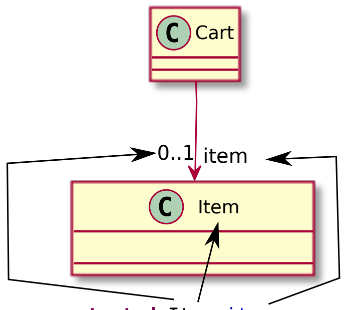
E.4.6 JAX Execution Workflow

It is recommended to follow this execution while working on the provided exercises.

Figure 151 – JAX Class Associations



```
protected List<Item> items = null;

   /**
    * field getters and setters
    */
   public List<Item> getItems() {
       if (this.items == null) {
           this.items = new List<Item>();
       }
       return this.items;
   }

   public void setItems(List<Item> element) {
       this.items = element;
   }
```

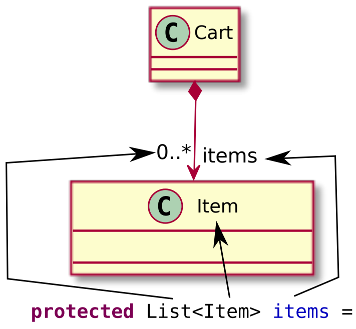


```
protected Item item;

   /**
    * field getters and setters
    */
   public Item getItem() {
       return this.item;
   }

   public void setItem(Item element) {
       this.items = element;
   }
```

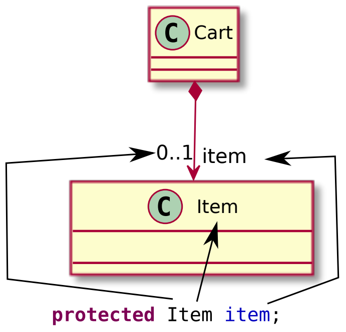
Source: Created by the author

1. Run “Record starting time”;
2. Run exercise test case (halt if test case succeeds);
3. Edit Java code according to diagram;
4. Return to item 2.

Figure 152 – JAX Class Compositions



```
protected List<Item> items = null;

  /**
   * field getters and setters
   */
  public List<Item> getItems() {
      if (this.items == null) {
          this.items = new List<Item>();
      }
      return this.items;
  }

  public void setItems(List<Item> element) {
      this.items = element;
  }
```



```
protected Item item;

  /**
   * field getters and setters
   */
  public Item getItem() {
      return this.item;
  }

  public void setItem(Item element) {
      this.items = element;
  }
```

Source: Created by the author

ICMC USP
SÃO CARLOS