UNIVERSITÄT PADERBORN

Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik
Zukunftsmeile 1
33102 Paderborn

Solving Heterogeneity for a Successful Service Market

PhD Thesis

Submitted to the Department of Computer Science
in Partial Fulfillment of the Requirements for the
Degree of

Doctor of Natural Science
(Dr. rer. nat.)

by

Svetlana Arifulina, M.Sc.

Thesis Supervisors:
Prof. Dr. Gregor Engels
and
Jun. Prof. Dr. Heiko Hamann

Frankfurt am Main, February 5, 2017

Abstract

This PhD thesis is written in the context of a new software development paradigm of the future called On-The-Fly Computing (OTF Computing). It is based on the idea of specialized service markets called On-The-Fly (OTF) markets. OTF markets have different properties and participants in OTF markets use different modeling techniques to perform the activity of service engineering. Such differences result in heterogeneity in OTF markets and complicate the execution of automated market operations like service matching as service specifications of different market actors cannot be automatically compared with each other. This PhD thesis proposes a solution to cope with the mentioned heterogeneity to foster the success of OTF markets and the OTF Computing paradigm.

In order to achieve the comparability of specifications in an OTF market, a formal intermediate representation called core language is introduced. Automated market operations for an OTF market are defined on a core language that optimally supports the execution of these operations in this market. This approach spares the effort for the definition of market operations and increases their quality.

The first contribution of this PhD thesis is the approach Language OPTimizer (LOPT), which supports the systematic design of a service specification language optimal for the execution of automated market operations in an OTF market. LOPT uses a comprehensive core language covering various structural, behavioral, and non-functional service properties. LOPT performs a configuration of this language based on formalized market properties and a knowledge base containing the configuration expertise. As a result, an optimal core language for an OTF market is obtained automatically.

The second contribution of this PhD thesis is the application of the Model Transformations By-Example (MTBE) technique to define transformations from proprietary specification languages of market actors to the optimal core language. The transformations are defined in a user-friendly manner, i.e., without a need of market actors to have any knowledge in the language design. The proposed approach mtbe generates transformations based on example mappings between concrete specifications in both languages. The derivation approach applies the idea of genetic algorithms having rich genetic operators, which allow the effective and efficient exploration of the solution space of possible model transformations.

Zusammenfassung

Diese Dissertation ist im Kontext eines neuen Paradigmas im Software Engineering mit dem Namen On-The-Fly Computing (OTF Computing) entstanden. OTF Computing basiert auf der Idee von spezialisierten On-The-Fly (OTF) Märkten. OTF Märkte haben unterschiedliche Eigenschaften und die Marktakteure in diesen Märkten benutzen verschiedene Modellierungstechniken für das Service Engineering. Diese Unterschiede resultieren in Heterogenität und erschweren deshalb die Ausführung von automatisierten Marktoperationen, da Servicebeschreibungen von verschiedenen Marktakteuren nicht automatisch miteinander verglichen werden können. Für das beschriebene Problem bietet diese Dissertation eine Lösung um einen erfolgreichen OTF Markt zu ermöglichen.

Für die Vergleichbarkeit von Servicebeschreibungen in einem OTF Markt wird eine formale Zwischenrepräsentation (Kernsprache) eingeführt. Die Marktoperationen werden auf Basis der Kernsprache definiert, die die optimale Ausführung der automatisierten Marktoperationen in dem gegebenen OTF Markt unterstützt. Dies reduziert den Aufwand die Operationen zu definieren bei gleichzeitiger Erhöhung der Definitionsqualität und ermöglicht neuen Marktakteuren einen einfacheren Marktzugang.

Der erste Beitrag dieser Dissertation ist der Ansatz Language OPTimizer (LOPT). Dieser ermöglicht das systematische Design einer Servicebeschreibungssprache, die die Ausführung von automatisierten Marktoperationen optimal unterstützt. LOPT nutzt als Basis eine reichhaltige Kernsprache, die strukturelle, verhaltensbezogene und nicht-funktionale Serviceeigenschaften beinhaltet. LOPT konfiguriert diese Sprache basierend auf formalisierten Markteigenschaften und einer Wissensbasis mit Konfigurationsexpertise, um eine optimale Kernsprache zu erstellen.

Der zweite Beitrag dieser Dissertation ist die Anwendung des Model Transformation By-Example Ansatzes um den Marktakteuren ohne Expertise im Sprachdesign Transformationen von ihren proprietären Sprachen in die optimale Kernsprache zu ermöglichen. Der beschriebene Ansatz mtbe generiert Transformationen auf Basis von Beispielabbildungen zwischen Servicebeschreibungen zweier Sprachen. Dabei wird die Idee genetischer Algorithmen angewendet.

Danksagung

An dieser Stelle möchte ich den vielen Menschen Danke sagen, die mich auf verschiedenste Art und Weise bei dieser Arbeit unterstützt haben.

In erster Linie bedanke ich mich bei meinem Doktorvater Gregor Engels. Er hat mich über mehrere Jahre zu diesem Ergebnis geführt, durch gute und schwierige Zeiten. Durch seine hohen Erwartungen habe ich mich weiterentwickelt und seine Unterstützung war ein wichtiger Baustein für meinen beruflichen Werdegang. Ich bedanke mich auch herzlich bei den Mitgliedern meiner Prüfungskommission: Heiko Hamann, Heike Wehrheim, Lorijn van Rooijen und Anthony Anjorin. Danke für Eure Zeit und die gründliche Betrachtung, die mir geholfen haben meine Doktorarbeit weiter zu verbessern.

Gregor hat mir außerdem ermöglicht im Sonderforschungsbereich 901 „On-The-Fly Computing" zu arbeiten. Bei der engen Zusammenarbeit mit Marie Christin Platenius im Teilprojekt B1 „Parametrisierte Servicespezifikationen" habe ich viel gelernt aber auch viel Spaß gehabt. Ein besonderer Dank geht an Wilhelm Schäfer, der erheblich zu der Qualität unserer wissenschaftlichen Ergebnisse und Publikationen beigetragen hat. Ich möchte mich außerdem für eine tolle Zusammenarbeit bei den weiteren ProfessorInnen und KollegInnen aus dem SFB 901 bedanken.

Während meiner Zeit als wissenschaftliche Mitarbeiterin habe ich mich sehr wohl in der AG Engels gefühlt. Meine Kolleginnen und Kollegen waren nett und hilfsbereit, was eine gute Zusammenarbeit und viele spannende Diskussionen ermöglicht hat. Ich danke Euch allen für die schöne Zeit! Für die besondere Unterstützung bei den Vorbereitungen auf meine Verteidigung möchte ich Marvin Grieger, Marie Christin Platenius und Simon Schwichtenberg danken.

Zusätzlich haben auch viele StudentInnen zu den Forschungsergebnissen meiner Dissertation beigetragen. Ich bedanke mich bei den TeilnehmerInnen der Projektgruppe „AppSolut", bei Kavitha Jagannath und Vahide Taherinajafabadi für ihre Masterarbeiten und bei allen SHKs des Teilprojektes B1.

Aus dem tiefsten Herzen bedanke ich mich bei den für mich wichtigsten Menschen: meiner Mutter Alevtina Jäger und meinem Partner Hendrik Schreiber. Sie glauben immer an mich und geben mir den Halt jeden Tag aufs Neue weiterzumachen. Ein lieber Dank geht auch an meine Freundinnen und Freunde, die auf diesem Weg die Daumen für mich gedrückt haben. Hier möchte ich insbesondere Galina Besova, Dominik Steenken, Marlene Rathgeber und Eugen Wiens erwähnen. Ein Dank für die Unterstützung geht auch an die Mitglieder der Familie Schreiber und ihre Partner. Ich schätze mich glücklich, Euch alle an meiner Seite haben zu dürfen.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

This PhD thesis is written in the scope of the Collaborative Research Centre 901 "On-The-Fly Computing" [151] supported by the German Research Foundation (DFG). The Collaborative Research Centre 901 aims at developing concepts and techniques for a new software development paradigm of the future called *On-The-Fly Computing (OTF Computing)*. OTF Computing is based on the idea of specialized service markets called *OTF markets*. Services are distributed over OTF markets according to the type of their functionality. Each OTF market contains services of a certain functionality type. Figure 1.1 illustrates an OTF market of `Tourism` as an example.

Market actors participating in an OTF market have the following types: *service providers*, *service requesters*, and *brokers*. Service providers offer services for usage or for sell. Service requesters request customized services satisfying certain requirements. Brokers bring together the requirements of service requesters and the services of service providers. Brokers have expertise in trading services in a certain OTF market, and serve as intermediaries between providers and requesters in the market of their expertise.

The main artifacts of an OTF market are *software services*, *service specifications*, and *requirements specifications*. Software services are the subject of trade in OTF markets. Software services (or services in this PhD thesis) represent software provided over the Internet and identified by a Uniform Resource Identifier. Figure 1.1 shows such example services as `Hotel reservation` or `PB city portal`. Service providers describe services in service specifications containing their certain functional and non-functional properties. Service requesters describe functional and non-functional requirements on a service to be discovered in requirements specifications.

Market operations defined for each OTF market are depicted in Figure 1.2. Market operations are performed on the artifacts of OTF markets and divided into manual and automatic.

`Specify a service` is the operation to describe functional and non-functional properties of a service with a certain specification language. This operation plays a central role in OTF markets as specifications serve as a basis for the automated market operations. Requesters perform this market operation to manually specify their requirements, while service providers use it to manually specify the properties of their services. The inheritance relation between ser-

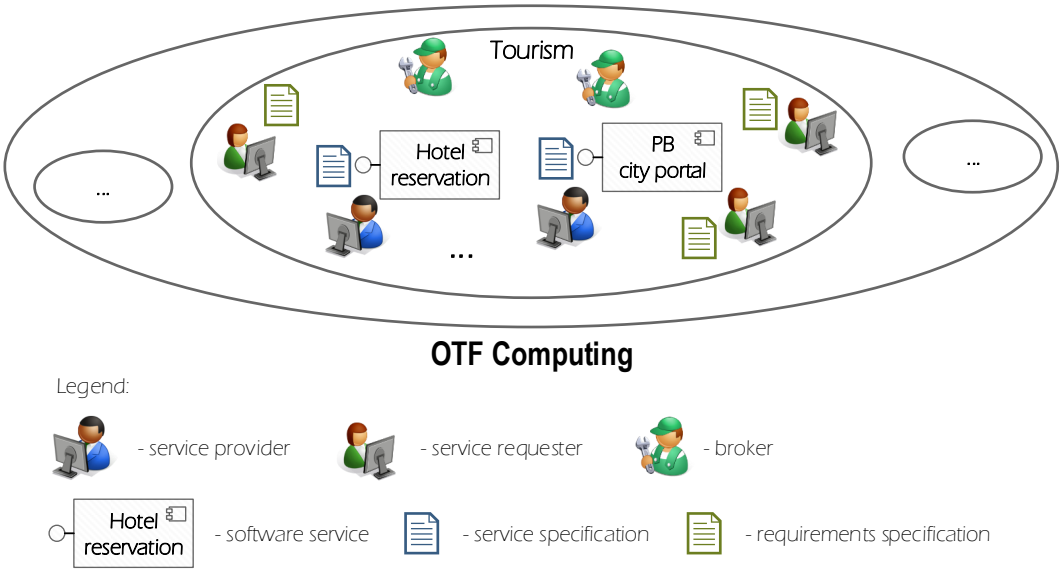


Figure 1.1: OTF markets

vice requesters and service providers depicted in Figure 1.2 means that service providers perform all operations of service requesters and their own in addition. As a result, either a service specification or a requirements specification is created. This PhD thesis considers service requesters with a technical expertise, who can understand and create formal service specifications. Requesters with no technical expertise are considered as a future work.

`Provide a service` is the operation of implementing and offering a service based on a certain service specification. Service providers perform this operation to offer services for sell in different OTF markets. These services are composed by brokers to build customized services required by requesters.

`Match a service` is the operation of comparing properties of a particular service with the given requirements based on their specifications. The goal is to calculate the degree of compliance of the service described by the service specification with the requirements described by the requirements specification. This operation is performed automatically using a special software called matcher, which is available in an OTF market.

`Discover a service` is the operation of filtering out those services, which fit to the requirements specified in the given requirements specification. This operation is automatically performed on requirements and service specifications by a broker, who uses the operation `Match a service` in the background.

`Compose a service` is the operation of composing existing services into a new service. A broker performs this operation automatically with the help of a special composition software. This operation uses the operation `Discover a service` based on requirements and service specifications. The operation produces a composed service and its specification as a result.

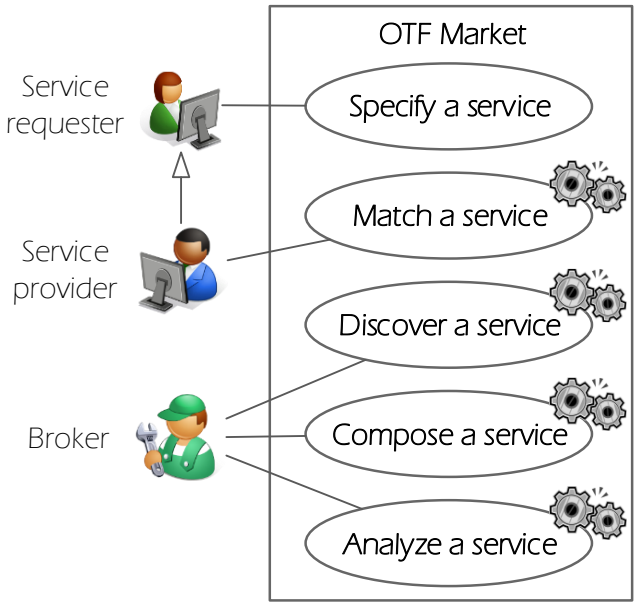


Figure 1.2: Market operations

`Analyze a service` is the operation of analyzing the quality of a composed service. The goal is to check its functional correctness and its non-functional properties based on given requirements specification and the service specifications of its constituent services. A broker performs this operation automatically by using different kinds of verifiers, analyzers, and simulators.

The challenges for OTF markets are presented in the following. Based on these challenges, problem statements for this PhD thesis are identified.

This PhD thesis starts with **Challenge 1: Efficient automated market operations**. In the software engineering, market operations described above are performed *manually* by large software houses. The innovative idea of OTF Computing is to automate the manual execution of these operations.

Software architects usually design a new software system. Software developers search for software components suitable to be reused for the given design and integrate them in a software system. This is analogous to performing the market operations `Discover a service`, `Match a service`, and `Compose a service`, which would be automated in an OTF market and, thus, facilitate the discovery and integration of software components. Then, developers and testers check the quality of the integrated system. This is analogous to performing the operation `Analyze a service`, which would be automated in an OTF market and, thus, increase the quality of the resulting software system.

Market operations have to be performed as efficient as possible in order to cope with the following market characteristics. On the one hand, market operations have to be performed on a high amount of different artifacts created in different modeling languages. On the other hand, the results of the market operations

computed earlier can be barely reused. Since an OTF market is highly dynamic, new services having a better quality could be used instead of the old ones, and the services found before can become unavailable.

Challenge 2: Formal comprehensive service specifications is the next challenge in OTF markets. The automated market operations listed in Figure 1.2 can be performed on service specifications only. Since implementation details of services are encapsulated, relevant information about provided or searched services is abstracted in their service specifications.

Service specifications in OTF markets have to be comprehensive. Comprehensive specifications contain a description of functional properties of a service, like its structure or its behavior, as well as its non-functional properties, like performance, privacy, or price. Performing market operations on comprehensive specifications yields more reliable results because such specifications cover more semantics of a service in comparison to simple specifications, which usually describe structural properties of services only. Considering semantic information about services increases the confidence in results of the market operations.

Formal comprehensive service specifications are a prerequisite for efficient automated market operations. Comprehensive specifications can be written in different formal and informal notations. For example, an informal notation in the form of a natural language can be used. However, it would complicate the efficient execution of the automated market operations because informal specifications are ambiguous and do not have a well-defined semantics.

Since formal specifications require a formal definition of the syntax of their specification languages, specification languages considered in OTF markets must have a formal syntax definition. A formal semantics definition for the specification languages is needed as well. For example, a formal definition of behavioral semantics of the underlying service specifications is required in OTF markets for performing the operation `Analyze a service`.

Challenge 3: Heterogeneity is caused by the fact that service providers and service requesters in OTF markets use different notations to describe their services and requirements. They choose different general-purpose and domain-specific specification languages and their dialects, which may be either established or may be chosen for other reasons, e.g., legacy notations. As a result, OTF Computing lacks a common cross-market specification standard as all market actors follow their own standards. In order to overcome the lack of standardization, specifications written in different languages have to be unified so that they become comparable for performing the automated market operations.

To tackle the lack of standardization, OTF Computing could enforce its own specification standard. For example, Web Service Description Language (WSDL) [28] being a standard for the SOA architecture [4] could be used. As a consequence, all market actors would have to provide specifications in this standard, in order to participate in OTF markets. Such a decision would introduce a significant market barrier [127] for market actors, because they would have to put a significant effort to enter the market. Furthermore, such a specification

standard would not consider specifics of single OTF markets. All these reasons would reduce the acceptance of OTF Computing. Thus, the enforcement of a specification standard is not desirable. As a result, other techniques to cope with the heterogeneity of specifications in OTF markets have to be developed.

Heterogeneity in OTF markets leads to **Challenge 4: User-friendly language transformation**. To perform the automated market operations, heterogeneous specifications have to be comparable. This requires advanced language design techniques like language transformations for heterogeneous specifications. For example, market actors would have to transform their specification languages, if the market operations are not defined for these languages.

However, market actors are usually only users of specification languages but not their designers. Thus, they lack the expertise in language design, which is required to create the necessary artifacts and to perform the language transformation. Therefore, the language transformation for market actors has to be user-friendly, i.e., market actors should be able to perform language transformations in a simple way adapted to their expertise. This would allow them to enter OTF markets with their own specifications and to transform them into other specification languages. Such a language transformation has to preserve as much semantics of the transformed languages as possible. The semantics preservation would enable to transfer as much knowledge about a service described in its service specification as possible.

Challenge 5: Market acceptance concerns the acceptance of OTF markets. If the acceptance of OTF markets is high, many market actors would actively use them, thus, adopting the vision of OTF Computing.

Market actors in an OTF market provide and use services of the same kind of functionality. Thus, the corresponding OTF market has to support the execution of the automated market operations in a way specific for this market. These market operations have to deliver reliable results to market actors to support their competing objectives. Based on these results, service providers should be able to sell as many services as possible to the highest possible revenue, while service requesters should get the most suitable service for their requirements with the lowest possible cost. In addition, the entrance in OTF markets has to be facilitated. The market success of market actors would increase the acceptance of OTF markets and, as a result, of the OTF Computing paradigm.

As mentioned above, designing service markets like OTF markets involves different challenges. This PhD thesis focuses on solving heterogeneity in OTF markets, in order to contribute to their success. Based on the challenges presented above, problems motivating this PhD thesis were identified. The problem statements are discussed in detail in the following Section 1.2. Later on, this PhD thesis proposes concepts and techniques for solving them. These concepts and techniques contribute to the success and acceptance of OTF markets by market actors.

1.2 Problem Statements

In this chapter, two problem statements are derived from the challenges presented in Section 1.1. Figure 1.3 shows the connection between the challenges and the derived problem statements. Detailed descriptions of the problem statements are presented in Sections 1.2.1 and 1.2.2.

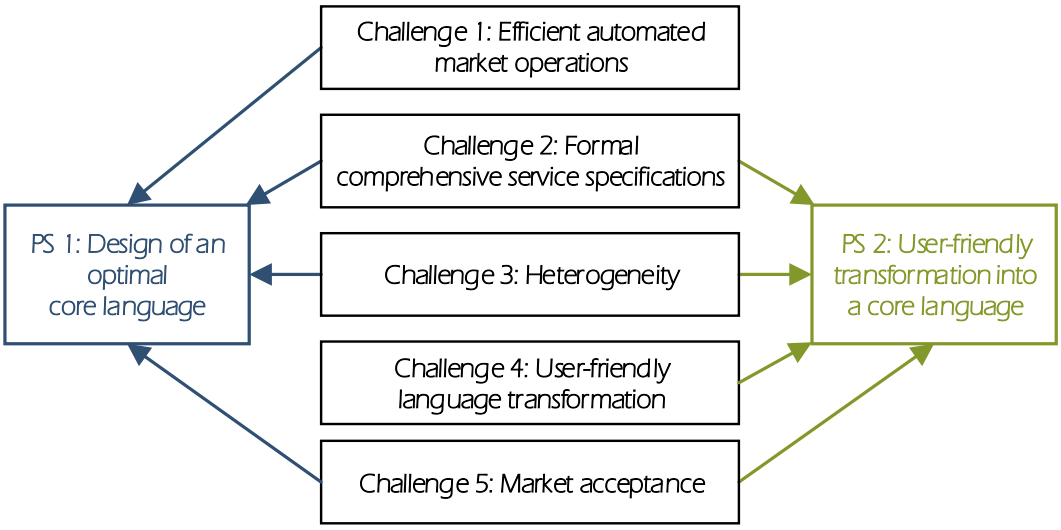


Figure 1.3: Problem statements derived from the challenges in Section 1.1

1.2.1 PS 1: Design of an Optimal Core Language

According to **Challenge 1**, the market operations in an OTF market have to be automated and efficient. For that, as described in **Challenge 2**, service specifications serving as a basis for these automated operations have to be formal and comprehensive. Regarding to **Challenge 3**, these formal comprehensive service specifications are heterogeneous. Additionally, **Challenge 5** states that the automated market operations have to be supported in a market-specific way. Therefore, a mechanism is needed to foster the comparability of the service specifications, in order to be able to perform the automated market operations in an OTF market. Furthermore, the execution of automated market operations have to consider the specifics of the OTF market, in order to increase the success of its market actors.

In order to address these challenges, one solution would be to provide market operations for each pair of all existing service specification languages and their dialects. Assume that one or several responsible market actors have to produce all those operation definitions. This solution would require these actors to have access to all these languages and to have the expertise in them. However, existing specification languages used by market actors are proprietary and diverse.

Market actors would most probably provide no access to them. Furthermore, it is also infeasible to assume that some market actors could have the expertise in all these languages.

An alternative and more feasible solution is to provide a common intermediate representation, for which automated market operations can be defined and performed. This idea is similar to the theory of compiler construction, in which an intermediate representation was proposed to cope with the large amount of existing programming languages [2]. The intermediate representation for an OTF market is called a *core language*. The core language abstracts from heterogeneous service specifications by containing only those language constructs, which are relevant for performing the automated operations in the given OTF market. Market actors have to transform their languages into the core language and, then, can start using the automated market operations. One transformation per language would be needed with the expertise in the core language only.

Existing approaches propose to design core languages according to the following main design principles.

1. **Core language as an intersection:** Approaches like DUALLY [89] propose to construct a core language as an intersection of language constructs of languages, whose comparability is aimed. The considered languages must have a formal syntax definition. Their semantics is formalized by mapping their language constructs to the language constructs of the DUALLY core language. The main advantage of this design principle is that the core language serves as a semantic bridge, which allows to transform specifications from one language into another. The main disadvantage is that such a core language might contain a small amount of language constructs if certain languages have very few language constructs in common.

   According to **Challenge 3**, service requesters and service providers use different specification languages with different expressiveness. Therefore, if the core language was an intersection, then it would be barely expressive because of the diversity of language constructs in these specification languages. As a consequence, results of the market operations based on specifications in such a core language would be unreliable because very few service properties would be considered.

2. **Core language as a union:** Approaches like UnSCom [112] propose a core language for specification of software services, which is a union of existing specification languages. Analogously to DUALLY, the considered languages have a formal syntax definition, while the semantics of the languages is formalized by mappings to the UnSCom language. The main advantage of this design principle is that the core language is comprehensive and includes various kinds of service properties. Thus, it can be used to create highly comprehensive specifications. However, the main disadvantage is that operations cannot be performed efficiently on specifications

written in such a language because processing highly comprehensive specifications takes much computation time.

If the core language of the OTF market was a union, then a very expressive core language would be produced because of the diversity of specification languages existing in an OTF market. Such a core language would allow to create highly comprehensive service specifications. However, this high comprehensiveness would hamper the efficiency of the automated market operations. As a consequence, results of the market operations would be very reliable but the market operations could not be performed efficiently.

3 Core language as a structure in between: A core language as a structure in between is a language, which is more expressive than an intersection of arbitrary languages but less comprehensive than their union. Such a language can be customized for the needs of the given scenario and the operations defined on its specifications.

A structure in between would be the most appropriate design principle for core languages in OTF Computing. One advantage is that the core language could have any possible language constructs beyond the intersection but within the union. Comprehensive service specifications describing service properties really needed for the automated market operations can be created. Based on the choice of language constructs in the core language, service specifications can be adapted to the market specifics. This would allow to design a core language, which is exactly suitable for the automated market operations thus increasing the acceptance of the corresponding OTF market.

The choice of the design principle proposing the core language as a structure in between leads to the next question for this PhD thesis: Which language constructs should be a part of the core language for an OTF market so that its operations can be optimally executed? In order to answer this question, the connection between the automated market operations and the core language has to be understood. The more suitable is the core language for these operations, the more reliable the results of their application are and the more efficient their application is. Thus, each OTF market requires a core language, which is *optimal* for the automated market operations of this OTF market.

Therefore, the first problem being studied as a part of this PhD thesis is:

How to design a core language for service specification optimal for efficient automated market operations in an OTF market?

The following research questions will be answered in the scope of this problem:

1. What is a systematic process for the design of an optimal core language in an OTF market?

2. What measurable notion of language optimality can be used for the design and evaluation of an optimal core language?
3. How can properties of an OTF market be leveraged for the design of an optimal core language?
4. How are conflicting requirements considered for the design of an optimal core language?
5. How to increase the reuse of existing languages for the design of an optimal core language?

1.2.2 PS 2: User-Friendly Transformation into a Core Language

According to the problem statement **PS 1**, the automated market operations in an OTF market are defined for an intermediate representation called a core language. A core language is designed to optimally support these automated market operations. According to **Challenge 2** and **Challenge 3** (see Figure 1.3), market actors specify their services and requirements using arbitrary formal and comprehensive specification languages. Based on **Challenge 4**, specifications in these languages have to be transformed into the core language in a user-friendly manner, in order to perform the automated market operations on them. As stated in **Challenge 5**, the specifics of each OTF market have to be considered, in order to improve the entrance of market actors in OTF markets.

One possibility to conduct such a transformation is to engage an expert in a given OTF market, for example, a broker. The broker performs the automated market operations using a software available in the market, which performs different market operations and operates on a certain optimal core language. However, in this scenario, the broker should be able to transform arbitrary existing specification languages used by market actors. According to **PS 1**, that would lead to an infeasible effort for the broker due to the missing access to proprietary languages of service providers and due to the necessary expertise in knowing diverse specification languages.

Another alternative possibility is to let the market actors transform their specifications themselves. In this scenario, a service provider or a service requester would only need to learn the optimal core language used by the broker. Both market actors are able to do that, because this PhD thesis considers service requesters and service providers, who have a necessary technical expertise to understand and write formal service specifications. A technically inexperienced requester is considered as a future work. This would also result in significantly less transformation effort in comparison to the previous scenario, in which a broker needed to learn all existing languages in an OTF market.

As stated in **Challenge 4**, the language transformation for market actors in the OTF market has to be designed user-friendly because market actors lack the

necessary expertise in language design. This means that market actors cannot work on language definitions directly, but they have expertise in creating formal specifications by using editors of their languages. Market actors usually learn the semantics of a language based on its informal language definition and a set of example specifications. These example specifications represent typical language constructs of the given language and cover its most probable usage contexts. Thus, user-friendly techniques have to be provided for market actors to transform their service specifications into the core language.

According to **Challenge 2**, service specifications must have a formally defined syntax and semantics. This implies that these formal definitions have to be considered during the user-friendly language transformation in the OTF market. Based on **Challenge 5**, the market-specifics have to be considered. In this case, each language used in an OTF market is transformed into the optimal core language for this market.

Therefore, the second problem being studied as a part of this PhD thesis is:

How to perform a semantic-preserving transformation of specification languages in a user-friendly way?

The following research questions will be answered in the scope of this problem:

1. How can specification languages be transformed by a user having no expertise in language design?
2. For which specification languages is the user-friendly transformation applicable?
3. What is a suitable representation of the formal semantics of specification languages for the user-friendly transformation?
4. How can the quality of the language transformation be measured for its evaluation?

1.3 Tour of the PhD Thesis

The key note of this PhD thesis is solving the two presented problem statements. At the beginning, the related work is analyzed for relevant approaches, which can be applied for solving these problem statements. Based on the knowledge gained from the investigated approaches, requirements on the solution are collected. Afterwards, the proposed solution is illustrated by presenting its parts solving each problem statement in a separate chapter. The PhD thesis is concluded by summing up the results and by giving an outlook for future work.

Chapter 2 investigates existing approaches related to the problem statements of this PhD thesis presented in Section 1.2. Existing approaches are grouped

into two areas corresponding to the problem statements. A systematic literature review is conducted for each area. The reviews are presented correspondingly in Sections 2.1 and 2.2.

Chapter 3 presents requirements on a solution for the problem statements and an overview of the solution. The requirements listed in Section 3.1 are derived from the problems statements described in Section 1.2 and related approaches described in Chapter 2. Based on these requirements, an overview of a solution satisfying them is illustrated in Section 3.2. The solution consists of two approaches, each addressing one of the presented problem statements.

Chapter 4 introduces the approach Language Optimizer (LOpt) for the design of an optimal core language. The chapter starts by Section 4.1 defining the notion of a core language and its optimality. Section 4.2 presents the approach LOpt and its parts including the comprehensive core language and a configuration approach for its optimization regarding a given OTF market. Section 4.3 describes the application and evaluation of the presented approach.

Chapter 5 presents the approach Model Transformation By-Example (mtbe) for the user-friendly language transformation using genetic algorithms. This chapter begins by defining the main notions necessary for the design of the approach. Section 5.1 explains the approach and its parts, where each genetic operator is described in a corresponding subsection. Section 5.2 shows the application and evaluation of the presented approach.

Chapter 6 presents the summary and future work for this PhD thesis. Section 6.1 summarizes the results of this PhD thesis and points out their benefits and limitations. The final Section 6.2 gives an outlook for future work that can be conducted to improve and extend the presented approaches.

1.4 Overview of Publications

On the topic of this PhD thesis, the author and her colleagues published the following papers at various international conferences. All papers were peer-reviewed and presented by one of the authors at the corresponding conference.

[9] tackles the topic of ensuring the quality of formal specifications by considering the behavioral semantics of specification languages. In this paper, the concept of test coverage is applied to the approach of Test-Driven Semantics Specification, which is used for graph-based specification of semantics. The concepts of this paper are used as preliminary work for the formal definition of behavioral semantics in approaches developed in this PhD thesis.

[6] presents the idea of coping with the heterogeneity in OTF markets. The paper introduces a framework consisting of a comprehensive core language, whose parts can be mapped onto different existing languages, in order to enable their integration for the automated market operations like service matching. The concepts of this paper serve as a basis for the solution idea of this PhD thesis.

[8] introduces the market-optimized service specification language and match-

ing. This paper describes the automated approach for the configuration of a language-matcher pair for an OTF market. The configuration is performed by customizing a comprehensive, modular service specification language and its matching for market properties. As a result, the needs of service providers and requesters is suited the best that increases the acceptance of the OTF market. The concepts of this paper represent the solution approach of this PhD thesis.

[7] extends the idea of [8] by designing a market-optimized service specification language for service composition. Such a language allows the broker to create market-optimized specifications of composition templates and market-optimized requirements specifications. Afterwards, the broker uses a market-optimized service matching to build the composed service. The concepts of this paper are elaborated as an extension for this PhD thesis.

[117] tackles the topic of a model-driven definition of matching processes by a broker in an OTF market. This paper proposes a framework that supports a broker in reusing existing matching approaches and combining them in executable matching processes. The concepts of this paper are used to design and evaluate the solution approach of this PhD thesis.

[116] describes the matching of incomplete service specifications covering the signatures and privacy policies of services. This paper shows how fuzzy matching can be applied to such specifications. The concepts of this paper are used to design and evaluate the solution approach of this PhD thesis.

[10] presents the tool support for the solution approach of this PhD thesis. This paper describes the framework SeSAME for specification of services, requirements on services and service compositions, for matching of service and requirements specifications, and for analyzing the resulting composed service. This tool support was used for proof-of-concept and evaluation of the solution approach of this PhD thesis.

2 Related Work

In this chapter, existing approaches are investigated, which address the problem statements of this PhD thesis presented in Section 1.2.

In this PhD thesis, these related approaches are investigated in a systematic literature review, which procedure is oriented on the "Guidelines for performing systematic literature reviews in software engineering" by Kitchenham et al. [81, 82]. One benefit of a systematic literature review is the fact that such review is unbiased and repeatable to a certain extent. Another benefit of such review is a well-defined methodology that is used to conduct it. Thereby, due to the mentioned benefits, systematic literature reviews have higher research validity compared to unsystematic techniques for investigating related work.

Each systematic literature review has to be documented in the form of a protocol, which allows other researchers to repeat the described review process. The protocol describes how the review was conducted, what research questions were answered, and what methods were used. In the protocol, a search strategy, explicit inclusion and exclusion criteria for literature sources and an evaluation of the found literature sources have to be defined and documented.

Existing approaches related to this PhD thesis are grouped into two areas correspondingly to the two problem statements presented in Sections 1.2.1 and 1.2.2. For each area of related approaches, a systematic literature review is conducted, a description of the review protocol is given followed by an overview of the review results and their evaluation. The reviews were conducted during the time period from August to September 2014. All approaches appeared later than September 2014 are not considered. The reviews are presented in Sections 2.1-2.2.

2.1 Related Work on Design of an Optimal Core Language

This section presents the systematic literature review of approaches addressing the problem of the design of an optimal core language. The review protocol is described in Section 2.1.1. Section 2.1.2 presents the set of literature sources obtained according to the presented protocol followed by their evaluation.

2.1.1 Review Protocol

The protocol of this systematic literature review starts with listing *research questions*, which existing approaches considered in this review treat. These research questions generalize the research questions from Section 1.2.1 (the generalization is performed, in order to find more related approaches and to gain knowledge from more general approaches in this area):

1. What techniques exist to design a core language? What techniques exist to design an optimal specification language?
2. What measurable notions of language quality or language optimality exist for the design and evaluation of core languages or specification languages?
3. What knowledge can be leveraged for the design of a core language, a service specification language, or an optimal specification language?
4. How are conflicting requirements considered for the design of a core language, a service specification language, or an optimal specification language?
5. What service specification languages exist? How to increase the reuse of existing languages in a core language, a service specification language, or an optimal specification language?

In this area, systematic approaches for language design and optimization are investigated. Interesting are those approaches, which are proposed for the design of core languages but also for the optimization of specification languages. Those approaches are considered, which provide methods to compare several specification languages and determine the most suitable one for a certain purpose. For the design and optimization, measurable notions of language quality or language optimality are considered for specification languages, in general, and for core languages, in particular. Consideration and resolution of conflicting requirements for different kinds of specification languages is also of interest for this review. Existing service specification languages and different possibilities of their reuse during the language design are investigated as well.

The description of research questions is followed by a description of *search strategies*. The first search strategy, which was used for this category of related approaches, is the *broad search using keywords*. The search was conducted in digital libraries, conference proceedings, and the Internet. Searched digital libraries were Google Scholar [58], ACM Digital Library [1], IEEE Xplore [68], and Springer Link [135]. The considered conference proceedings were from the International Conference on Service-Oriented Computing (ICSOC) from 2008 to 2014 and the International Conference on Web Services (ICWS) from 2009 to 2013 as these conferences play a central role in the community of service-oriented computing. For the Internet search, the search machine of Google [57] was used.

The second search strategy, called *in-depth search*, was applied to the literature sources found based on the broad search strategy. According to this search

Table 2.1: Keywords for the broad search

Keywords type	Keywords
Subject	Description, description language, description standard, language, model, modeling language, specification, specification language, specification standard.
Subject property	Architecture, best, common, component, configurable, core, customized, extensible, format, good, ideal, intermediate, interoperability, modular, optimal, optimum, perfect, quality, service, structure, tailored.
Action	Aggregate/aggregation, build/building, combine/combination, compose/composition, configure/configuration, consolidate/consolidation, construct/construction, create/creation, define/definition, design, develop/development, engineer/engineering, environment, evaluate/evaluation, framework, integrate/integration, merge, optimize/optimization, plan/planning, unify/union.

strategy, references of the found literature sources are investigated for being relevant to the conducted review. The same inclusion and exclusion criteria listed below were used for both presented search strategies.

For the broad search, *keywords* used for search queries have to be determined. Keywords were derived from the research questions presented above and refined by related terms from the area of language engineering. For each of these keywords, synonyms and alternative spellings were collected using the online thesaurus [115]. The final set of keywords is presented in Table 2.1.

The keywords are grouped into three types: `Subject`, `Subject property`, and `Action`. The first type `Subject` refers to a subject, to which existing approaches apply, e.g., `specification language`. The second type `Subject property` refers to a certain characteristic of the subject, which existing approaches consider, e.g., `configurable`. The third type `Action` refers to a certain action, which existing approaches apply to the subject, e.g., `compose`. A keyword for one search query was set either as a combination of the keywords of the types `Subject` and `Subject property` or as a combination of the keywords of the types `Subject` and `Action`. For example, `configurable specification language` and `specification language composition` were used as search queries.

After keywords for the search queries have been defined, *inclusion and exclusion criteria* for found literature sources have to be stated. Inclusion criteria describe properties of literature sources to be considered in the review, while exclusion criteria describe properties of literature sources to be omitted.

The inclusion criteria for this review are presented below:

1. Only literature sources in the area of computer science.
   Since this PhD thesis focuses on the research in the area of computer science, only sources presenting research results in this area are considered.
2. Only literature sources written in English.
   Since this PhD thesis is written for an international reader and, therefore, in English, all presented approaches have to be available in English, too.
3. Only literature sources with the length of at least 6 pages.
   Since profound existing approaches providing enough detail on the solution are interesting for this review, the size of 6 pages corresponding to the minimum size of long papers at most conferences are considered. All literature sources, which have less pages, mostly lack either an evaluation part or enough details on the presented approach.
4. Only literature sources available for the University of Paderborn for free.
   Since the University of Paderborn gains an access to a broad range of articles in the digital libraries mentioned above, literature sources requiring additional payment are not considered in this review.
5. Only literature sources covering keywords from two combined keywords types (both `Subject` + `Subject property` or both `Subject` + `Action`).
   Since literature sources covering the combination of all three types of keywords are hardly possible to find and a search with only one type of keywords is too general for the stated research questions, search queries containing two types of keywords seemed to be feasible for this review.

Exclusion criteria for this review are described in the following:

1. Literature sources dealing with programming languages.
   Searching for the keywords presented above also delivers literature sources describing design techniques or different properties of programming languages instead of modeling languages. Since the focus of this PhD thesis lies at specification languages, especially because the specification of services and not their implementation is investigated, literature sources dealing with programming languages are omitted for this review.
2. Literature sources dealing with natural languages or computational linguistics.
   Another set of found literature source describe language engineering techniques from the area of computational linguistics applicable to natural languages. Since the focus of this PhD thesis are formal languages instead of natural languages, the latter literature sources are omitted.
3. Literature sources presenting a new modeling language only.
   Such literature sources are mostly limited to describing a certain (domain-specific) modeling language without telling much about its development. The reason for that would be either the use of a standard language engineering technique or the lack of a systematic process. Since this PhD thesis focuses on systematic design techniques for specification languages, this kind of literature sources is considered to be out of scope of this review.

4. Literature sources about software architectures.
   During the search for the keyword combination like `specification language architecture`, literature sources regarding software architectures were found. Since this topic is out of scope of this PhD thesis, such literature sources are omitted in this review.
5. Literature sources tackling visual modeling languages from the usability point of view.
   According to the keywords, literature sources tackling design techniques for visual languages with the emphasis on usability were found as well. Since the focus of this PhD thesis are specification language and their optimization for automatic operations like service matching, the usability of such languages is out of scope.

2.1.2 Evaluation of the Review Results

Section 2.1.2.1 presents literature sources obtained by following the protocol described in Section 2.1.1.

2.1.2.1 Overview of the Review Results

Table 2.2 gives an overview of the literature sources found based on the broad search strategy using the keywords from Table 2.1 and the literature sources obtained based on the in-depth search strategy. The found literature sources are grouped into several categories, in which approaches using similar techniques are collected.

Table 2.2: Categories of literature sources found in the review

Category	Reference
Design of core languages	[12, 60, 90, 99, 129, 156]
Language architectures and design approaches	[3, 13, 17, 33, 34, 35, 44, 54, 55, 56, 76, 96, 100, 113, 121, 128, 131, 134, 150, 166]
Language composition	[19, 25, 37, 42, 48, 52, 61, 74, 84, 98, 118, 120, 139, 143, 159, 161]
Language quality	[27, 62, 65, 85, 94, 114]
Service specification languages	[16, 18, 103, 105, 112, 148]

2.1.2.2 Evaluation Summary

Existing approaches investigated in this systematic literature review answer the research questions listed in Section 2.1.1 as follows:

1. What techniques exist to design a core language?

Various approaches for the design of a core language are listed in Table 2.2, category *Design of core languages*. The presented core languages are mostly

defined for a family of specification languages. For example, DUALLY [90] introduces a core language for *interoperability of architectural specifications* or UEML [156] a core language for *interoperability of enterprise specifications*.

The investigated approaches define the content of a corresponding core language as a set of language constructs common for a considered family of specification languages. For example, the DUALLY core language contains all common language constructs among different architecture description languages. Domain-specific language constructs are omitted in the core language.

The approaches also define how to transform existing languages into the core language and how to transform specifications in the core language into other formats used to perform further tasks, e.g., a certain type of analysis. For example, KLAPER [60] is a core language for *performance and reliability analysis of component specifications*. KLAPER contains only language constructs modeling properties relevant for the analysis, and specifications are transformed into KLAPER specifications, based on which analysis models are generated.

As a summary, the presented core languages are too specific for their domains to be applicable to the problem statements of this PhD thesis. Furthermore, in comparison to the investigated approaches, a core language as a structure in between is a feasible solution for the OTF market as explained in Section 1.2.1. *However, the presented design principles for core languages can be followed for the OTF market as well. These design principles include designing a family of languages, and restricting a core language to obtain a domain-specific language, which models only properties relevant for the chosen domain.*

What techniques exist to design an optimal specification language?

The category *Language architectures and design approaches* in Table 2.2 presents approaches introducing different language architectures and language design approaches. None of these approaches tackles the development of an optimal specification language directly. However, they propose design principles for a good language architecture and methods to configure languages with respect to a certain usage context. A set of properties for a good language design can be derived based on investigated approaches. For example, Karsai et al. [76] propose guidelines that aim to improve the design of domain-specific languages (DSLs). Some example guidelines are *to reuse existing languages using only those domain concepts relevant for the purpose of a language*, *to avoid redundancies*, and *to enable modularity*.

The property of modularity is desired for the language reuse and its better maintenance using a corresponding extension mechanism. For example, Akehurst et al. [3] suggest *a modular specification* of the syntax of the Object Constraint Language (OCL). The authors define that a language module contains a concrete syntax definition, an abstract syntax definition and a mapping from the concrete to the abstract syntax. *An extension mechanism* was proposed enabling to unify existing OCL extensions and to reuse the defined modules. Another example is the approach of Bae et al. [13] proposing to *modularize* the UML language defined as a metamodel into a set of submetamodels, in order to

cope with the complexity of UML for learning and developing editors. For that, the approach UMLSlicer was introduced for extracting submetamodels corresponding to different UML diagram types. A verification of interdependencies and consistency is also supported for the resulting submetamodels.

Further property is the definition of viewpoints, which enable different users work with different parts of a language. For example, Fan et al. [44] suggests a framework supporting a process *to derive a new architecture description language* from scratch or based on a reuse of existing languages. This process includes specifying viewpoints and designing the relations between the created languages and their viewpoints. Another approach of Goldschmidt et al. [56] presents *a taxonomy of view-based modeling* from the tool perspective. The authors explain the definitions of view, view type and view point, their relations, and a classification of view types, views, and editor capabilities, which influence the way of how modelers work with views. This classification can be used for a comparison or a new development of view-based specification approaches.

The principle of reuse of existing specification languages results in an efficient and less error-prone way to design a language. For example, Zschaler et al. [166] consider *families of domain-specific languages (DSLs)*, which are sets of languages having core constructs and well-defined variations to serve similar application domains. Introducing a family of DSLs fosters *a systematic reuse* of their language constructs and *an explicit systematic way to specify variability* between the DSLs. The variability supports adding and removing language constructs and integrating the DSLs with other existing languages.

The property of orthogonality aims at avoiding redundancies in a language and, thus, increasing its maintainability. This property is described in the design principles by Paige et al. [113]. Orthogonality requires that only one way exists to describe a concept in a language, in order to avoid redundancies. As a result, it fosters a better understandability and maintainability of a language.

In order to customize a specification language for a slightly different usage context, several approaches propose to explicitly specify variability points in a language and assist in choosing a suitable alternative at the language design time. For example, Rosemann et al. [121] introduce a concept of *a configurable reference modeling language* that allows to explicitly capture language configuration alternatives and to choose the necessary alternatives for a current language configuration. As a part of this approach, a reference modeling language is extended with configuration patterns, which represent decision points in the reference language and possible alternatives for them. Then, the choice of suitable alternatives can be done during the *customization* of the reference modeling language. As a result, a reference language is configured at design time and its customized configuration is used for modeling.

Further customization techniques are proposed in the category *Language composition* in Table 2.2. These approaches suggest to extend description languages with a customized information or to configure a language by composing it from different metamodels.

For example, Di Ruscio et al. [37] present the framework BYADL, which allows the modeler to incrementally customize a new architecture description language (ADL) based on existing ones by extending existing ADLs with domain-specific properties, views, or analysis aspects. The syntax of the new ADL is defined as a metamodel and its semantics is defined as a mapping to the core language of DUALLY [90] that fosters its interoperability. The framework defines the composition operators *match, inherit, reference, and expand* operating on metamodel classes.

Another example is the approach by Wende et al. [161] to *modularize* a language considering both its syntax and its semantics and to compose a new language based on its modularized parts. The approach of modularization produces a language consisting of *modules*, which are syntactically and semantically self-contained, reusable components. Composition operations are introduced for the abstract syntax, concrete syntax and semantics of languages. The language composition approach consists of a composition model describing language modules, a composition language describing a composition of modules, and a composition technique deriving the syntax and semantics of the composed language.

As a summary, on the one hand, the proposed language design principles aim at creating a language design of high quality but do not optimize a language according to specific goals. On the other hand, the configurability based on variations is the most closest means to create an optimal language found in this literature review, but they still do not aim at delivering an optimal specification language for a given usage context. *Therefore, a new systematic approach to design an optimal core language for the service specification has to be developed based on the investigated techniques such as viewpoints or customization.*

2. What measurable notions of language quality or language optimality exist for the design and evaluation of core languages or specification languages? Various approaches evaluating the quality of specification languages are described in the category *Language quality* in Table 2.2. These approaches propose to evaluate the language quality using different metrics and to compare different languages based on the obtained quality values.

For example, Henderson-Sellers et al. [65] suggest using several metrics to measure the size and complexity of languages defined as metamodels. The authors apply their metrics to several existing languages, e.g., Unified Modeling Language (UML) and Business Process Modeling Notation (BPMN), and compare the complexity of these languages based on the obtained values.

Krogstie et al. [85] propose a framework for the language quality evaluation. In the scope of this framework, the authors define such quality properties as the level of comprehensiveness of the language with respect to the corresponding domain, the extend of possible automatic reasoning on a language, or the appropriateness of a language with respect to standards and organizational context. Quality values for Business Process Modeling Notation (BPMN), UML activity diagrams, and Extended Enterprise Modeling Language (EEML) are obtained and compared using the presented framework.

However, these approaches do not provide any explicit definition of language optimality or any of its measurements. *Therefore, a new notion of language optimality has to be defined in this PhD thesis.*

3. What knowledge can be leveraged for the design of a core language, a service specification language, or an optimal specification language? Several approaches presented in the category *Language architectures and design approaches* in Table 2.2 propose to model the goals of a language or to consider a conceptual model of the application domain and corresponding stakeholders for the language design. For example, the derivation of a DSL by Fan et al. [44] includes among others determining the goals of the language to be designed, identifying stakeholders and a conceptual model of the given domain. *For this PhD thesis, it is necessary to identify those properties of OTF markets that are relevant for the language design of an optimal core language and how these properties can be leveraged.*

4. How conflicting requirements are considered for the design of a core language, a service specification language, or an optimal specification language? Conflicting requirements can appear at several points. One reason for conflicting requirements are different usage contexts of a language. For example, Zschaler et al. [166] suggest to handle such conflicts by explicitly modeling the variability in a language and customizing it for the given usage context. Conflicting requirements can also arise, when reusing existing specification languages in a new specification language. For example, different levels of granularity of the languages to reuse might cause conflicts. The approach by Pottinger et al. [118] identifies conflicts, which may occur during language composition, and propose ways for their resolution. The authors introduce a merge operator for language composition, define several kinds of conflicts regarding the abstract syntax of the language and its representation, which might happen during the merging, and propose means for a resolution of these conflicts.

In this PhD thesis, conflicting requirements might occur during the design of an optimal core language, e.g., during the language reuse. Conflicting requirements have to be identified and techniques to resolve them have to be developed.

5. What service specification languages exist?
Different specification languages and standards are presented in the category *Service specification languages* in Table 2.2. Specification approaches USDL, PCM, UnSCom, and others by W3C, OASIS and OMG are considered.

Barros et al. [16] present a unified service description language (USDL) applied in the industry. It supports the operations of discovery, composition, provisioning, delivery, and access. USDL captures different business, operational, and technical service properties. USDL is designed under requirements of the *comprehensive service specification* with high expressive power, modularity and extensibility of specifications. USDL is structured in modules and provides the interconnection of its modules, thus, bringing different service properties together. USDL varies based on different contexts. USDL specifications also support the reuse of existing external service specifications in WSDL or BPEL.

The Palladio Component Model (PCM) [18] is a specification language for *description of performance-relevant properties* of components broadly used in the academia. This work is interesting for this review because the terms "service" and "component" are considered synonyms for this PhD thesis as specification techniques and operations for both are very similar. PCM supports the specification of components, assembly, allocation, and usage profiles, which are created by different stakeholders during the development process of a service-oriented software system. These specifications serve as a basis for the simulation that calculates the expected response time and other performance properties of the modeled system. Using these simulation results, design decisions regarding the software system under consideration can be improved iteratively.

A standardized framework for *specification of software components* was proposed by Overhage [112]. This framework is also applicable for services as explained in the description of PCM. UnSCom defines a comprehensive language for service specification unifying existing approaches. Thus, UnSCom enables to describe structure, behavior, and quality properties of services. The authors also define design principles, which UnSCom have to follow, in order to be able to serve as an international standard for service specification.

W3C proposes a set of *standards for web services*. For service specification, W3C proposed such specification languages as WSDL [28], SAWSDL [45], Web Service Choreography [77], and OWL-S [91]. WSDL is an XML-based language for specification of service interfaces and their operations. WSDL uses XML Schema [26] to describe data structures and operation signatures. SAWSDL extends WSDL with additional semantics in the form of references to ontologies. Ontologies model concepts of a given domain in a formal way using, in particular, the Web Ontology Language OWL [93]. The Web Service Choreography describes the observable behavior of collaborating services including their interaction, ordering, and interaction constraints. OWL-S describes service functionality, service category and service interaction. Similar to SAWSDL, OWL-S specifications refer to ontologies.

As well as W3C, OASIS [103] suggests a set of standards for web services, which are often named *WS-* standards*. Their standards include among others the specifications of service security (WS-Security) [102], service reliability (WS-Reliability) [72], service trust (WS-Trust) [101]. WS-Security defines a standard for secure messaging among services using different security models and encryption techniques. WS-Reliability defines a standard for reliable messaging including a definition of reliability and reliable communication protocols. WS-Trust defines a standard to enable trusted messaging including exchange of security tokens and establishment of trust relationships.

Object Management Group (OMG) [104] proposes standards developed for different academic and non-academic institutions, and end users. Among all specifications proposed by OMG, the most interesting for this review are *modeling and metadata specifications* [105]. They include the general-purpose Unified Modeling Language (UML) [108] broadly used for modeling and analyzing soft-

ware and among others services in the industrial and academic contexts. OMG also provides so-called UML Profiles, which are light-weight extensions of the UML for modeling special kinds of software systems or software properties, e.g., a UML profile for Modeling and Analysis for Real-time and Embedded Systems (MARTE) [107] or a UML profile for quality of service and fault tolerance [106].

These service specification languages aim to describe different service properties for service discovery, or service composition. Since these languages are established for the service specification, they have to be considered for the reuse in an optimal core language for OTF Computing. However, none of these languages can be used as an optimal core language, because they are not explicitly optimized for performing the service operations in different service markets. In addition, the relation between the reliability of operation results and the efficiency of the service operations was not an object of a thorough study in these approaches. *As a summary, an optimal core language in the OTF market has to be build anew by reusing the established specification languages identified in this literature review.*

How to increase the reuse of existing languages in a core language, a service specification language, or an optimal specification language? Languages that can be reused for the design of an optimal core language are listed in the category *Service specification languages* in Table 2.2. For the reuse of existing languages, the approaches for language composition presented in the category *Language composition* in Table 2.2 can be applied as well. These approaches can be used, in order to compose existing languages into an optimal core language, so that these languages get reused within the core language. *For this PhD thesis, suitable approaches to compose existing languages during the design of a core language shall be selected.*

2.2 Related Work on User-Friendly Language Transformation

This section presents the systematic literature review of approaches addressing the problem of the user-friendly language transformation described in Section 1.2.2. Section 2.2.1 describes the review protocol. Section 2.2.2 gives an overview of the results obtained according to this protocol and their evaluation.

2.2.1 Review Protocol

Similar to the protocol in Section 2.1.1, this protocol begins with a list of *research questions*, which are tackled by existing approaches considered in this review. Research questions for this area of related work are a generalized version of the research questions presented in Section 1.2.2.

1. What approaches for user-friendly language transformation exist?

2. Which specification languages these approaches are applicable to?
3. How is the semantics of specification languages considered for user-friendly transformation?
4. How can the quality of the language transformation be evaluated?

In this area of related work, approaches for transforming specification languages in a user-friendly manner are investigated. Another goal of this review is to find out, how the semantics of languages is considered for the transformation. In addition, the review investigates how existing approaches evaluate the quality of the language transformation.

After the research questions for the review are presented, *search strategies* used to find related literature have to be described. The first search strategy applied in this review is the *broad search for classifications of language transformation approaches* using keywords. The reason for this strategy is the fact that the area of language transformations is broadly researched and systematized, especially in comparison to the area of language optimization investigated in Section 2.1. Based on this systematization, the goal is to find out what existing approaches tackle user-friendly language transformations in some extent.

From the found classifications of model transformation approaches, categories of relevant approaches are identified. The in-depth search strategy investigates references in the literature sources for existing approaches from the relevant categories. In order to obtain a better knowledge about the found approaches, other literature sources by the same authors published on this topic are investigated as well. Afterwards, the broad search using keywords is applied for the years not covered by the review, in order to find further existing approaches for the relevant categories. The same keywords, inclusion and exclusion criteria are used for both for broad and in-depth search. Finally, the found approaches are compared with each other using criteria relevant for the problem statement presented in Section 1.2.2.

The search was conducted in the following digital libraries: Google Scholar [58], ACM Digital Library [1], IEEE Xplore [68], and Springer Link [135]. For the Internet search, the search machine of Google [57] was used.

Keywords used for search queries in the broad search strategy have to be determined. The first keyword is the term `model transformations` established for referring to the transformation of languages having a formal definition. Since classifications of model transformation approaches were investigated, the keyword `classification` and its synonyms `review`, `survey`, and `taxonomy` were considered. Only literature sources fully covering the defined keywords are considered since a focus on classification of model transformations is feasible and a further broadening of the search space is not required.

After keywords have been defined, *inclusion and exclusion criteria* for found literature sources are described. The inclusion criteria are the same as in Section 2.1.1. In addition, only one exclusion criteria was defined: Literature sources comparing the classifications of model transformations are omitted in

this review. Since the focus of this review is to understand the classification of model transformations and not to compare the classification methods, such approaches are irrelevant for this review.

2.2.2 Evaluation of the Review Results

Section 2.2.2.1 presents the evaluation of the review results on classification of model transformation approaches. The goal of this evaluation is to understand what kinds of model transformation approaches exist, and which of them address the research questions of this review. Section 2.2.2.2 provides an overview of the review results for the category of model transformation by-example approaches chosen after the analysis of the classifications. Afterwards, the evaluation of the review results is described.

2.2.2.1 Evaluation of Classifications of Model Transformation Approaches

Table 2.3 introduces literature sources for classification of model transformation approaches found based on the broad search strategy. The category of approaches addressing the research questions of this review is highlighted.

Table 2.3: Search results based on the broad search strategy

Authors	Title	Reference
Czarnecki et al.	Classification of Model Transformation Approaches	[30]
Czarnecki et al.	Feature-Based Survey of Model Transformation Approaches	[31]
Kappel et al.	**Model Transformation By-Example: A Survey of the First Wave**	[75]
Mens et al.	A Taxonomy of Model Transformation	[95]
Taentzer et al.	Model Transformations by Graph Transformations: A Comparative Study	[146]

The survey by Kappel et al. is the only one considered further in this review as it tackles model transformation by-example techniques, which is a user-friendly approach to model transformations.

Kappel et al. present a survey of model transformation by-example approaches [75]. These approaches emerged to tackle the problem that it is difficult for modelers to create model transformations because the modelers only know the concrete syntax of a language and not its formal language definition. In addition, modelers often work only on a subset of a certain language (a so-called view), and, thus, want to have a model transformation for the relevant part of the language but not the whole language definition. A further obstacle for modelers to learn the formal definition of a language is a gap between the

representation of language elements in the concrete syntax and the representation in the language definition. Considering all the issues mentioned above, the user-friendly technique of model transformation by-example was proposed.

The idea of model transformation by-example is based on similar techniques of query-by-example [165] and programming by-example [87]. According to model transformation by-example, model transformations are derived based on examples given by a modeler in the concrete syntax of a language. The authors introduce two types of approaches: demonstration-based and correspondence-based. In demonstration-based approaches, modifications of example models by a modeler in the editor are recorded, and a model transformation for the corresponding languages is derived based on these recordings. In correspondence-based approaches, a modeler gives correspondences between example models of the languages, and a model transformation is derived based on these correspondences. Afterwards, the authors group existing model transformation by-example approaches into exogenous transformations, which are transformations defined between two different languages, and endogenous transformations, which are transformations defined for models of the same language.

Why other surveys are not considered further are given in the following.

Czarnecki et al. introduce a feature-based survey of model transformation approaches [31], which is based on their early work [30]. Its aim is to analyze existing approaches and to make design decisions in model transformation tasks explicit. However, the authors do not address the user-friendly aspect of model transformation approaches.

Mens et al. propose a taxonomy of model transformation [95], which assists in choosing a language for model transformation and tools for a transformation task. Mens et al. indeed mention the aspect of usability, however, their term refers to model transformation languages or tools but not to model transformation approaches.

Taentzer et al. compare four graph-based model transformation approaches and their tool support in the comparative study [146]. This comparative study is not considered further in this review because the authors do not address the user-friendly aspect of model transformation approaches.

2.2.2.2 Evaluation of Model Transformation By-Example Approaches

In the previous section, the category of Model Transformation By-Example (MTBE) approaches is selected for being relevant in this review. The survey of Kappel et al. [75] describes the classification of existing MTBE approaches. This section evaluates literature sources describing such approaches.

The investigated MTBE approaches have to deal with exogenous transformations, i.e., transformations between different languages. Service and requirements specifications written in languages other than the core language have to be transformed into the core language, for which the automated market operations are defined. A further restriction is the consideration of only correspondence-

based approaches and no demonstration-based ones. The reason is the highly interactive nature of demonstration-based approaches [86]. The modeler is guided to demonstrate concrete parts of the model transformation iteratively in an interactive manner. Since the OTF market is designed to be highly automated and actors of the OTF market are distributed, such interaction is not really possible for the design of model transformations.

Table 2.4 lists approaches obtained using both in-depth and broad search strategies. The in-depth search strategy was based on the survey by Kappel et al. [75]. The broad search strategy was applied for literature sources from the year 2012 to the year 2014 because literature sources till the year 2011 are adequately covered by the survey of Kappel et al. [75]. For the evaluation, the literature sources are grouped so that a group corresponds to a MTBE approach having evolved over years. Each group is highlighted with a different color.

Table 2.4: Search results based on the in-depth and broad search strategies

Authors	Title	Reference
Balogh et al.	Model Transformation by Example Using Inductive Logic Programming	[14]
Dolques et al.	From Transformation Traces to Transformation Rules: Assisting Model Driven Engineering Approach with Formal Concept Analysis	[40]
Dolques et al.	Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis	[41]
Dolques et al.	Easing Model Transformation Learning with Automatically Aligned Examples	[39]
Faunes et al.	Genetic-Programming Approach to Learn Model Transformation Rules from Examples	[46]
García-Magariño et al.	Model Transformation By-Example: An Algorithm for Generating Many-to-Many Transformation Rules in Several Model Transformation Languages	[53]
Kessentini et al.	Model Transformation as an Optimization Problem	[79]
Kessentini et al.	Search-based Model Transformation by Example	[80]
Saada et al.	Learning Model Transformations from Examples using FCA: One for All or All for One?	[125]
Saada et al.	Generation of Operational Transformation Rules from Examples of Model Transformations	[124]
Strommer et al.	A Framework for Model Transformation By-Example: Concepts and Tool Support	[142]

Table 2.4: Search results based on the in-depth and broad search strategies

Authors	Title	Reference
Varró et al.	Model Transformation by Example	[154]
Varró et al.	Automating Model Transformation by Example Using Inductive Logic Programming	[155]
Wimmer et al.	Towards Model Transformation Generation By-Example	[162]

Table 2.5 presents the evaluation performed for six approaches presented as groups in Table 2.4 based on chosen comparison criteria. Requirements on the user-friendly transformation required in the OTF market motivate the comparison criteria. The rationals for them are explained in the following.

The criteria `Representation of abstract syntax` and `Representation of example models` were chosen to evaluate how an abstract syntax and example models have to be defined to serve as input for any MTBE approach. `Definition of model correspondences` is essential to understand the form, in which existing MTBE approaches allow to define correspondences between example models. The way to define such correspondences has to be suitable for market actors in the OTF market. `Evaluation of the quality of models` indicates whether an approach provides any means to evaluate the quality of the input example models as their quality reflects in the quality of the transformation learned based on these example models. In order to get an overview about the learning algorithms applied for MTBE, the criterion `Underlying learning technique` is considered.

`Type of model transformations` is chosen, in order to understand the output format of existing MTBE approaches. `Executability of model transformations` indicates, whether a resulting model transformation can be directly executed on models, in order to perform the language transformation. In order to understand the expressiveness of resulting model transformations, the kinds of correspondences used in rules of this transformation are considered (`Correspondences in rules`). For the evaluation of resulting model transformations, how the transformation is evaluated (`Evaluation of model transformations`) and what example are used (`Evaluation examples`) are considered. The final criteria are `Automation of the approach` and `Tool support`, which are important for experiments with the corresponding MTBE approach and its possible integration into the tool support for the OTF market.

Table 2.5: Comparison of existing MTBE approaches

Comparison criterion	Dolques et al.	Faunes et al.	García-Magariño et al.	Kessentini et al.	Varró et al.	Wimmer et al.
Representation of abstract syntax	Concepts according to RCA	Sets of fact templates	Metamodels with OCL constraints	Metamodels	Metamodels mapped to predicates	Metamodels with OCL constraints
Representation of example models	Lattices obtained using RCA	Sets of facts	Instances of metamodels	Sets of predicates	Prolog clauses	Instances of metamodels
Definition of model correspondences	Trace links	Model pairs	Model pairs	Model pairs, mapping blocks based on predicates	Model pairs	Trace links
Evaluation of the quality of models	No	No	No	No	No	No
Underlying learning technique	RCA, transformation patterns	Genetic programming	Inference algorithm	PSO, simulated annealing	Inductive Logic Programming	Pattern-based reasoning
Type of model transformations	JESS rules	JESS rules	ATL rules	Trace links	VIATRA2 transformation	ATL rules
Executability of model transformations	Directly	Directly	Directly	Not mentioned	Directly	Directly

Table 2.5: Comparison of existing MTBE approaches

Comparison criterion	Dolques et al.	Faunes et al.	García-Magariño et al.	Kessentini et al.	Varró et al.	Wimmer et al.
Correspondences in rules	1-to-1, 1-to-N	Arbitrary	Arbitrary	1-to-1, N-to-N	1-to-1, N-to-N	1-to-1
Evaluation of model transformations	Proof-of-concept	Testing with known transformation results	Proof-of-concept	Evaluation of fitness and precision of model transformations	Testing on example models	Proof-of-concept
Evaluation examples	Latex → HTML, UML class diagrams → entity-relationship models	UML class diagrams → relational databases, (nested) sequence diagrams → statecharts	11 examples	UML class diagrams → relational schemata	UML class diagrams → relational databases	UML class diagrams → relational databases
Automation of the approach	Automated	Automated	Automated depending on examples	Automated	Automated with manual refinement	Semi-automated with user interaction
Tool support	Exists but unavailable	Exists but unavailable	Exists but unavailable	Not mentioned	Available	Available

To summarize, all the presented MTBE approaches are either fully automated or semi-automated. They output directly executable model transformation covering 1-to-1, N-to-N or arbitrary correspondences for language constructs. All of the approaches work either directly on metamodels and their instances, or transform them in an internal format first. Concerning the evaluation of resulting model transformation, three of six approaches use a solid evaluation (testing or comparison based on chosen metrics). However, none of the approaches mentions how the quality of the considered set of example models is evaluated.

2.2.2.3 Evaluation Summary

The research questions of this review are answered as follows:

1. What approaches for the user-friendly language transformation exist? Section 2.2.2.1 presents the investigation of existing classifications of model transformation approaches. From the existing classifications, only the survey by Kappel et al. [75] describes model transformation by-example (MTBE) approaches allowing the user-friendly language transformation. Based on this survey, six existing MTBE approaches are compared in Section 2.2.2.2 according to the comparison criteria relevant for the problem statement in Section 1.2.2.

2. What specification languages these approaches are applicable to? The most MTBE approaches are applicable to languages with the abstract syntax defined as metamodels. For example, the approach by García-Magariño et al. requires languages represented in the form of Ecore metamodels [140] refined with OCL constraints [109]. The approaches by Dolques et al. and Faunes et al. use different formats needed for their inference algorithms like concepts for the relational concept analysis or a set of fact templates correspondingly.

3. How the semantics of specification languages is considered for the user-friendly transformation? In the MTBE approaches, the semantics of specification languages is considered in the form of examples. Examples represent common usage patterns of the source and target specification languages and semantic mappings between them. This guarantees that meaningful semantic correspondences are considered in the resulting model transformation.

4. How the quality of the language transformation can be evaluated? The existing MTBE approaches make use of the following techniques: testing, proof-of-concept, or evaluation of predefined metrics. A test suite of models with a known transformation result is used for testing of resulting model transformations. These models usually differ from the models used for learning of the model transformation. Another possibility is to use a set of model transformations, which are expected to result from given example models. In this case, the resulting model transformation is compared to the expected model transformation. For the evaluation as proof-of-concept, the developed approach is applied to certain source and target languages. The evaluation of such metrics as a fitness of the resulting model transformation is possible for the approaches using optimization algorithms.

For OTF Computing, no existing MTBE approach can be used in an OTF market without further modifications.

On the one hand, a large search space of possible transformations exists in an OTF market. This is because there are no restrictions on possible combinations of language constructs from the source and target languages in the transformation rules as well as there are no restrictions on combination of such rules in a model transformation. On the other hand, a perfectly correct transformation is not required in the OTF market. The transformation has to transform specifications into the core language so that the service matching of these specifications delivers results with sufficient reliability.

In order to cope with the large size of the search space and to find a sufficiently correct model transformation, a metaheuristic optimization approach is suitable. Metaheuristic is an approximate algorithm that allows an efficient and effective solution finding to hard combinatorial optimization problems [78]. It is especially suitable for solving optimization problems under the condition of incomplete information or limited computation capacity [20]. From the approaches presented in Table 2.5, only the approaches of Faunes et al. und Kessentini et al. use a metaheuristic in their learning algorithms. They use genetic algorithms and particle swarm optimization correspondingly.

In order to increase the quality of the learned model transformation, the quality of example models serving as an input for the approach has to be evaluated and, if possible, improved by market actors. A resulting model transformation has to be directly executable without any manual adaptation needed from market actors as they do not have the required expertise to modify the transformation. Since market actors need the model transformation to enter the OTF market, the corresponding MTBE approach has to deliver it as fast as possible.

According to the requirements stated above, the approaches by Faunes et al. and Kessentini et al. do not satisfy them. The approach by Faunes et al. works on sets of facts as a representation for example models. The approach by Kessentini et al. works on sets of predicates as a representation for example models and does not produce a directly executable model transformation. Both approaches do not consider the quality of example models and, thus, do not provide means to evaluate or improve it. Regarding the convergence to an optimal solution, both approaches use random-based operators. In order to achieve a faster convergence, less random operators are required in the OTF market. As a result of all the facts mentioned above, a new mtbe approach is developed in this PhD thesis.

3 Requirements and Overview of the Solution

This section introduces requirements and an overview of the solution developed in this PhD thesis. Section 3.1 presents requirements on a solution for the problem statements described in Section 1.2. Requirements are derived from the motivation, problems statements and the related approaches described above. Section 3.2 shows an overview of the solution satisfying these requirements.

3.1 Requirements on the Solution

This section starts with requirements on the approach for the design of an optimal core language listed in Section 3.1.1. Afterwards, Section 3.1.2 presents requirements on the user-friendly language transformation approach.

3.1.1 Requirements on Design of an Optimal Core Language

The following requirements are derived from the motivation scenario described in Section 1.1, the problem statement described in Section 1.2.1, and the results of the systematic literature review described in Section 2.1. The design approach developed in this PhD thesis is called LOpt (Language Optimizer).

Requirements on an optimal core language:

R.1.1.1: An optimal core language shall have a formal syntax definition, in order to facilitate the automated execution of market operations on specifications written in the core language. This requirement is based on Challenge 2 described in Section 1.1 and the design principles for core languages summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.2: An optimal core language shall have a formal definition of the semantics for those parts, for which a formal definition of the semantics is required for the execution of the automated market operations. This requirement is based on Challenge 2 described in Section 1.1 and the design principles for core languages summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.3: An optimal core language shall provide all language constructs for the specification of structural, behavioral, and non-functional service properties, which are relevant for the automated execution of market operations in an OTF market, because the goal of the core language is to optimally support these automated market operations. This requirement is derived from the problem statement described in Section 1.2.1 and the design principles for core languages summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.4: A core language shall be optimal for the automated market operations in a given OTF market. The optimal core language of an OTF market contributes to the success of that market. This requirement is based on the problem statement described in Section 1.2.1.

R.1.1.5: Service specifications written in an optimal core language shall serve as basis for the definition and execution of the automated market operations, in order to avoid the infeasible effort of defining the operations for all existing specification languages. This requirement is based on the problem statement described in Section 1.2.1 and the design principles for core languages summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.6: An optimal core language shall have a modular structure because it facilitates the reuse and maintainability of its parts. This requirement is based on the best practices for language design summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.7: An optimal core language shall be orthogonal. Orthogonality of a language guarantees that each modeled concept is expressed in exactly one good way in this language. Orthogonality is a principle of a good language design that fosters a better maintainability of a language. This requirement is based on the best practices for language design summarized in the Research question 1 in Section 2.1.2.2.

R.1.1.8: An optimal core language shall reuse existing service specification languages of market actors. This shall foster the acceptance of an optimal core language in an OTF market, because established language constructs and their semantics already approved by market actors are reused. This requirement is based on the problem statement described in Section 1.2.1, the best practices for language design summarized in the Research question 1 in Section 2.1.2.2, and the existing specification languages and language composition approaches summarized in the Research question 5 in Section 2.1.2.2.

R.1.1.9: An optimal core language shall be designed to facilitate transformations from other service specification languages. The goal is to enable the transformation and, thus, a reuse of specifications written

by market actors for the execution of the automated market operations. This requirement is based on Challenge 3 described in Section 1.1 and the design principles for core languages summarized in the Research question 1 in Section 2.1.2.2.

Requirements on the design approach LOpt:

R.1.2.1: The design approach LOpt shall provide a systematic approach to create an optimal core language fulfilling the requirements *R.1.1.1*–*R.1.1.9*. This requirement is based on the investigation of existing design techniques for languages summarized in the Research question 1 in Section 2.1.2.2.

R.1.2.2: The design approach LOpt shall be automated. The automated execution of LOpt allows to find an optimal core language without subsequent manual effort from users. This requirement is motivated in Section 1.1 as market actors have no expertise in language design.

R.1.2.3: The design approach LOpt shall leverage the properties of an OTF market of its application, in order to consider its specifics during the language design. This requirement is based on the problem statement described in Section 1.2.1 and the discussion in the Research question 3 in Section 2.1.2.2.

R.1.2.4: The design approach LOpt shall use a measurable definition of the language optimality. This definition has to be used to find a core language, for which the application of the automated market operations delivers the best results regarding the definition. This requirement is based on the problem statement described in Section 1.2.1 and the discussion in Research question 2 in Section 2.1.2.2.

R.1.2.5: The design approach LOpt shall adhere to the guidelines for the design of specification languages. This requirement is based on guidelines like defining viewpoints or using customization summarized in the Research question 1 in Section 2.1.2.2.

R.1.2.6: The design approach LOpt shall solve conflicts arising during the design of an optimal core language. This requirement is based on the problem statement described in Section 1.2.1 and the discussion in the Research question 4 in Section 2.1.2.2.

3.1.2 Requirements on User-Friendly Language Transformation

The following requirements are derived from the motivation scenario in Section 1.1, the problem statement in Section 1.2.2, and the results of the systematic literature review in Section 2.2. The design approach for the user-friendly model transformation is called mtbe (Model Transformation By-Example).

Requirements on the model transformation:

R.2.1.1: The model transformation shall apply to languages having a formal definition of the abstract syntax. This requirements is based on the problem statement in Section 1.2.2 and the evaluation summary in Section 2.2.2.3.

R.2.1.2: The model transformation shall be directly executable on specifications of the source language. No manual refinement or adaptation of the model transformation by market actors shall be required. This requirements is based on the problem statement described in Section 1.2.2 and the evaluation summary in Section 2.2.2.3.

R.2.1.3: The model transformation shall cover the relevant parts of the language only. The reason is that a modeler often uses only parts of their language because of the large size and high complexity of many specification languages. Therefore, a more focused transformation of the relevant language parts into the core language is required, in order to cope with this complexity. This requirement is based on the motivation scenario in Section 1.1 and the evaluation summary in Section 2.2.2.3.

R.2.1.4: The model transformation shall contain arbitrary correspondences between language constructs of the source and target languages (i.e., 1-to-1, 1-to-N-, N-to-1, and N-to-M). This requirement is based on the the evaluation summary in Section 2.2.2.3.

R.2.1.5: The model transformation shall lead to acceptable matching results of specifications transformed in the optimal core language. This requirements is based on the problem statement in Section 1.2.2.

Requirements on the user-friendly language transformation approach (mtbe):

R.2.2.1: The mtbe approach shall use the technique of Model Transformation By-Example to create a model transformation. As a result, mtbe shall derive a model transformation based on correspondences between the source and target languages given in the form of pairs of example specifications written in these languages, i.e., example models. The mappings have to be given at the level of example specifications because most market actors in the OTF market are assumed to have no expertise in language design and, thus, cannot specify fine-grained mappings. This requirement is based on the choice of the Model Transformation By-Example technique in Section 2.2.2.1.

R.2.2.2: The mtbe approach shall work on formal language definitions of the source and target languages and produce a directly executable model transformation for these languages. This requirement is based on the requirements *R.2.1.1* and *R.2.1.2*.

R.2.2.3: The mtbe approach shall be able to evaluate the quality of the given example models with respect to the given languages. Thus, market actors can control and improve the quality of their models. This requirement is motivated in Section 2.2.2.3.

R.2.2.4: The mtbe approach shall be able to identify language parts relevant for the model transformation. It is based on *R.2.1.3*.

R.2.2.5: The mtbe approach shall leverage the knowledge from the mappings between example models. This knowledge has to be used in the metaheuristic operators of the mtbe approach. It is motivated by the requirement that the metaheuristic has to converge effectively to an optimal solution as explained in Section 2.2.2.3.

R.2.2.6: The mtbe approach shall use a metaheuristic for the derivation of a model transformation. A metaheuristic allows to effectively and efficiently find a solution to an optimization problem with a large search space. This requirement is motivated in Section 2.2.2.3.

R.2.2.7: The mtbe approach shall create a model transformation consisting of rules with arbitrary correspondences between language constructs (i.e., 1-to-1, 1-to-N-, N-to-1, and N-to-M) combined in a control flow of arbitrary complexity. This requirement is based on the requirement *R.2.1.4*.

R.2.2.8: The mtbe approach shall be independent from any concrete model transformation language. The approach shall provide a possibility to create model transformations in a chosen supported model transformation language. This requirement is motivated by the fact that some market actors might have enough expertise to work on the model transformations directly. Thus, market actors can use different intuitive or familiar notations to modify the resulting model transformations. This is similar to arbitrary languages, which market actors are able to use for specifying services (see Section 1.1).

R.2.2.9: The mtbe approach shall define a fitness of model transformations specific for the OTF market. The new fitness definition is important because the main goal of the model transformation in the OTF market is to create service specifications, which can be used for the reliable service matching. This goal differs from the fitness functions used in existing approaches, which optimize the model transformation with respect to the set of example models only. This requirement is based on the requirement *R.2.1.5*.

3.2 Overview of the Solution

Section 2.1.2.2 and Section 2.2.2.3 state that no approach exists, which can be directly applied as a solution to the problem statements from Section 1.2 and which satisfies the requirements from Section 3.1. Thus, this section presents an overview of the new solution, which solves the presented problem statements and satisfies the requirements. This solution aims at overcoming the heterogeneity in OTF markets and significantly contributes to their success and acceptance.

Figure 3.1 introduces an overview of the solution, which consists of two approaches LOpt and mtbe. Each approach addresses one problem statement.

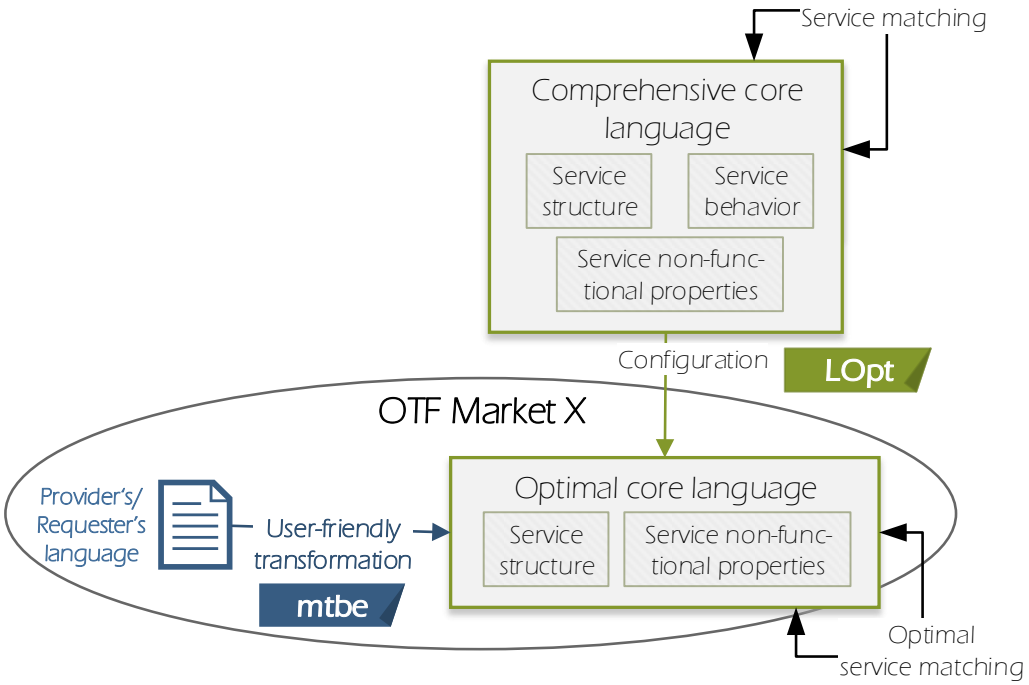


Figure 3.1: Overview of the solution

LOpt is the approach to design an optimal core language for a given OTF market. LOpt contains a `comprehensive core language`, which serves as a basis for optimal core languages in different OTF markets. The comprehensive core language covers various structural, behavioral, and non-functional properties of services and allows to specify services comprehensively. LOpt configures the comprehensive core language to obtain a core language that optimally supports automated market operations in the considered OTF market.

Figure 3.1 shows an application of LOpt to a concrete OTF market. The `optimal core language` for this market is created by omitting those parts of the comprehensive core language, which are irrelevant for performing the automated market operations in this market. In this case, these are `Service behavior` properties.

For the configuration, LOpt uses properties of the given OTF market as well as the expertise in the domain of service engineering. As depicted in Figure 3.1, the market operation of service matching is defined for the comprehensive core language as well as for the optimal core language. The service matching is configured based on properties of the optimal core language and of the underlying OTF market. The configuration of the service matching is out of scope of this PhD thesis and our papers [7, 8, 117] give more information on this topic.

mtbe aims at supporting market actors to transform their existing specifications into the optimal core language of the considered OTF market. The approach does not require any expertise in language design from market actors.

mtbe allows the market actors to specify correspondences between semantically equivalent example specifications in their specification language and the optimal core language. To create these specifications, the market actors have to learn the syntax and semantics of the corresponding optimal core language. Based on these knowledge, the market actors have to map their example specifications, which are representative for the usage of their language, into the optimal core language. The market actors are responsible for the semantic equivalence of the created mappings, which they also can check by using such automated techniques as DMM [132].

Based on these correspondences, a language transformation between these languages is derived using a metaheuristic derivation algorithm. This derivation algorithm allows the effective and efficient exploration of the solution space of possible model transformations. The obtained language transformation preserves the semantics of the considered languages defined in the given examples. As a result, market actors can perform the optimal market operation defined for the optimal core language on their service and requirements specifications.

In the following part of this PhD thesis, each approach is described in a separate chapter. Chapter 4 describes the approach LOpt for the design of an optimal core language, while Chapter 5 presents the approach mtbe for the user-friendly model transformation.

4 Design of an Optimal Core Language

This chapter presents the approach LOpt for the design of an optimal core language for a given OTF market. This approach provides a solution for the problem statement presented in Section 1.2.1 and satisfies the requirements stated in Section 3.1.1 that is shown as a part of the evaluation presented in Section 4.3. This chapter starts by describing the concept of an optimal core language in Section 4.1. Based on this concept, Section 4.2 describes the approach LOpt. Section 4.3 presents the evaluation of this approach on case studies.

4.1 Concept of an Optimal Core Language

This section begins with a formal definition of a core language for an OTF market. For this purpose, Section 4.1.1 extends the classic language definition with a new formal definition of the language pragmatics. Section 4.1.2 uses this definition of the language pragmatics to define the notion of the core language optimality. This notion defines, which core language is considered to be optimal with respect to the execution of the automated market operations in the considered OTF market.

4.1.1 Core Language Definition

Figure 4.1 introduces a formal language definition used in this PhD thesis. This definition is based on the notions by Stahl et al. [138] and Völter et al. [157], and the notation of UML class diagrams [108] is chosen to illustrate it. A formal language definition is required because specification languages have to be defined formally, in order to create formal service specifications in OTF markets (see Challenge 2 in Section 1.1).

As depicted in Figure 4.1, a `Language` consists of three parts: `Abstract Syntax`, `Semantics`, and `Pragmatics`.

The `Abstract Syntax` represents modeling elements and their relations between each other [138, p. 29]. Abstract syntax is independent from any concrete representation of modeling elements and their relations. Several techniques to define the abstract syntax of a language exist. Metamodels are an established technique according to the model-driven software development paradigm [24, 71, 137]. A metamodel describes the abstract syntax in the form

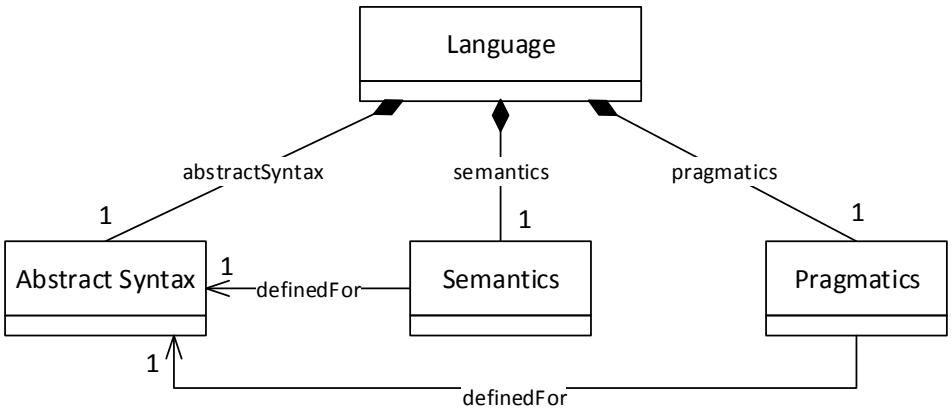


Figure 4.1: Language definition based on [138] and [157]

of a visual model [157, p. 27]. Syntactic metalanguages are another way to define the abstract syntax in the form of rules. One well-known syntactic metalanguage is the Extended Backus-Naur Form [69].

The `Semantics` is defined for the language elements described in the abstract syntax (depicted by the association `definedFor` between the classes `AbstractSyntax` and `Semantics`). Völter et al. [157, p. 26] distinguish between the static and dynamic semantics of a language, which are both represented by the class `Semantics` in Figure 4.1. The static semantics defines a set of constraints, which a specification in this language has to conform to. A broadly used declarative language for specifying the static semantics is the Object Constraint Language (OCL) [109]. The dynamic, or execution, semantics defines the behavior of an executable specification. For the specification of the dynamic semantics, different notations exist, e.g., graph-based approaches like Dynamic Meta Modeling (DMM) [64, 132].

The `Pragmatics` is defined for the abstract syntax of a language as well. Stachowiak defines a model as pragmatic, if all its elements assist explicitly specified users in fulfilling specified operations for a certain time period [136]. In order to formalize the notion of language pragmatics, Figure 4.2 presents an *extended language definition* introduced in this PhD thesis. The language pragmatics is formalized as a set of operations performed on this language. Operations are represented by the new class `Operation` highlighted as gray. They are defined for the abstract syntax of the language and executed on specifications written in it (represented by the association `definedFor` between the classes `Operation` and `AbstractSyntax`).

Based on the extended language definition given in Figure 4.2, Definition 1 formalizes the definition of a core language in an OTF market. This definition is used later for the formalization of language optimality.

A core language consists of an abstract syntax as_{CL}, a semantics sem_{CL}, and a pragmatics $prag_{CL}$. The abstract syntax as_{CL} is defined using metamodels.

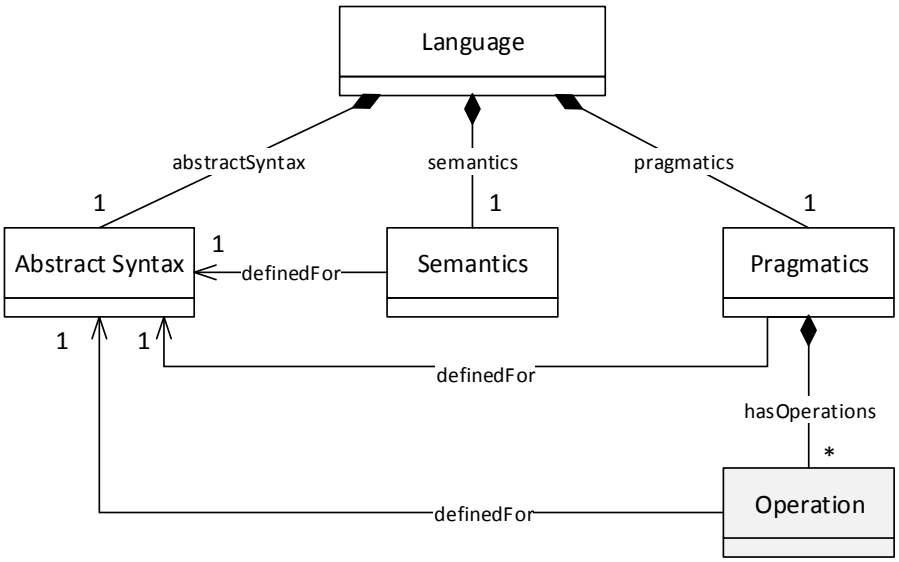


Figure 4.2: Extended language definition

The reason for this choice is the central usage of metamodels in the paradigm of model-driven software development (MDSD) [137, p. 13]. This PhD thesis follows the principles of MDSD due to such goals like abstraction, separation of concerns, and interoperability. The semantics sem_{CL} represents both static and dynamic semantics of the core language. The pragmatics $prag_{CL}$ contains the operation of the service matching μ formalized in the following.

Definition 1 (*Core language CL*)

Given as_{CL} : AbstractSyntax, sem_{CL} : Semantics, $prag_{CL}$: Pragmatics, and μ : Operation, where μ is the operation of service matching, then CL : Language is the definition of a core language, for which holds:
$CL = \{as_{CL}, sem_{CL}, prag_{CL}\}$ and $prag_{CL} = \{\mu\}$.

In order to formalize the operation of service matching, Definition 2 formalizes the set of all specifications written in a core language. This set contains all specifications, which conform to its abstract syntax as_{CL}.

Definition 2 (*Specifications of a core language $Specs_{CL}$*)

Given as_{CL} of the core language CL and the predicate ***instances*** *that determines the instances of as_{CL}, then $Specs_{CL} = \{instances(as_{CL})\}$ is the set of specifications of the core language CL.*

Definition 3 formalizes the operation of service matching for a core language.

The service matching calculates, to which degree the service described in the given service specification *spec* satisfies the requirements stated in the requirements specification *req*. The service matching takes as input a service specification *spec* and a requirements specification *req*, which are both written

in the same core language. Then, it outputs the degree of matching, which can be expressed in different formats, e.g., a real number in the range from 0 to 1.

Definition 3 (*Service matching μ*)

Given a service specification $spec \in Specs_{CL}$ and a requirements specification $req \in Specs_{CL}$, then $\mu : Specs_{CL} \times Specs_{CL} \rightarrow [0,1]$ is the operation of the service matching defined for the core language CL, which determines, to which the degree from 0 to 1 the service described in $spec$ satisfies req.

4.1.2 Core Language Optimality

The notion of the core language optimality is strongly connected with the notion of the core language pragmatics and, thus, with the operations executed on specifications of the core language. In this PhD thesis, the operation of service matching is considered for the core language optimality. Other automated market operations like service analysis are a future work. The optimality of the core language for an OTF market is determined by the quality of the service matching executed on specifications written in this core language.

The quality of the service matching can be characterized by the following two properties: *efficiency* and *effectiveness*. Efficiency defines the capability to provide appropriate performance with respect to the amount of resources used under stated conditions [70]. Effectiveness defines a measurable degree of how good a system achieves its objectives [63]. Formal definitions of these properties for the service matching are given in the following.

Efficiency and effectiveness of the service matching are measured on a test collection. A *test collection* represents properties specific for an OTF market, for which this test collection was created. It contains specifications, which show a representative usage of the core language in the considered market. Thus, through a test collection, the specifics of this market are considered by the calculation of the matching efficiency and effectiveness.

Definition 4 describes the notion of a test collection defined for a core language CL. A test collection is a set of pairs of a requirements specification and a service specification. For each pair of a service specification and a requirements specification from a test collection, it is truly known (based on expert knowledge), to which degree the service described by the service specification matches the requirements specified in the requirements specification.

Definition 4 (*Test collection TC*)

Given a natural number N and the oracle **match** $: Specs_{CL} \times Specs_{CL} \rightarrow [0,1]$ *that truly knows, to which degree a service specification matches a requirements specification, then $TC = Specs_{CL} \times Specs_{CL} \times [0,1]$ is a test collection containing N specification pairs and their expected matching result:*
$p_i = (spec, req, match(spec, req)) \in TC$.

The matching efficiency is measured as the run time, which the service matching requires to calculate matching results for a given test collection. Thus, the run time is relative to the realization of the service matching and the chosen test collection. Definition 5 formalizes the notion of the matching run time as the mean run time to compute a matching result for one specification pair.

Definition 5 (*Average matching run time* rt_{μ}^{TC})

Given μ and the run time $rt(\mu(spec_{p_i}, req_{p_i}))$ necessary to compute a matching result for $p_i = (spec_{p_i}, req_{p_i}, match(spec_{p_i}, req_{p_i})) \in TC$, then $rt_{\mu}^{TC} = \sum_{i=1}^{N} rt(\mu(spec_{p_i}, req_{p_i}))/N$ is the average run time for the service matching μ measured for a specification pair of the test collection TC.

Definition 6 formalizes the matching efficiency as a reversed matching run time per specification pair of a given test collection. The definition indicates that the longer the service matching takes to compute matching results, the lower is the matching efficiency, and vice versa.

Definition 6 (*Average matching efficiency* ε_{μ}^{TC})

Given the average matching run time rt_{μ}^{TC} of a test collection TC, then $\varepsilon_{\mu}^{TC} = 1/rt_{\mu}^{TC}$ is the average matching efficiency for the test collection TC.

The effectiveness of the service matching is defined by a measurable degree of how good the services matching to the stated requirements are retrieved. This objective can be measured by the metrics of *precision* and *recall* used in the area of information retrieval for measuring the effectiveness of the retrieval process [126]. In the following, the formalization of these metrics is presented.

Definitions 7–8 formalize the sets of triples from a test collection TC with positive and negative matching results.

Definition 7 (*Triples with positive matching results* P_{θ}^{TC})

Given a threshold $\theta \in [0, 1]$ and a test collection TC, then $P_{\theta}^{TC} = \{p_i \in TC \mid \mu(spec_{p_i}, req_{p_i}) \geq \theta\}$ is the set of triples with positive matching results.

Definition 8 (*Triples with negative matching results* N_{θ}^{TC})

Given a threshold $\theta \in [0, 1]$ and a test collection TC, then $N_{\theta}^{TC} = \{p_i \in TC \mid \mu(spec_{p_i}, req_{p_i}) < \theta\}$ is the set of triples with negative matching results.

The service matching outputs a positive matching result for a specification pair, if its service specification complies to the requirements specification to an acceptable extent. A threshold θ determines this extent, for which each matching result with an equal or higher value is considered to be positive. Otherwise, the matching result is considered to be negative. Thus, positive and negative matching results are defined relatively to a certain threshold θ.

Depending on the realization of the service matching, its computed positive and negative matching results might be incorrect. Definitions 9–12 formalize

the sets of triples with true/false positive and true/false negative matching results [163]. These sets are computed for specifications from a test collection, in which their true matching results are known based on the expert knowledge.

Definition 9 (*Triples with true positive matching results* TP_θ^{TC})

Given a threshold $\theta \in [0,1]$ *and a test collection* TC*, then* $TP_\theta^{TC} = \{p_i \in P_\theta^{TC} \mid match(spec_{p_i}, req_{p_i}) \geq \theta\}$ *are triples with true positive matching results.*

Definition 10 (*Triples with false positive matching results* FP_θ^{TC})

Given a threshold $\theta \in [0,1]$ *and a test collection* TC*, then* $FP_\theta^{TC} = \{p_i \in P_\theta^{TC} \mid match(spec_{p_i}, req_{p_i}) < \theta\}$ *are triples with false positive matching results.*

Definition 11 (*Triples with true negative matching results* TN_θ^{TC})

Given a threshold $\theta \in [0,1]$ *and a test collection* TC*, then* $TN_\theta^{TC} = \{p_i \in N_\theta^{TC} \mid match(spec_{p_i}, req_{p_i}) < \theta\}$ *are triples with true negative matching results.*

Definition 12 (*Triples with false negative matching results* FN_θ^{TC})

Given a threshold $\theta \in [0,1]$ *and a test collection* TC*, then* $FN_\theta^{TC} = \{p_i \in N_\theta^{TC} \mid match(spec_{p_i}, req_{p_i}) \geq \theta\}$ *are triples with false negative matching results.*

Based on Definitions 9–12, Definition 13 and Definition 14 formalize the notions of precision and recall correspondingly. Both definitions are based on the definitions of precision and recall given by Salton et al. [126, p. 164] and defined relatively to a threshold θ and a test collection TC.

The matching precision is a measurement, which indicates how precise the service matching retrieves positive matching results for a given test collection [126, p. 164]. It is measured as the ratio of the number of true positive matching results to the number of matching results identified as positive, i.e., true positive and false positive matching results.

Definition 13 (*Matching precision* π_θ^{TC})

Given the sets of matching results TP_θ^{TC} *and* FP_θ^{TC}*, then* $\pi_\theta^{TC} = |TP_\theta^{TC}|/(|TP_\theta^{TC}| + |FP_\theta^{TC}|)$ *is the matching precision.*

The matching recall is a measurement, which indicates how good the service matching can retrieve all matching results considered as positive by the oracle [126, p. 164]. It is a ratio of the number of true positive matching results to the number of true positive and false negative matching results.

Definition 14 (*Matching recall* ρ_θ^{TC})

Given the set of matching results TP_θ^{TC} *and* FN_θ^{TC}*, then* $\rho_\theta^{TC} = |TP_\theta^{TC}|/(|TP_\theta^{TC}| + |FN_\theta^{TC}|)$ *is the matching recall.*

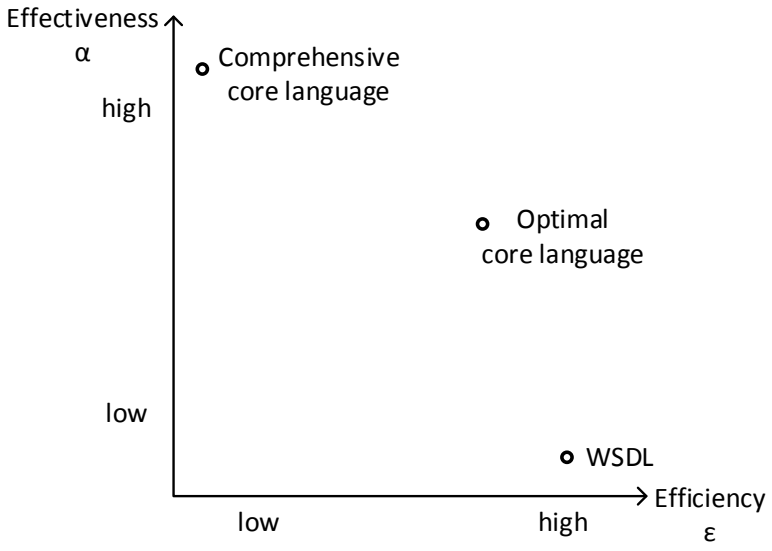


Figure 4.3: Core language optimality wrt. efficiency and effectiveness

Finally, Definition 15 formalizes the effectiveness of the service matching. Effectiveness is a measurement of the quality of the service matching indicating how good the service matching retrieves correct matching results. It is defined as a harmonic mean over the precision and recall [153]. Similar to the matching efficiency, the effectiveness is defined relatively to a realization of the service matching μ, to a test collection TC, and a threshold θ.

Definition 15 (*Matching effectiveness* α_{μ}^{TC})

Given the matching precision π_{θ}^{TC} *and the matching recall* ρ_{θ}^{TC}*, then* $\alpha_{\mu}^{TC} = 2 \cdot \pi_{\theta}^{TC} \cdot \rho_{\theta}^{TC} / (\pi_{\theta}^{TC} + \rho_{\theta}^{TC})$ *is the matching effectiveness.*

The efficiency and effectiveness build a trade-off for the service matching. Assume an OTF market with three languages depicted in Figure 4.3 considered as a core language. Figure 4.3 depicts the comprehensive core language having a very high matching effectiveness, WSDL having a very high matching efficiency and an optimal core language, for which both values are optimized for the underlying OTF market. Having test collections and the service matching defined for these languages, the matching effectiveness and efficiency can be measured according to Definitions 6 and 15.

On the one hand, the higher is the efficiency of the service matching, the lower is its effectiveness (this holds under the assumption that the service matching is always performed for all parts of a service specification and no abortion in the middle of the matching process happens). For example, a comprehensive core language results in a high effectiveness because it covers different structural, behavioral and non-functional service properties, a comparison of which leads to reliable matching results. However, the matching efficiency on its specifications

is low because comparing all these service properties results in a high run time.

On the other hand, a high efficiency of the service matching cannot be achieved simultaneously with its high effectiveness (this holds under the assumption that the service matching is always performed for all parts of a service specification and no abortion in the middle of the matching process happens). The reason is that a high efficiency results from a low run time, which can be achieved, only if specifications contain few service properties without many details.

For example, the well-known specification language Web Services Description Language (WSDL) [28] covers only operations of services. This fact results in a low run time of matching WSDL specifications as no time-consuming matching of either behavioral or non-functional properties is considered. However, this leads to a low effectiveness of the service matching because no reliable matching results can be obtained by comparing the service operations only.

Finally, an *optimal core language* covers less service properties from the comprehensive core language but more service properties in comparison to WSDL. As a result, the optimal core language has a *lower effectiveness* than the comprehensive core language and *lower efficiency* than WSDL, thus, serving the trade-off between the efficiency and effectiveness in a best possible way for the underlying OTF market. A core language is optimal for an OTF market with respect to a realization of the service matching for this core language, a test collection reflecting representative usage scenarios of this language in this market and a threshold defined to measure the effectiveness of the service matching.

Definition 16 gives the notion of an optimal core language of this PhD thesis.

Definition 16 (*Optimal core language CL_{opt}*)

The core language $CL^{opt}_{\mu,TC}$ is pareto-optimal with respect to the service matching μ, a test collection TC, and a threshold $\theta \Leftrightarrow$
$\nexists L$: *Language* ($(\alpha^{TC}_{\mu}(L) \geq \alpha^{TC}_{\mu}(CL^{opt}_{\mu,TC}) \wedge \varepsilon^{TC}_{\mu}(L) > \varepsilon^{TC}_{\mu}(CL^{opt}_{\mu,TC}))$ $\vee$ $(\alpha^{TC}_{\mu}(L) > \alpha^{TC}_{\mu}(CL^{opt}_{\mu,TC}) \wedge \varepsilon^{TC}_{\mu}(L) \geq \varepsilon^{TC}_{\mu}(CL^{opt}_{\mu,TC}))$).

The definition states that a core language is pareto-optimal relatively to matching efficiency and matching effectiveness measured for a given test collection. For the optimal core language, service matching for all other languages cannot achieve equal (or higher) effectiveness with higher efficiency, or higher effectiveness with equal (or higher) efficiency. In other words, improvement in one parameter is impossible without worsening the second one. According to this definition, core languages producing higher effectiveness at lower efficiency or higher efficiency at lower effectiveness also lie on the pareto frontier and are considered pareto optimal (if these also satisfy the conditions stated in Definition 16). The presented definition of the language optimality allows to develop a core language with quality measurable based on the introduced metrics of the efficiency and effectiveness.

The next section presents the approach LOpt for designing a core language optimal with respect to Definition 16.

4.2 The Approach LOpt

This section presents the approach LOpt (Language Optimizer) for the design of an optimal core language with respect to the definitions given in Section 4.1. LOpt consists of a comprehensive core language and a configuration approach. The comprehensive core language integrates existing specification languages that results in its high matching effectiveness but its low matching efficiency (see Figure 4.3). The configuration approach customizes the comprehensive core language so that the resulting customized language optimizes the trade-off between the efficiency and effectiveness for a given OTF market.

Section 4.2.1 introduces an overview of the design phase and the application of the approach. Section 4.2.2 describes the development of the comprehensive core language, which serves as a basis for the configuration. Section 4.2.3 presents the configuration approach that obtains an optimal core language for a given market from the comprehensive core language. Section 4.2.4 illustrates the application of the developed configuration approach including how the configuration approach obtains an optimal core language for a certain OTF market and how this optimal language is used for its automated market operations.

4.2.1 Overview of the LOpt Approach

The overview of the LOpt approach is introduced in Figure 4.4.

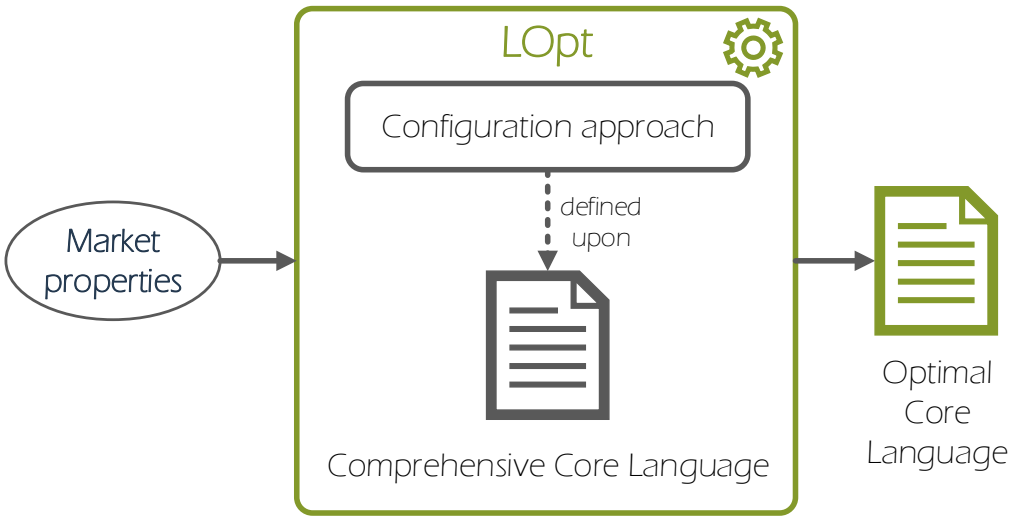


Figure 4.4: Overview of the LOpt approach

LOpt takes market properties as an input and outputs a core language, which is optimal for the service matching in this OTF market according to the definition of the language optimality given in Section 4.1.2. LOpt is an automated approach that requires a manual effort only for formalizing the properties of a

given OTF market. The approach is deterministic and, thus, produces the same optimal core language for the markets with the same properties.

LOpt finds an optimal core language by customizing a `Comprehensive Core Language`. The comprehensive core language is designed in a way to facilitate this configuration. The comprehensive core language describes a variety of structural, behavioral, and non-functional service properties, which enable to create comprehensive service specifications using this language. Different service properties from the comprehensive core language can be used to configure an optimal core language for an arbitrary OTF market. The comprehensive core language reuses existing established languages for service specification that fosters its acceptance by market actors and its compliance to existing languages for transformations into the optimal core languages.

Section 4.1.2 explains that such comprehensive service specifications cannot be optimal for each and every OTF market. Thus, the `configuration approach` considers the specifications of an OTF market described as `Market properties`. In order to obtain an optimal core language for an OTF market, a market actor with the corresponding expertise sets the relevant market properties to describe this market and also their individual strategy. The configuration is performed based on measurable goals leading to the fact that the optimality of a core language can be objectively evaluated. The configuration approach creates the `Optimal Core Language` using the view building mechanism on the comprehensive core language described in Section 4.2.2.5.

To apply LOpt to an OTF market, market properties have to be specified and the configuration approach has to be applied based on them. After an optimal core language has been configured, it can be used as long as the properties for this market do not change. If some changes in the comprehensive core language or the configuration approach occur, the currently used core language might not be optimal anymore. In this case, it is desirable to run LOpt again, in order to obtain the core language optimal with regard to the recent changes.

The comprehensive core language and the configuration approach of the LOpt approach are developed for all OTF markets in the OTF Computing. They can be adapted if changes are necessary. After a language adaptation, those parts of the configuration approach might need to be adapted, which are defined upon the changed language parts. For the adaptation of the configuration approach only, no changes in the comprehensive core language are necessary.

4.2.2 Comprehensive Core Language

This section presents an approach to develop a comprehensive core language for service specifications. The development approach describes, how existing languages are collected for reuse and integrated stepwise in the comprehensive core language. Then, techniques to check the quality of the comprehensive core language are introduced.

4.2.2.1 Overview of the Development Approach

Figure 4.5 illustrates the approach to develop a comprehensive core language. In order to develop a comprehensive core language, three steps are necessary.

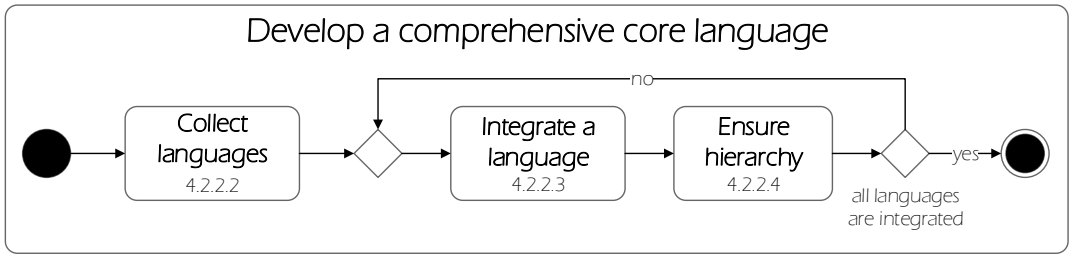


Figure 4.5: Development approach for a comprehensive core language

This approach is a set of guidelines for a language engineer, who can apply them to systematically obtain a comprehensive core language for some application domain. This PhD thesis applies this approach to the domain of the service specification, however it can be applied to other application domains as well. The approach assists a language engineer to systematically perform language integration, while ensuring a package hierarchy of the integrated language. This results in fewer errors during the integration and a language structure that serves as a solid basis for the configuration. In comparison, in a non-systematic approach, a language engineer would integrate languages and language constructs in an arbitrary order that would unnecessary complicate the integration process and, thus, result in an error-prone integration approach.

The first step `Collect languages` of the development approach aims at identifying service specification languages for the reuse in the comprehensive core language. A language engineer collects existing established languages and identifies the set of service properties for the comprehensive core language.

The step `Integrate a language` aims at integrating the languages collected in the previous step into the comprehensive core language. The language engineer performs the integration stepwise starting with the structure of the languages and continuing with their language constructs.

After a successful language integration, the language engineer ensures the quality of the obtained language structure. The step `Ensure hierarchy` introduces methods to check and improve this quality. The language engineer proceeds with the process until all identified languages are integrated into the comprehensive core language.

4.2.2.2 Collect Languages

Figure 4.6 presents the step `Collect languages` refined by two substeps. In the first step, a language engineer chooses existing languages for the reuse in the comprehensive core language. Their language structure is also adapted, in

order to facilitate the later configuration of the comprehensive core language. In the second step, the language engineer prioritizes the languages and uses the language with the highest priority as an initial for the language integration.

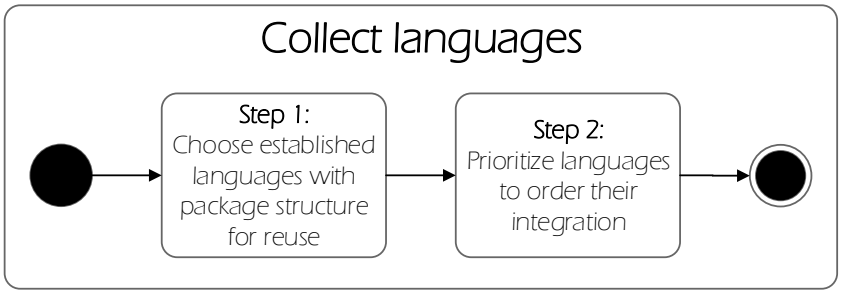


Figure 4.6: Step to collect languages for reuse

Collect languages: Step 1 The language engineer starts by choosing which specification languages to reuse in the comprehensive core language. Based on her expertise, the language engineer collects established languages with the same goal as the comprehensive core language, i.e., service specification in OTF markets. Ideally, the chosen languages shall be comprehensive, i.e., describe structural, behavioral, as well as non-functional service properties. The language engineer spares the effort by choosing comprehensive languages, but it is not a hard prerequisite as languages, which focus on a certain service property, may refine parts of the comprehensive core language.

The approach LOpt customizes the comprehensive core language to obtain an optimal core language. Akehurst et al [3] claim that a modular structure of a language facilitates its customization. A *modular structure* realizes the principle of separation of different concerns in a language. Its main advantage is that parts of the language can be defined, refined, tested, maintained, and reused separately. To facilitate the configuration in LOpt, the comprehensive core language shall have a modular structure that allows to configure on the level of its language parts as well as their language constructs.

Since the comprehensive core language is built by reusing established specification languages, their modular structures have to be reused as well. This PhD thesis considers languages with their abstract syntax defined in the form of a metamodel as motivated in Section 4.1.1. Such languages usually have a modular structure realized by *packages* [71], which are used to partition and extend metamodels. Packages group language constructs representing related modeling elements with the purpose to reduce the complexity of a language and to facilitate its reuse. Packages can import each other, where all language constructs of the imported package become visible for the importing package.

The approach LOpt leverages the package structure for the configuration of the comprehensive core language. The configuration can start at the level of

packages and continue at the level of language constructs. This is how groups of language constructs contained in a package can be selected or omitted for the optimal core language. This method leads to less effort in comparison to the case, when every language construct has to be considered separately.

Formalization with Ecore In order to formally describe the abstract syntax of domain-specific languages (DSLs) in the form of metamodels, this PhD thesis considers the modeling language Ecore [141]. Ecore enables to define the package structure of a language as well as language constructs of the packages. Figure 4.7 presents the excerpt from the abstract syntax of Ecore. Since Ecore realizes Essential Meta Object Facility (EMOF) and MOF is self-describing, this abstract syntax is described in Ecore as well.

Figure 4.7 shows that an Ecore metamodel consists of a root package being an object of the type `EPackage`. Each package consists of a set of subpackages (see `eSubpackages`) and a set of metamodel classes (see `eClassifiers`).

Metamodel classes are represented by objects of the type `EClass`, which inherits from the class `EClassifier`. A metamodel class contains a set of attributes and references (see `eStructuralFeatures`). The class `EAttribute` models class attributes, while the class `EDataType` models attribute types.

The class `EReference` stands for containments and associations. The attribute `containment:EBoolean` indicates, whether a reference is a containment (`containment=true`) or a reference is an association (`containment=false`). References belong to their source class (see the association `eContainingClass`). The association `eReferenceType` points to the target class of a reference. The inheritance relations of a class are represented by the class `EGenericType`. Objects of this type point at a superclass of the given class using the association `eClassifier` to the class `Classifier`, which the class `EClass` inherits from. The class `EGenericType` is contained in the class `EClass`.

Running example As a running example, a comprehensive service specification language was chosen, which abstract syntax and its package structure are formally defined using Ecore. For that, the Unified Service Description Language (USDL) [15] is introduced in detail. USDL focuses on describing technical and commercial information about services.

Figure 4.8 shows the package structure of USDL. The USDL metamodels are described in detail in [15]. USDL consists of packages, which do not nest any further packages and consist of classifiers only.

Collect languages: Step 2 In `Step 2` illustrated in Figure 4.6, the language engineer has to prioritize the collected languages according to her expertise. The prioritization is primarily guided by the operations defined for the languages, a formal semantics definition existing for them, and the extent of their reuse.

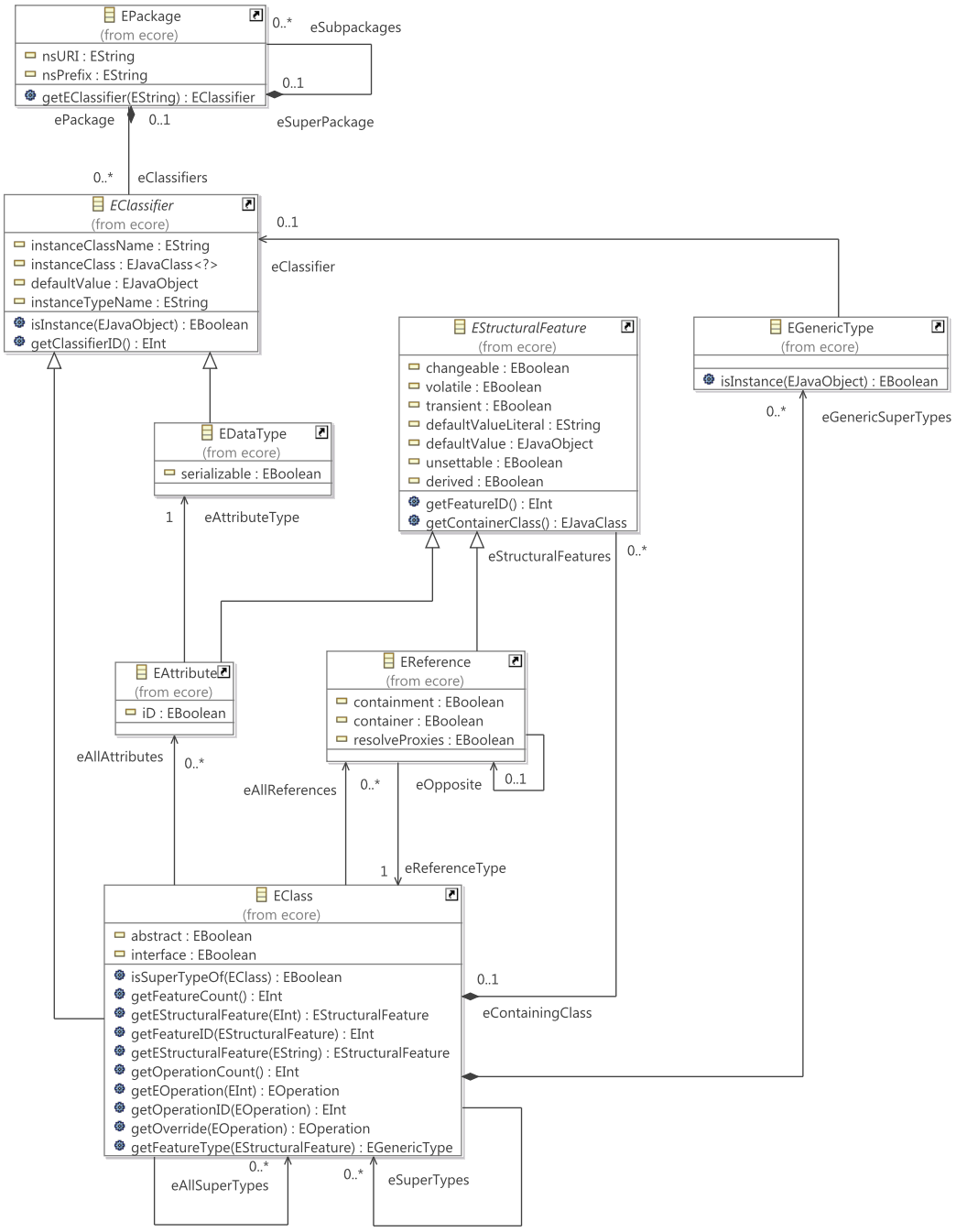


Figure 4.7: Excerpt from the abstract syntax of the Ecore modeling language

It determines the order of languages for the reuse in the comprehensive core language. The highest priority language is taken as a basis for the integration.

During the reuse, the language constructs of the language with the highest priority will be extended by language constructs of the languages with lower

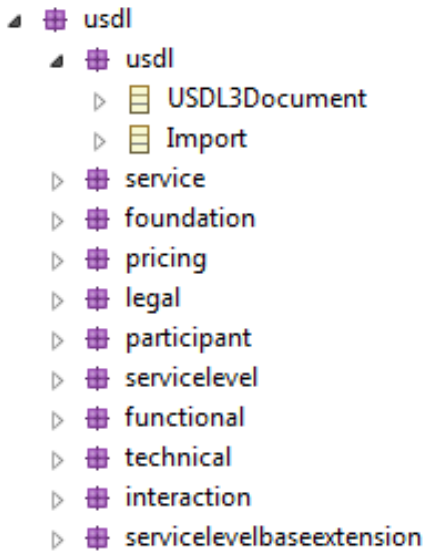


Figure 4.8: USDL package structure

priorities. As a result, the operations of the highest priority language will be preserved to a large extent but might need a modification with respect to new or extended language constructs. The operations of the lower priority languages would need to be partially defined anew for the language constructs, which are integrated into the language constructs of the highest priority language. The operations of the lower priority languages could also be extended by the new language constructs of the integrated language, in order to foster their reuse.

4.2.2.3 Integrate a Language

In the step `Integrate a language` introduced in Figure 4.5, the language engineer stepwise integrates the initial set of packages with other languages according to the order of their priorities. The integration is performed for the first level packages, then for their nested subpackages, and then for their language constructs. For that purpose, the order of the package integration is determined first. Then, the language constructs of these packages are integrated one after another into the packages of the comprehensive core language. After language constructs of all packages have been successfully integrated, the process finishes.

The language engineer performs the integration based on her knowledge of the semantics of the collected language. The language engineer knows the semantics of the languages based on their informal or formal language definitions. An informal language definition may be a textual specification, which gives insights into the meaning of language constructs. A formal language definition is a specification using some formalism that may be used to automatically check language properties. With the help of these semantics definitions, the language engineer solves conflicts arising for language constructs during the integration.

Figure 4.9 shows the step `Integrate a language`.

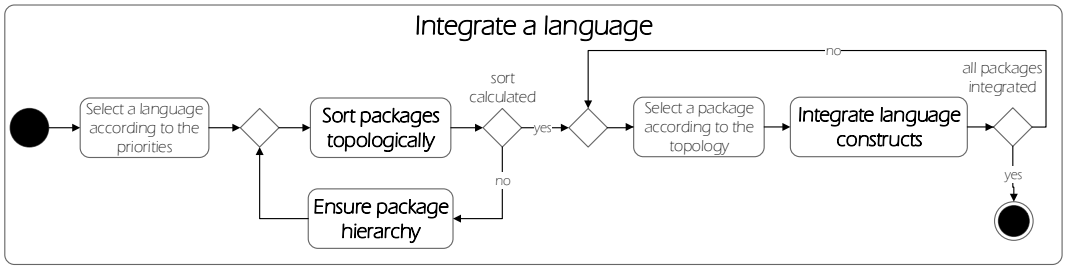


Figure 4.9: Step to integrate a language

The step begins with selecting the next language for the integration according to the priorities assigned in the step `Collect languages`. Then, a topological sort of the packages of the selected language is calculated in the step `Sort packages topologically` to determine the order of their integration. The calculated order helps the language engineer to integrate a language systematically in comparison to the case, when packages are integrated in an arbitrary order. The language engineer can integrate metamodel classes with their containments and inheritance relations easier, because the target classes of these dependencies are already integrated. This leads to a less error-prone integration compared to a non-systematic approach.

Topological sort of packages The topological sort builds upon dependencies between packages. The package p_1 builds upon the package p_2, if at least one metamodel class from p_1 either contains a metamodel class from p_2 or inherits from a metamodel class from p_2. Associations are not considered for the sort calculation, because they result in a case, when no correct topological sort can be obtained due to cyclic dependencies. Cyclic dependencies of associations is a rather common case in the language design that leads to the decision not to consider them for the sort calculation. The consequence of that decision is the fact that some of association dependencies cannot be integrated directly because the integration of their target classes might be unfinished yet. In this case, the language engineer has to integrate such associations later.

The algorithm calculates a topological sort automatically by first collecting all packages of the language. For each package, a set of packages, which this package depends on regarding containments, and a set of packages, which this package depends on regarding inheritance, are calculated. For that, the targets of all containments and of all inheritance of this package are checked. If the target belong to another package, then this package is added to the corresponding set.

The calculation continues by identifying a set of packages, which are independent from other packages based on both containments and inheritance. Based on the set of independent packages, further sets containing packages depending only of the packages of the previous sets are built. This process continues until

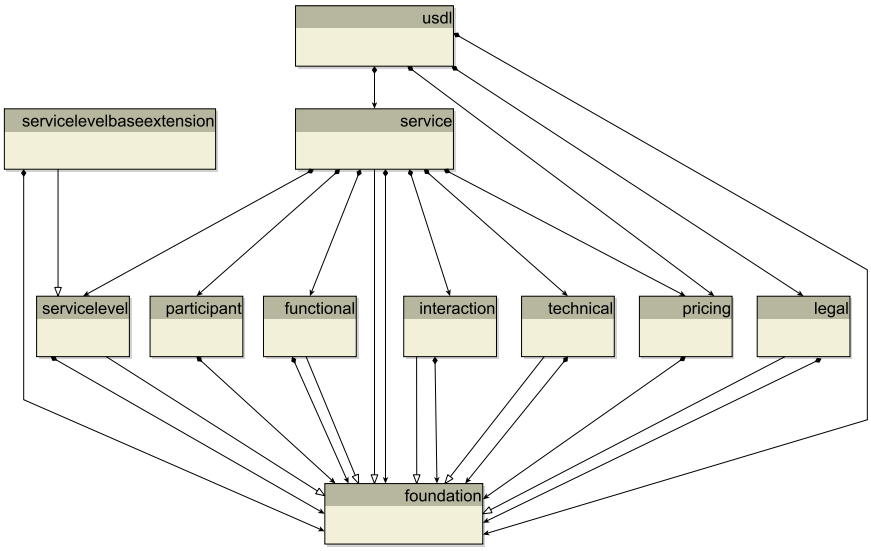


Figure 4.10: Topological sort of USDL packages

all sets are formed and each set in the calculated order depends to the previous sets only. If any of the sets appears to be empty, then a correct topological sort does not exist and the algorithm finishes with the error message that no topological sort can be calculated.

As an example, the algorithm for the topological sort is applied to the packages of USDL depicted in Figure 4.8. Figure 4.10 visualizes the containment and inheritance dependencies for its packages as a directed graph. In this graph, nodes represent the packages, while edges represent the dependencies. As a result, the following order is obtained: `sortedPackages` = <`foundation`, `servicelevel`, `participant`, `functional`, `interaction`, `technical`, `pricing`, `legal`, `servicelevelbaseextension`, `service`, `usdl`>.

For certain languages, a topological sort of packages cannot be calculated, because cyclic dependencies of containments or inheritance exist, and it is impossible to obtain an explicit order of the packages because the packages are dependent on each other. In order to obtain a correct topological sort, a hierarchy of the language packages has to be ensured. A hierarchy means that no cyclic dependencies between the packages exist, i.e., there are no two packages p_1 and p_2, for which at least one metamodel class from p_1 contains a class from p_2 or inherits from a class from p_2, and at least one class from p_2 contains a class from p_1 or inherits from a class from p_1. If a topological sort cannot be calculated, the language engineer has to perform the substep `Ensure package hierarchy` described in Section 4.2.2.4 to constructively resolve cyclic dependencies in the language (see Figure 4.9). The language engineer performs this substep until a correct topological sort can be calculated.

Integrate language contructs The comprehensive core language shall be orthogonal as stated in the requirement *R.1.1.7* from Section 3.1.1. According to Paige et al. [113], *orthogonality* means that each concept has to be modeled in the language in exactly one good way, i.e., no language constructs describing the same concept several times exist. In order to realize the orthogonality, for each package according to the presented order, its language constructs are integrated into the packages of the comprehensive core language. The language engineer integrates the language constructs based on their semantics and her expertise in languages of the considered application domain. The substep `Integrate language constructs` illustrated in Figure 4.9 provides the language engineer with detailed systematic guidelines for the integration.

As depicted in Figure 4.9, the step `Integrate language constructs` is repeated for all language constructs in this package, and for all packages of the language to be integrated. As a result, the comprehensive core language is created by reusing established languages. During the integration, mappings between the integrated languages and the comprehensive core language emerge. These mappings are used later for transforming specifications written in the integrated languages into specifications in the comprehensive core language. The method of user-friendly model transformations introduced in Chapter 5 is applied only to new languages having no already existing mappings.

Topological sort of classes The integration procedure of language constructs starts with calculating a topological sort of classes in the selected package, in order to determine their integration order. This is performed similar to calculation of a topological sort for packages. A topological sort is obtained for inheritance and containment dependencies between metamodel classes within one package. Dependencies between classes from different packages are already considered in the package sort. The metamodel class c_1 depends on the metamodel class c_2, if either c_1 inherits from c_2 or c_1 has a containment with the target at c_2.

This order helps the language engineer to integrate classes according to their dependencies among each other, i.e., when the language engineer adds containment and inheritance dependencies, their target classes are already integrated in the package. This fact results in a systematic integration approach in comparison to a non-systematic approach, when the integration of language constructs is performed in an arbitrary order.

Figure 4.11 shows an excerpt of the USDL package `foundation`, whose language constructs are sorted topologically. The calculate topological order is: `sortedClasses` = <`AddressItem`, `CopyrightProtectedElement`, `Description`, `DependencyTarget`, `Expression`, `FunctionalElementRef`, `ServiceLevelElementRef`, `TimeEntity`, `Artifact`, `IdentifiableElement`, `Location`, `Time`, `Classification`, `ElectronicAddress`, `PhysicalLocation`, `TimePattern`>.

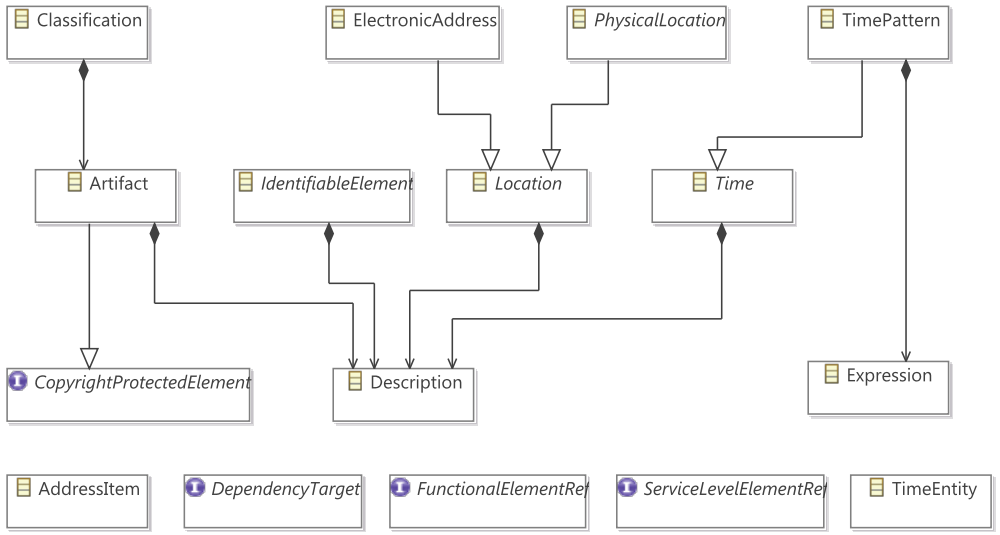


Figure 4.11: Excerpt from the USDL package `foundation`

Integrate classes According to the calculated order, the language engineer checks each class. If the language engineer can find an existing class for the reuse, then they integrate both classes into one. The integration extends an existing class with properties of a new class, integrating attributes and references of the classes. Otherwise, if the language engineer can find a packages of the comprehensive core language for the reuse, then the class and its properties (attributes and references) are added to this package with dependencies to existing classes, if necessary. If such package does not exist, a new package is created in the comprehensive core language, and the class is added to this package with dependencies to existing classes, if necessary.

The integration of two metamodel classes follow the guidelines below. The leading class in the integration is the class from a higher priority language according to the prioritization described in Section 4.2.2.2. Thus, class names or their granularity are preserved from the higher priority language. The attributes of an existing class are extended with either lowest cardinalities or new attributes. As the granularity of the language with the highest priority has to be preserved, no refinement or grouping of existing attributes is performed.

For the integration of enumerations as a special case of metamodel classes, the language engineer adds all literals, which are not a part of the existing enumeration yet. In case of conflicts, the language with the higher priority leads. The integration of both classes and attributes avoids that several language constructs represent the same concept in the comprehensive core language. As a result, the desired orthogonality of the integrated language is supported.

After the attributes have been integrated, dependencies have to be integrated as well. The integration starts with the inheritance and continues with containments. The target classes of the dependencies have already been integrated

due to the topological order of the class integration. The integration procedure continues with associations. For all kinds of dependencies, for each dependency of the class to integrate, the language engineer checks whether a dependency to reuse exists. In this case, the cardinalities of the existing dependency are set to the lowest possible. If no dependency for the reuse can be found, then the language engineer adds the dependency. For associations, a check whether the target class exists is required due to the fact that association dependencies are not considered in the topological sort.

Semantics of the integrated language The semantics definition of the integrated language results from the informal and formal definitions of its constituent languages. The language definition of the language with the highest priority serves as a basis. The existing language definition has to be extended with the semantics of the new language constructs, and the definition of the extended language constructs has to be modified.

The semantics can be formally defined using the Dynamic Meta Modeling (DMM) approach [133, 132]. The authors propose the Test-Driven Semantics Specification (TDSS) approach to create DMM specifications with high quality [133]. TDSS starts by creating a DMM specification formally defining the semantics of a given language. In order to check the quality of this specification, a test suite is defined. Each test consists of an example model and its expected semantics described as traces of execution events. The actual semantics of test models is calculated based on the created DMM specification. Then, the actual and expected semantics are compared indicating the quality of this specification.

A DMM specification assists in solving semantic conflicts arising during the language integration. For that, the existing DMM specification for the language with the highest priority has to be extended with regard to the new and modified language constructs. In addition, the test suite has to be modified and extended, as well. Based on the new DMM specification for the integrated language and the new test suite, the quality of the semantics definition is checked. For every test case, for which the actual semantics differs from the expected one, either the DMM specification or the test suite are improved. As a result, the semantics of the integrated language is improved regarding conflicts and is checked for having high quality.

4.2.2.4 Ensure Package Hierarchy

This step aims at constructively solving cyclic dependencies between packages or classes prohibiting the calculation of a correct topological sort. This leads to a higher quality of the language design in comparison to the case, when cyclic dependencies exist. A correct topological sort leads to a package or class hierarchy, in which each package or class builds upon one or several other packages or classes having no backward dependencies to this one. If two dependencies (whether inheritance or containment) exist between two packages or classes x_1

and x_2 so that one dependency goes from x_1 to x_2, while the second one goes from x_2 to x_1, then the package hierarchy is not given.

According the the syntax definition of Ecore, no cyclic inheritance or containment dependencies between two metamodel classes are allowed. Cyclic dependencies of a combination of inheritance and containment are possible for metamodel classes as Ecore does not check cyclic dependencies over different types of dependencies. A language engineer has to resolve such dependencies constructively because the language engineer is an expert in her language and, thus, considers both the syntax of the language and its semantics. For packages, the goal is to redistribute the classes between the considered packages so that all existing cyclic dependencies are solved and no new cyclic dependencies are introduced. For classes, the goal is to modify the cyclic dependencies by moving them to other classes or by introducing new classes as their targets.

4.2.2.5 View Building

This section introduces a view building mechanism for specification languages having a formal language definition in Ecore. LOpt uses this mechanism for the configuration of the comprehensive core language (see Figure 4.4).

According to Goldschmidt et al. [56], a *view type* is a set of metamodel classes, whose instances can be displayed to a modeler by a view. A view type is defined at the metamodel level, i.e., for the abstract syntax of a language, while a view is an application of the view type to specifications in this language. The configuration approach creates an optimal core language as a view type containing a subset of the language constructs of the comprehensive core language required for the service matching in a given OTF market.

Figure 4.12 presents a formal definition of a view type given in the form of an Ecore metamodel. A view type is defined for languages specified in the Ecore modeling language presented in Figure 4.7. A view type replicates the package structure of a language but contains only a subset of its packages and their language constructs. Furthermore, language constructs in a view type can be reduced by their attributes or references. As a result, a view type consists of placeholders, which represent real packages and metamodel classes, which however can contain less classes, attributes or references.

A `ViewType` contains a set of placeholders for packages (`EPackagePlaceholder`), placeholders for classifiers (`EClassifierPlaceholder`), placeholders for attributes (`EAttributePlaceholder`), and placeholders for references (`EReferencePlaceholder`). Each of these placeholder types refers to the corresponding Ecore class standing for packages (`EPackage`), classifiers (`EClassifier`), attributes (`EAttribute`), and references (`EReference`). Using placeholders, a metamodel class can be added to a view type without all its attributes or references. Similarly, not all classes of package have to be added to its placeholder and, thus, to a view type. If a view type contains two classes, one of which is a superclass of another one, then the inheritance is automatically added between

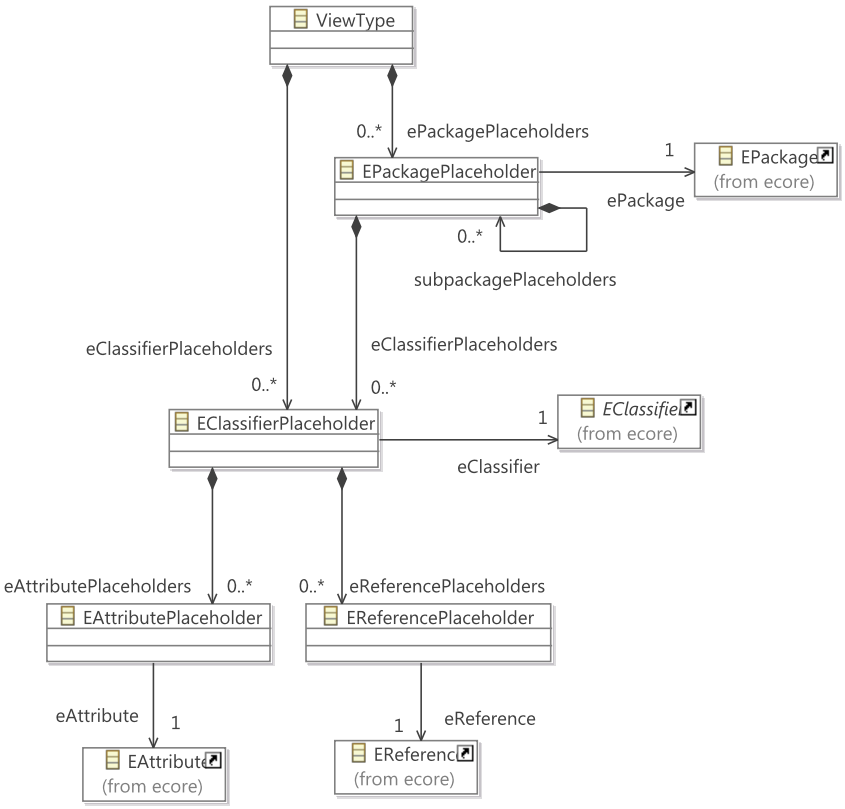


Figure 4.12: Definition of a view type

these classes. The information about the nested structure of package placeholders is modeled in the view type by the containment `subpackagePlaceholders`.

A language engineer creates a view type by adding the corresponding placeholders and selecting the language constructs for them. The language engineer performs the selection of the placeholders stepwise. At the beginning, the language engineer adds a package placeholder to the view type and selects a package for it. Then, she adds classifier placeholders to the package placeholder and selects classes for them. Finally, the language engineer adds attribute placeholders and reference placeholders to the classifier placeholders.

After the selection is done, the algorithm for building a view type creates a view type as an instance of the metamodel in Figure 4.12. During the creation, it is guaranteed that the package, subpackage or class containing the added packages, classes, attributes and references are automatically added to the view type as well. For example, if a class is added to the view type, then its package has to be added to the view type as well. The algorithm also ensures that the created view type is *well-formed*. For that, for all classes of the view type, it is checked that the target classes of their reference belong to this view type as well. The algorithm for building a view type reveals possible errors and feedbacks them to the language engineer.

4.2.3 Configuration Approach

This section describes the development of the configuration approach introduced in Section 4.2.1. The configuration approach customizes the comprehensive core language developed in Section 4.2.2 to obtain an optimal core language for an OTF market. Section 4.2.3.1 presents an overview of the configuration approach. Section 4.2.3.2 introduces how the properties of an OTF market are formalized for the configuration. Section 4.2.3.3 shows the formalism for describing the configuration logic. Section 4.2.3.4 illustrates the configuration knowledge base containing the configuration logic specified using the presented formalisms. Section 4.2.3.5 describes the configuration procedure performing the configuration upon the configuration knowledge base.

4.2.3.1 Overview of the Configuration Approach

Figure 4.13 introduces the configuration approach of LOpt. It consists of a configuration procedure, a configuration knowledge base, and two formalisms for modeling market properties and for modeling configuration rules.

The configuration procedure is responsible for creating an optimal core language based on the properties of a given OTF market. The configuration procedure takes as an input formalized market properties, which represent the characteristics of an OTF market relevant for finding an optimal core language. As an output, the configuration procedure produces an optimal core language for the described market based on the configuration logic from the configuration knowledge base.

The configuration knowledge base contains the configuration logic specified using the formalisms for market properties and configuration rules. The knowledge base is defined upon the comprehensive core language and describes how its parts have to be configured depending on a certain market property.

4.2.3.2 Formalization of Market Properties

In order to leverage the properties of OTF markets in the configuration approach, the relevant market properties have to be identified and formalized. The chosen market properties have to influence the service matching on specifications written in a certain core language. Changes in a market property for a OTF market should result in a need for a new optimal core language for it.

Main advantages of a formal representation of every OTF market are eliminating the ambiguity in the specification of market properties and allowing their automatic processing. The formal representation of OTF markets fosters their standardization, which enables a unified specification of different OTF markets. It is possible to categorize OTF markets, where each category groups OTF markets with the same formal market properties. For a market category, the same optimal core language can be used for all its OTF markets.

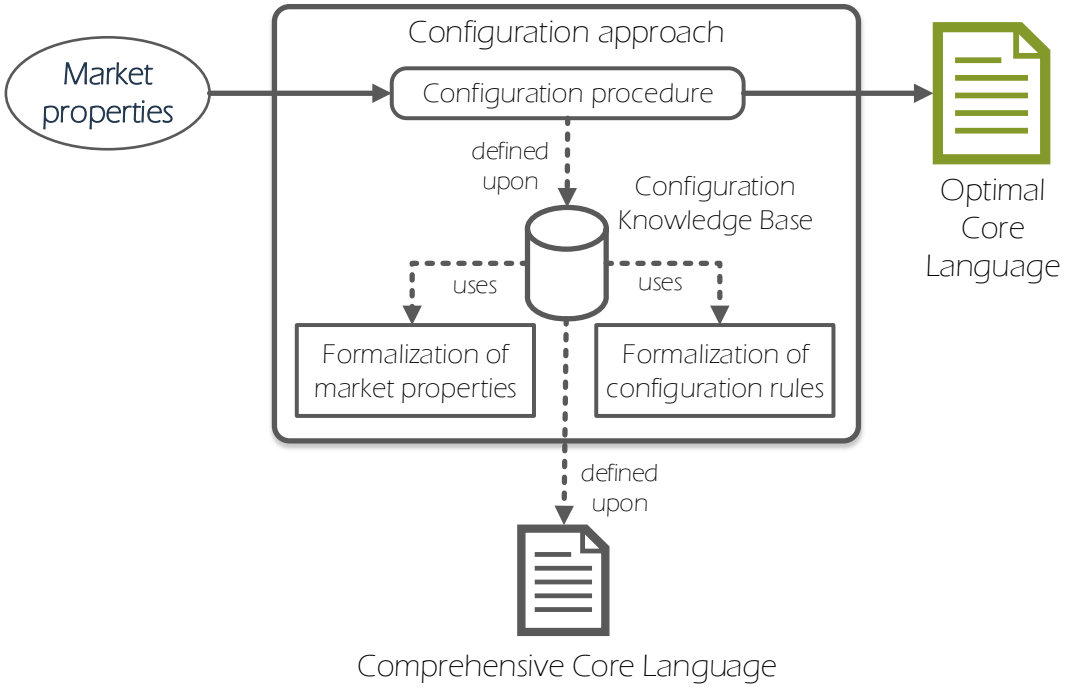


Figure 4.13: Overview of the configuration approach

This PhD thesis proposes an initial set of market properties. In our research, we used these properties to formalize several OTF markets and to compute optimal core languages for them [7, 8]. This set can be extended by further properties or the properties can be adapted during further studies of OTF markets.

The following seven market properties belong to this initial set:

- Standardization in an OTF market (`Standardization`);
- Size of a OTF market (`Market size`);
- Sensitive data involved in a OTF market (`Sensitive data`);
- Complexity of services in a OTF market (`Service complexity`);
- Way to profit in a OTF market (`Profit`);
- Trade-off focus for service matching (`Trade-off focus`).

Figure 4.14 presents the formalization of the identified market properties described in the form of an Ecore metamodel.

Each market property is represented as a concrete metamodel class, e.g., `Standardization`, inheriting from the abstract class `MarketProperty`. Each market property is characterized by its name and its range. The names are already presented above and equal to the names of the classes representing market properties. The range is a set of values, which can be assigned to characterize

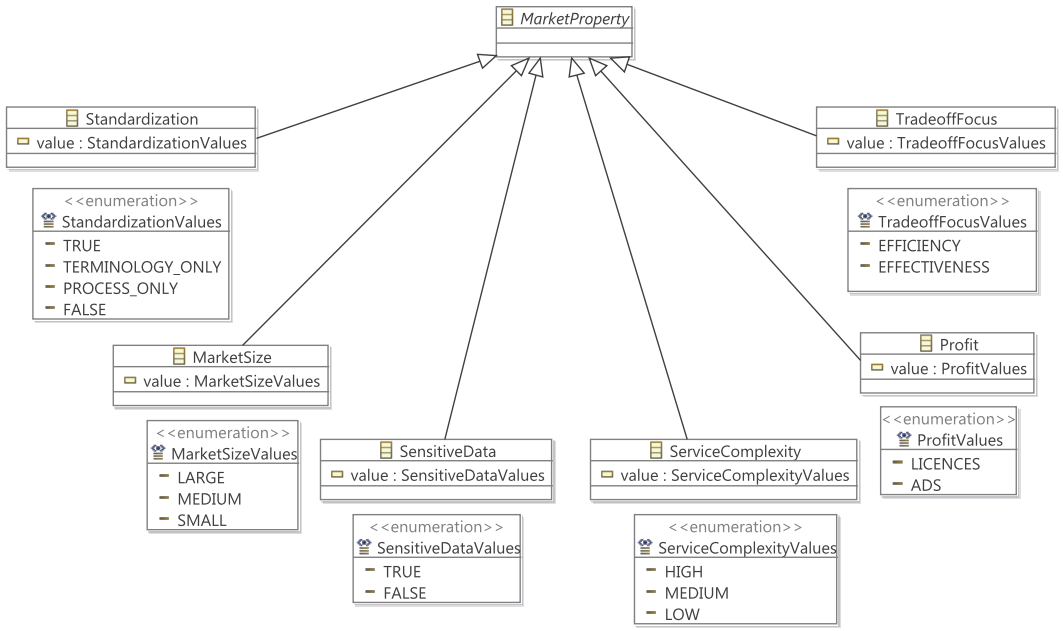


Figure 4.14: Formalization of market properties

a corresponding market property. Therefore, each metamodel class representing a market property has an attribute `value`. The type of this attribute is represented by an enumeration class containing the set of range values.

The market properties, their purpose and range values are explained in detail in the following.

The market property `Standardization` defines the degree, to which service specifications existing in an OTF market are standardized with respect to the terminology and processes. This market property influences the expressiveness of service specifications necessary for the matching. For example, if an OTF market is characterized by an established terminology and standardized processes, service specifications do not have to describe the structure and behavior of services in detail. The reason is that even simple service matching based on names delivers a high effectiveness in such an OTF market, because names are standardized and understood the same. Depending on the level of standardization, the suitable level of detail of the optimal core language that leads to both effective and efficient matching in the OTF market can be found.

The range of the market property `Standardization` is set by the enumeration `StandardizationValues` containing the values `TRUE`, `TERMINOLOGY_ONLY`, `PROCESS_ONLY`, and `FALSE` forming the range for `Standardization`. The range value `TRUE` stays for both the established terminology and standardized processes in an OTF market. The range value `TERMINOLOGY_ONLY` indicates that an OTF market has an established terminology but lacks a standardization of processes. In contrary, the range value `PROCESS_ONLY` is assigned for an OTF

market that has standardized processes, but lacks a standardization of its terminology. Finally, the range value `FALSE` stays for no standardization of either terminology or processes in an OTF market.

For assigning a value for `Standardization`, the existence of ontologies in the considered OTF market has to be investigated. Based on ontologies, the degree of terminology standardization can be determined. An ontology models the concepts of an OTF market in a formal way using, in particular, the Web Ontology Language OWL [93]. The concepts are represented by their names and relations. Thus, an ontology represents the terminology of an OTF market. Since each ontological class models a concept, it indicates a common understanding of that concept in an OTF market. If several ontological classes connected by an equality or synonym relations model a certain concept, it indicates that different naming variations exist for the same concept.

To quantify the degree of the terminology standardization, one possible metric is the amount of classes, which have close similarity values calculated using an ontology matching approach [43]. Many ontologically equal classes indicate that the same concept is modeled in several ways in the ontology. Thus, the terminology is not standardized. Otherwise, if the terminology is standardized, then one ontological class models exactly one concept in an OTF market.

Regarding the standardization of processes, different standards and ways of their description exist. In an OTF market, one example of standardized processes are templates for service compositions [160]. Such templates standardize the control and data flow of composed services realizing a certain kind of functionality. Thus, if a service requester searches for a concrete functionality of this kind, a suitable composed service is built based on a standardized process specified in the template.

The market property `Market size` represents the amount of provided services in an OTF market. The more provided services are offered in the market, the less efficient is the service discovery because all these services have to be matched with a given requirements specification. Thus, using this market property, the decision can be made about what level of detail of the optimal core language leads to an acceptable efficiency with respect to the amount of provided services.

The market property `Market size` has three values in its range. The range value `LARGE` stands for an OTF market having a lot of provided services. An example is the OTF market of tourism including, in particular, services for hotel or flight booking. This is an established OTF market having a high service demand and, thus, a lot of service providers offering different services. The range value `SMALL` stands for an OTF market having few provided services. Such markets either trade rare specific services or exist for a short period of time and have not yet developed to a larger size. An example of a small OTF market is the market of services for university management. Since this market is relatively new and existing services are highly customized for a concrete university, only few services are provided. The range value `MEDIUM` is assigned for an OTF market having the size in between large and small.

The market property `Sensitive data` indicates whether services provided in a OTF market operate on sensitive data. In this case, the importance of service privacy increases in such a market. Using this market property, the decision can be made about what privacy specification in the optimal core language leads to both effective and efficient service matching.

The next market property `Sensitive data` has two range values: `TRUE` and `FALSE`. The range value `TRUE` indicates that services in an OTF market process sensitive data. The range value `FALSE` indicates the opposite. According to privacy law, the so-called personally identifiable information (PII) is classified as sensitive data. According to the National Institute of Standards and Technology (NIST) [110], PII is defined as any information maintained by an institution, which can be used to identify an individual [92]. Examples of PII are a name, an address, a driver's license number, or an image of an individual.

The market property `Service complexity` formalizes the degree of complexity of the services provided in an OTF market. For complex services, a detailed specification of their behavior is necessary as a complex functionality is encapsulated in the service. Thus, using this market property, the decision is made about how detailed the service has to be specified in an optimal core language, in order to perform both efficient and effective service matching.

This property has three range values. The range value `HIGH` indicates that services in an OTF market provide complex functionality, while the value `LOW` indicates that service provide simple functionality. The value `MEDIUM` is assigned, when the complexity of services is in between high and low. To measure the service complexity, existing metrics for complexity evaluation of web service interfaces [130] or of service-oriented architectures [66] can be used.

The market property `Profit` defines how service providers make profit with their services. It influences the fact, whether the service price has to be described in the optimal core language. If service providers profit by selling licenses for their services, then payment mechanisms have to be described. Using this market property, it can be decided about what specification of price in the optimal core language results in both effective and efficient service matching.

The market property `Profit` is defined by two range values. The value `LICENSES` specifies that service providers make profit based on licenses for their provided services. This is a traditional payment model, which is used for the software provided for sell [111]. A user can either buy a license and use the service unlimited, or rent a service for some rental fee and use it unlimited during the given rental period [47]. Also the new pay-per-use models are becoming more popular [11]. The value `ADS` stands for services provided for free, where service providers make profit using advertisement. This payment model is especially relevant for mobile applications, which are often provided for free in app stores but have different ads embedded in their user interface.

The market property `Trade-off focus` is a special property, which refers to the trade-off between the efficiency and the effectiveness of the service matching defined for the optimal core language of an OTF market (as described in

Section 4.1.2). This market property is used for solving conflicts during the configuration, when contradictory decisions regarding the level of detail of the optimal core language result in a conflict. It has two range values `EFFICIENCY` and `EFFECTIVENESS`. The range value `EFFICIENCY` is assigned, then the configuration has to follow the strategy to keep the efficiency as high as possible. The range value `EFFECTIVENESS` is set, when the configuration has to keep the effectiveness of the service matching as high as possible.

4.2.3.3 Formalization of Configuration Rules

Figure 4.15 presents the second formalism of the configuration approach aiming at formalizing configuration rules in the form of an Ecore metamodel.

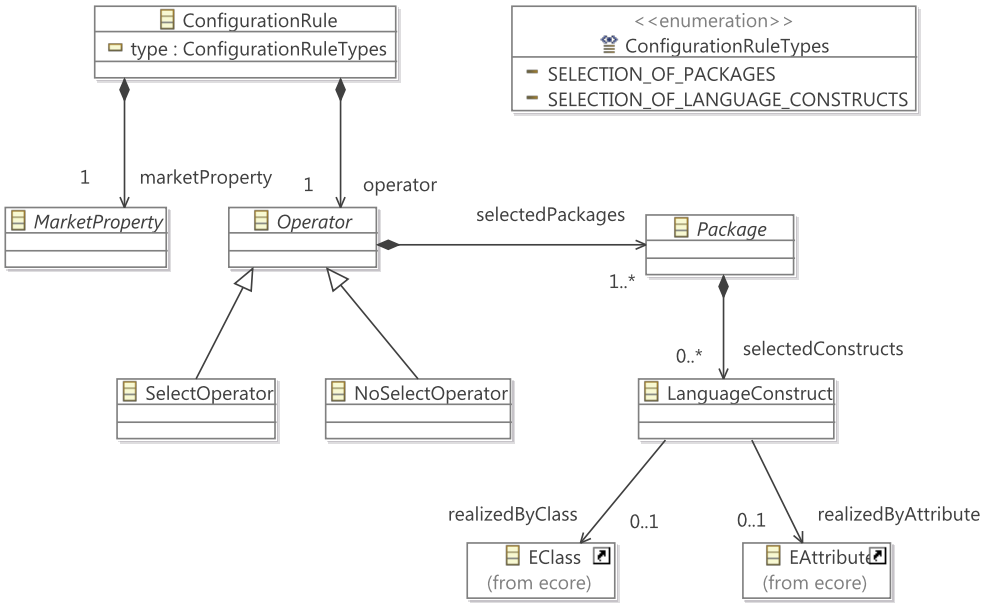


Figure 4.15: Formalization of configuration rules

A configuration rule describes, how the comprehensive core language has to be customized depending on a certain value of a certain market property. The changes on the comprehensive core language can be of two types. Firstly, certain packages from the comprehensive core language can be selected to be a part of an optimal core language. Secondly, language constructs can be selected to be a part of a certain package of the optimal core language. This conforms to the view building defined for the comprehensive core language, which allows to define a view type at the level of language packages and at the level of language construct within a package.

In Figure 4.15, the metamodel class `ConfigurationRule` models a configuration rule. A configuration rule can be of type `SELECTION_OF_PACKAGES` or `SELECTION_OF_LANGUAGE_CONSTRUCTS`. This is modeled by the metamodel class

ConfigurationRule having the attribute type with a value from the enumeration ConfigurationRuleTypes. Each configuration rule is defined for exactly one market property with an assigned range value. This is represented by a containment marketProperty from the class ConfigurationRule to the abstract class MarketProperty. Configuration rules can be defined for all market properties including all their range values presented in Figure 4.14.

Two different operators describe configuration actions on the comprehensive core language. A configuration rule is always defined for exactly one operator that is represented by a containment operator between the classes ConfigurationRule and Operator. These are a select operator (SelectOperator) and a no-select operator (NoSelectOperator). A select operator is responsible for a selection of packages or language constructs, while a no-select operator is used, when certain packages or language constructs have to be omitted.

Parameters of both operators are modeled in Figure 4.15 by the containment selectedPackages between the classes Operator and Package. Packages, which have to be selected or omitted in the optimal core language, have to be specified. Furthermore, it can be specified, which language constructs have be selected or omitted in a certain package (see selectedConstructs between the classes Package and LanguageContruct). Language constructs are realized by Ecore metamodel classes or their attributes as modeled by the associations realizedByClass and realizedByAttribute from the class LanguageContruct to the corresponding classes EClass and EAttribute.

Listing 4.1 shows a configuration rule cr with the type SELECTION_OF_PACKAGES.

Listing 4.1: Example configuration rule

```
1  cr: ConfigurationRule
2  cr.type = SELECTION_OF_PACKAGES
3
4  cr.marketProperty = mp
5    mp : Standardization
6    mp.value = TRUE
7
8  cr.operator = op
9    op : SelectOperator
10   op.selectedPackages = {fp}
11     fp : foundation
12     fp.selectedContructs = {Description, type}
```

The configuration rule cr is defined for the market property mp of the type Standardization. This market property has the range value TRUE indicating that this rule applies in an OTF market, which has both standardized terminology and processes. The rule is defined with the select operator op. This operator selects the package foundation presented in Figure 4.11 for the optimal core language. Furthermore, the rule selects the class Description and the attribute type as a part of this package.

4.2.3.4 Configuration Knowledge Base

The configuration knowledge base contains the configuration logic, which is used by the configuration procedure (see Figure 4.13). The configuration logic is described using the formalism for market properties and the formalism for configuration rules introduced in Sections 4.2.3.2 and 4.2.3.3. The configuration procedure applies relevant configuration rules from the knowledge base of given market properties to obtain the optimal core language. Thus, the knowledge base formalizes the knowledge needed for the configuration and, thus, supports a systematic and automated configuration procedure.

Section 4.2.2 describes that the packages as well as their language constructs of the comprehensive core language might change over time. In order to remain independent from a concrete implementation of service properties in the comprehensive core language, configuration rules in the configuration knowledge base are formulated as *language-independent*, i.e., over service properties instead of their concrete realization as packages and language constructs. In this manner, the configuration knowledge base can be reused for any service specification language supporting service properties specified in the configuration rules. A mapping between concrete packages and language constructs realizing a certain service property needs to be specified. If service properties are modeled as features, such mapping can be specified by traces connecting a feature to a single metamodel class and its attributes, to a single attribute, or to a set of metamodel classes and their attributes [158].

Service properties and their mapping Figure 4.16 presents the set of coarse-grained service properties, which are realized either as packages or as a set of language constructs and used in the rules of the configuration knowledge base. The notation of feature diagrams is used to represent these service properties. In general, feature diagrams aim to model common and distinct properties in a family of systems [29].

The feature diagram in Figure 4.16 specifies common and distinct service properties for a family of optimal core languages. This family results from the configuration of the comprehensive core language, which creates optimal core languages covering its different parts. The feature diagram is a tree of features with a root representing an optimal core language. The root contains all service properties, whose configurations represent all possible optimal core languages. All listed service properties together build the comprehensive core language. Service properties are modeled as optional features. Optional features might be added in one configuration and might be omitted in another [32].

The feature `Operation signatures` stands for a service property of operations constituting a service interface. A service hides its implementation and exposes its properties in the form of interfaces. An interface has one or several operations, which are described by their operation signatures.

The feature `Pre-/Postconditions` refers to the behavioral specification of

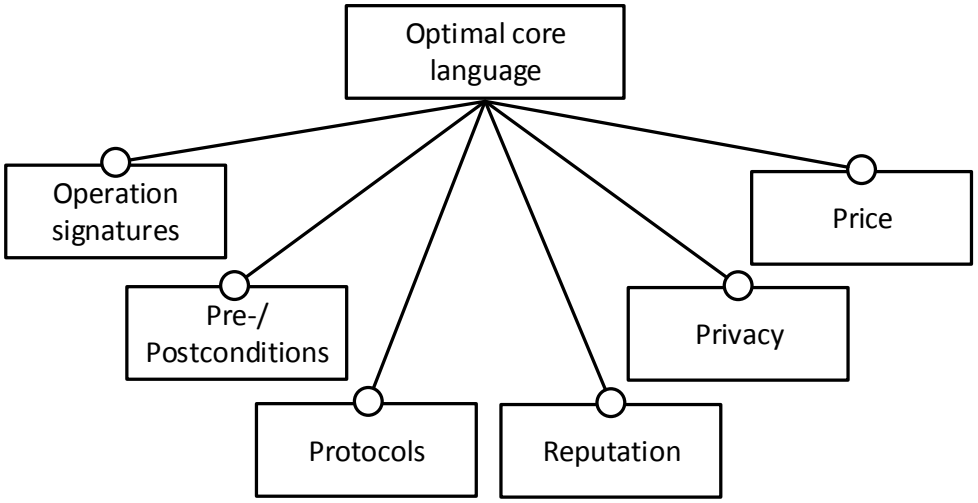


Figure 4.16: Service properties in the configuration knowledge base

service operations in the form of conditions. Pre-/Postconditions describe the semantics of a service. A precondition specifies properties necessary to hold for the execution of a service operation, while a postcondition specifies properties necessary to hold after this operation finishes its execution [97].

The feature `Protocols` supports another behavioral specification of services. Protocols are described for a service, whose interface has several operations. In this case, a protocol states the call order of these operations.

The features above describe the functionality of services, while the following features refer to different non-functional properties describing the service quality.

The feature `Reputation` refers to the reputation of service providers and requesters in an OTF market [59]. For example, this information indicates, how good a service satisfies the promised service level agreements. Another example is the reputation of a service requester, which states how reliable a requester is in paying the costs for the provided service.

The feature `Privacy` specifies, how a certain service handles the sensitive data of service requesters given as an input [116].

The feature `Price` describes the payment model for a service, i.e., how a provider charges the usage of a service. For example, the payment can take place according to such models as licenses, subscription-based, or pay-per-use [47].

Figure 4.17 shows fine-grained service properties used in the configuration rules. These properties are usually modeled either by one single language construct or by a small set of several language constructs. A mapping from concrete language constructs to the fine-grained features is necessary.

The coarse-grained feature `Operation signatures` from Figure 4.16 is refined by a set of optional fine-grained features. The feature `Operation names` stands for the name of an operation in its signature, and the feature `Operation parameters` refers to its parameters. The feature `Operation parameters` is

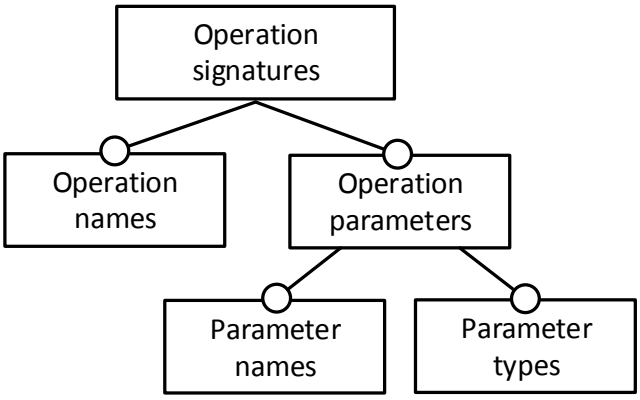


Figure 4.17: Detailed service properties of operation signatures

again refined by two optional features. The feature `Parameter names` stands for the names of operation parameters and `Parameter types` for their types.

For the configuration, a mapping from the packages and language constructs of the comprehensive core language to the coarse- and fine-grained features introduced above is needed.

Figure 4.18 shows the excerpt of a metamodel realizing the feature `Operation signatures` in the comprehensive core language. Language constructs from this metamodel are mapped onto the features in Figure 4.17.

The metamodel contains the class `Repository` representing a storage of service interfaces modeled by the class `OperationInterface`. An operation interface has a set of operations represented by the class `OperationSignature`. Both operation interfaces and operation signatures have a name (see the inheritance from the abstract class `NamedElement` having the attribute `entityName` of type `EString`). An operation signature has a set of parameters modeled by the containment to the class `Parameter`. A parameter has a name as modeled by the attribute `parameterName` of type `EString`. Furthermore, a parameter refers to its data type as modeled by the association to the abstract class `DataType`.

Table 4.1 shows the mappings from the feature diagram in Figure 4.17 to the language constructs from Figure 4.18.

Feature	Class / Attribute
`Operation signatures`	`OperationSignature`
`Operation names`	`entityName`
`Operation parameters`	`Parameter`
`Parameter names`	`parameterName`
`Parameter types`	`DataType`

Table 4.1: Mappings from features to language constructs

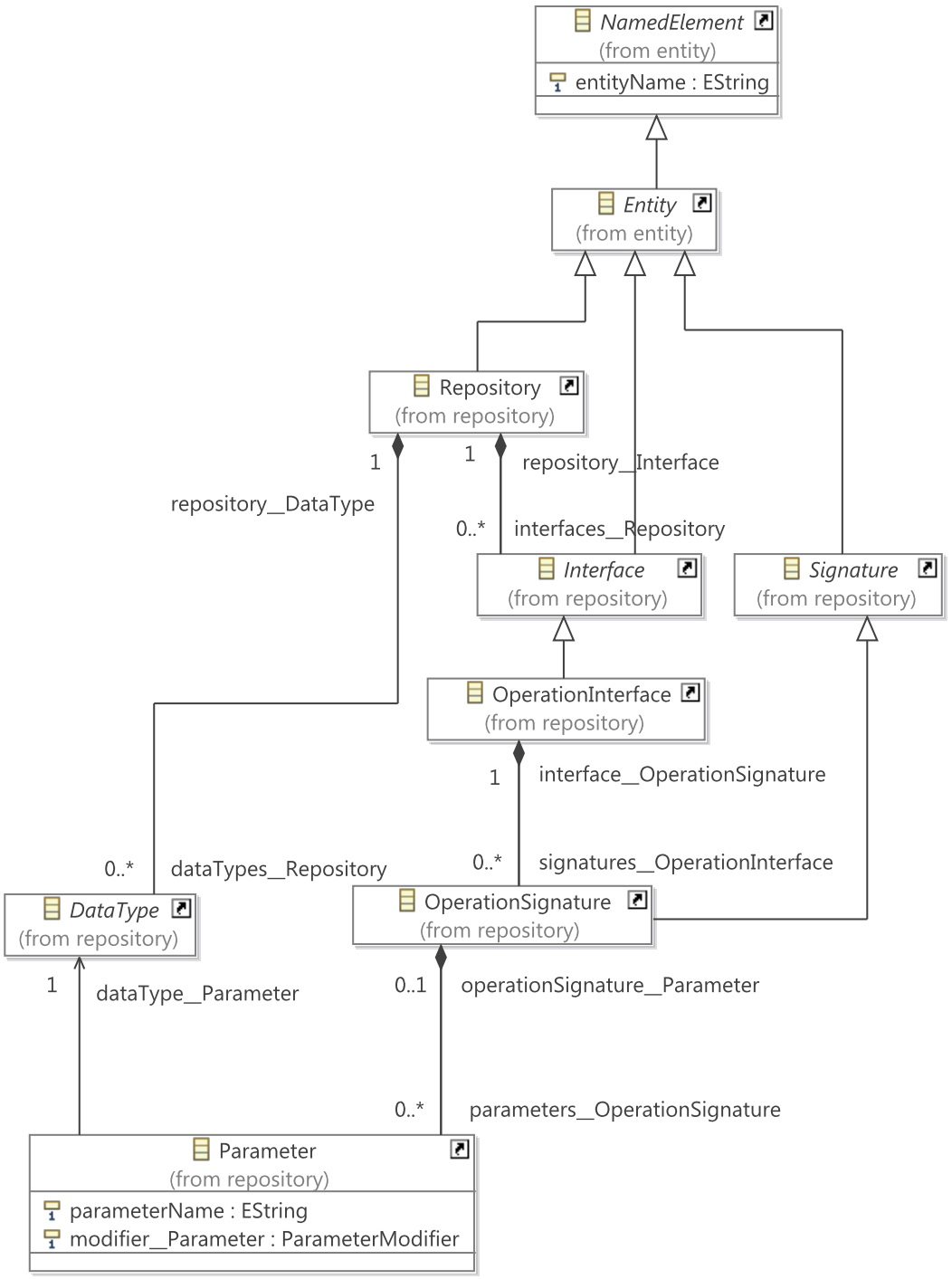


Figure 4.18: Excerpt of the metamodel of the comprehensive core language

Features are mapped either to classes or to attributes of the given metamodel. The feature `Operation signatures` is mapped to the metaclass `OperationSignature` representing service operations. Its constituent fine-grained feature `Operation names` is mapped to the attribute `entityName` inherited by the class `OperationSignature` from the class `NamedElement`. The feature `Operation parameters` is mapped to the metaclass `Parameter`. The feature `Parameter names` maps to the metamodel attribute `parameterName`. The feature `Parameter types` maps to the abstract class `DataType` and, as a consequence, to all concrete metaclasses inheriting from it.

Configuration knowledge base In the following, the configuration rules of the configuration knowledge base are presented. These configuration rules are also described in our papers [7, 8].

Listing 4.2 introduces a shortened notation for configuration rules on the example from Listing 4.1. This notation is used to describe configuration rules in the following.

Listing 4.2: Shortened notation for configuration rules

```
1 cr: ConfigurationRule SELECTION_OF_PACKAGES
2 cr.marketProperty = Standardization TRUE
3 cr.operator = SelectOperator {foundation: Description, type}
```

Listing 4.3 illustrates the configuration rules for the market property `Standardization`.

The rule `cr1` states that the feature `Operation signatures` has to be selected for the optimal core language in an OTF market with standardized terminology and processes. Operation signatures suffice for effective matching in such a market for several reasons. First, due to a standardized and established terminology, service providers and requesters use the same names and data types for the same concepts in signatures. As a result, the matching based on operation signatures produces reliable matching results with no need for behavioral specifications of operations with pre-/postconditions. Second, due to standardized processes, different services realize the same functionality according to the same process. Therefore, the specification as well as the matching of interface protocols can be omitted without much loss in the matching effectiveness.

The configuration rule `cr2` is defined for an OTF market with standardized terminology but no standardized processes. In such a market, the service matching based on operation signatures delivers reliable matching results as explained above. However, due to non-standardized processes, different services realize the same functionality using different processes. Therefore, the specification of interface protocols and their matching are needed.

The configuration rule `cr3` applies for markets having no standardized terminology but standardized processes. In this case, the features of `Operation signatures` and `Pre-/Postconditions` are required, in order to match the

structure and behavior of service operations. Due to the non-standardized terminology, operations cannot be reliably matched based on their signatures only. In addition, semantic matching of operation signatures using ontologies have to be performed. The behavior of operations have to be specified and matched using pre-/postconditions. After the operations are matched, the order of their execution is determined by the standardized processes of the OTF market.

Listing 4.3: Configuration rules for `Standardization`

```
cr1: ConfigurationRule SELECTION_OF_PACKAGES
cr1.marketProperty = Standardization TRUE
cr1.operator = SelectOperator {Operation signatures}

cr2: ConfigurationRule SELECTION_OF_PACKAGES
cr2.marketProperty = Standardization TERMINOLOGY_ONLY
cr2.operator = SelectOperator {Operation signatures, Protocols}

cr3: ConfigurationRule SELECTION_OF_PACKAGES
cr3.marketProperty = Standardization PROCESS_ONLY
cr3.operator = SelectOperator {Operation signatures,
Pre-/Postconditions}

cr4: ConfigurationRule SELECTION_OF_PACKAGES
cr4.marketProperty = Standardization FALSE
cr4.operator = SelectOperator {Operation signatures,
Pre-/Postconditions, Protocols}

cr5: ConfigurationRule SELECTION_OF_LANGUAGE_CONSTRUCTS
cr5.marketProperty = Standardization PROCESS_ONLY
cr5.operator = NoSelectOperator {Operation signatures:
Operation names, Parameter names}

cr6: ConfigurationRule SELECTION_OF_LANGUAGE_CONSTRUCTS
cr6.marketProperty = Standardization FALSE
cr6.operator = NoSelectOperator {Operation signatures:
Operation names, Parameter names}
```

The rule `cr4` states, which features to select in markets with no standardization of either terminology or processes. In such markets, `Operation signatures`, `Pre-/Postconditions`, and `Protocols` are added to the optimal core language. Due to the lack of standardization, the structure of service operations, their behavior and the order of their execution have to be specified and matched.

The configuration rules `cr5` and `cr6` are defined for markets lacking the standardization of terminology. The rules state that the fine-grained features `Operation names` and `Parameter names` contained in the coarse-grained feature `Operation signatures` have to be omitted. Due to the lack of standardized terminology, service providers and requesters use different operation names and parameter names for the same concepts. The matching would rely on parameter types only. Thus, names does not contribute much to the matching effectiveness and, thus, can be omitted in the optimal core language.

Listing 4.4 defines the configuration logic for the market property `MarketSize`.

Listing 4.4: Configuration rules for `MarketSize`

```
cr7: ConfigurationRule SELECTION_OF_PACKAGES
cr7.marketProperty = MarketSize LARGE
cr7.operator = NoSelectOperator {Protocols, Reputation}

cr8: ConfigurationRule SELECTION_OF_PACKAGES
cr8.marketProperty = MarketSize MEDIUM
cr8.operator = NoSelectOperator {Reputation}

cr9: ConfigurationRule SELECTION_OF_PACKAGES
cr9.marketProperty = MarketSize SMALL
cr9.operator = SelectOperator {Reputation}
```

The configuration rule `cr7` states that the features `Protocols` and `Reputation` have to be omitted in a large OTF market. In such markets, a lot of provided services exist. In order to match all these services with a given requirements specification, the service matching has to be performed very efficiently. For that, the time-consuming matching has to be avoided. Since the matching of service protocols involves model checking that is known as time-consuming, the protocols are omitted from the optimal core language.

The reputation has to be omitted because the reputation values can be falsified easily in large markets. Similar to large markets, `Reputation` has to be omitted for markets with a medium size as stated in the rule `cr8`. In contrary to the previous two rules, in small OTF markets, the reputation has to be specified and considered for matching (see `cr9`). In small markets, service providers cannot falsify their reputation easily because they are well-known. Therefore, the reputation values are reliable.

Listing 4.5 presents the rules for the next market property `SensitiveData`.

Listing 4.5: Configuration rules for `SensitiveData`

```
cr10: ConfigurationRule SELECTION_OF_PACKAGES
cr10.marketProperty = SensitiveData TRUE
cr10.operator = SelectOperator {Privacy}

cr11: ConfigurationRule SELECTION_OF_PACKAGES
cr11.marketProperty = SensitiveData FALSE
cr11.operator = NoSelectOperator {Privacy}
```

The rule `cr10` states that, if sensitive data is involved, then the privacy of services have to be selected for the optimal core language. Privacy is required due to the laws obliging the service providers to handle sensitive data accordingly and due to the wish of service requesters to keep their personal data save. Alternatively, if no sensitive data is involved, then no privacy has to be added to the core language (see the rule `cr11`).

Listing 4.6 presents the rules for the market property `ServiceComplexity`.

Listing 4.6: Configuration rules for `ServiceComplexity`

```
cr12: ConfigurationRule SELECTION_OF_PACKAGES
cr12.marketProperty = ServiceComplexity HIGH
cr12.operator = SelectOperator {Operation signatures,
Pre-/Postconditions, Protocols}

cr13: ConfigurationRule SELECTION_OF_PACKAGES
cr13.marketProperty = ServiceComplexity MEDIUM
cr13.operator = SelectOperator {Operation signatures,
Pre-/Postconditions}

cr14: ConfigurationRule SELECTION_OF_PACKAGES
cr14.marketProperty = ServiceComplexity LOW
cr14.operator = NoSelectOperator {Pre-/Postconditions, Protocols}
```

The rule `cr12` defines that, if services are complex, then their structure as well as their behavior have to be specified in the optimal core language. Complex services usually provide their functionality in several service operations, which must be called in a certain order. In this case, the operations often realize complex functionality, too. Thus, the structure as well as the behavior of service operations have to be specified for complex services. Furthermore, since the operations have a certain call order, the protocols have to be described, too. As a result, the features `Operation signatures`, `Pre-/Postconditions`, and `Protocols` are added to the optimal core language.

The configuration rule `cr13` states that, for services with medium complexity, the features of `Operation signatures` and `Pre-/Postconditions` are sufficient in the optimal core language. The reason for that is the assumption that services with medium complexity provide only few service operations, which, however, realize some complex functionality. Therefore, the description of protocols and their matching can be omitted for such services.

For services having low complexity, the rule `cr14` proposes to omit `Pre-/Postconditions` as well as `Protocols`. This is due to the fact that simple services mostly provide simple functionality realized by one operation only. Such operations can be reliably matched based on their operation signatures.

Listing 4.7 presents configuration rules for the property `Profit`.

Listing 4.7: Configuration rules for `Profit`

```
cr15: ConfigurationRule SELECTION_OF_PACKAGES
cr15.marketProperty = Profit LICENSES
cr15.operator = SelectOperator {Price}

cr16: ConfigurationRule SELECTION_OF_PACKAGES
cr16.marketProperty = Profit ADS
cr16.operator = NoSelectOperator {Price}
```

The rule `cr15` states that, if service providers make profit by selling licenses for their services, then the price of services has to be described in the optimal core language. The configuration rule `cr16` defines that, for services, whose

providers make profit by using advertising, the feature `Price` is unnecessary in the optimal core language. Providers usually offer this kind of services for free with the goal to profit from the clicks on advertising banners.

Listing 4.8 shows the configuration rules for the property `Trade-off focus`.

Listing 4.8: Configuration rules for `TradeoffFocus`

```
1 cr17: ConfigurationRule TIE_BREAKER
2 cr17.marketProperty = TradeoffFocus EFFICIENCY
3 prioritize NoSelectOperator
4
5 cr18: ConfigurationRule TIE_BREAKER
6 cr18.marketProperty = TradeoffFocus EFFECTIVENESS
7 ignore NoSelectOperator
```

The rules `cr17` and `cr18` have the type `TIE_BREAKER`. Their goal is to resolve conflicts in the application of the select and no-select operators. Such conflicts occur, if two applicable rules exist, from which one states to select a feature and another states to omit the same feature. The `TIE_BREAKER` rules state whether the focus of the trade-off is set to efficiency or effectiveness in the case of a conflict. If the trade-off focus is set to efficiency, then the rules omitting features get a higher priority than the rules selecting features. Otherwise, if the focus is set to effectiveness, the rules omitting features are ignored. Because of the tie breaker rules, the LOpt can deterministically configure an optimal core language for the same set of market properties.

4.2.3.5 Configuration Procedure

The configuration procedure applies rules from the configuration knowledge base depending on values of the input market properties (see Figure 4.13).

The configuration procedure applies `SELECTION_OF_PACKAGES` rules with the `SelectOperator` by adding the language packages corresponding to the coarse-grained features selected by the rule. As a result of the rule application, the selected language packages as well as *all* their language constructs are added to the optimal core language. The configuration procedure applies `SELECTION_OF_LANGUAGE_CONSTRUCTS` rules with the `SelectOperator` by adding packages with the *selected* language constructs only. The added language constructs correspond to the fine-grained features stated in the rule.

When applying `SELECTION_OF_PACKAGES` rules with the `NoSelectOperator`, the configuration procedure omits the coarse-grained features, which have been added by the rules with the `SelectOperator`. The added language packages and all their language constructs are removed from the optimal core language. The configuration procedure applies `SELECTION_OF_LANGUAGE_CONSTRUCTS` rules with the `NoSelectOperator` by omitting the specified fine-grained features already selected for the optimal core language. Thus, the corresponding language constructs are removed from their packages.

The configuration procedure builds the optimal core language as a view type of the comprehensive core language (see Section 4.2.2.5) . Based on a configuration of coarse- and fine-grained features selected by applied rules, placeholders for packages and language constructs are added to the view type based on the mapping of features to language constructs.

To apply rules, the configuration procedure has to address the following issues: *priorities of rule application* and *strategies for conflict solving*.

The configuration procedure has to set priorities of rule application to define the order, in which it executes the rules from the configuration knowledge base. Rule priorities can be based on rule types or market properties. Regarding the rule types, the configuration procedure can apply the rules with the `SelectOperator` first, followed by the rules with the `NoSelectOperator`. Regarding the market properties, the configuration procedure can apply rules regarding a certain order of market properties, e.g., the rules for `Standardization` followed by the rules for `MarketSize`.

The configuration knowledge base of LOpt builds the optimal core language from an empty set of packages. The reason for this decision is the idea to optimize the trade-off between the matching efficiency and effectiveness starting with the maximal efficiency because of the assumption that large OTF markets will prevail in the OTF Computing. Since the configuration procedure starts with an empty set, it applies rules with the `SelectOperator` first, in order to initially fill the core language with packages and language constructs. Afterwards, the rules with the `NoSelectOperator` are executed to further optimize the trade-off by decreasing the obtained effectiveness.

Starting with the maximal effectiveness is also possible. In this case, the comprehensive core language represents the initial core language, which has to be reduced, in order to optimize the trade-off. For that, some of the configuration rules should be changed from using the `SelectOperator` to using the `NoSelectOperator`. The rules with the `NoSelectOperator` have a higher priority than the rules with the `SelectOperator` and, thus, are executed first. Afterwards, rules with the `SelectOperator` are applied to increase the effectiveness of the trade-off if necessary.

Regarding the prioritization with respect to market properties, the configuration procedure of LOpt handles them as equally important. Alternatively, one way to prioritize them could be the user input. For example, the users of LOpt could give a higher priority to those properties, which they consider as especially important in their OTF market. They could also prioritize those properties, which values they are especially confident in. The configuration procedure would apply rules according to the given priorities of market properties. In this strategy, the user preferences are stronger considered in comparison to the strategy, in which the configuration starts with the minimal efficiency.

The configuration knowledge base contains rules, which stay in conflict with each other. For example, the configuration rule `cr2` selects the service property of `Protocols` while the rule `cr6` omits it. These two rules are applicable for

the same OTF market, if the terminology in the market is standardized and the market is large. This kind of conflicts reflects the trade-off between the matching efficiency and effectiveness because, in the considered OTF market, `Protocols` are needed to increase the effectiveness but, simultaneously, they have to be omitted to maintain the efficiency.

In order to solve such conflicts, special rules of type `TIE_BREAKER` are introduced. These rules state that the users of LOpt have to specify their own focus of the trade-off: either on efficiency or on effectiveness. If the trade-off focus is set to efficiency, then the configuration procedure applies the rules with `NoSelectOperator` omitting the already selected packages and language constructs in the optimal core language. Otherwise, the configuration procedure ignores all rules with the `NoSelectOperator`. In this way, the configuration procedure considers user preferences regarding the trade-off.

4.2.4 Configure and Use the Optimal Core Language

This section describes the usage of an optimal core language and illustrates the application of the configuration procedure to a concrete OTF market [7]. For different OTF markets, different core languages optimally support the execution of automated market operations in these markets. Brokers, as intermediaries between service requesters and service providers, use these core languages to describe service compositions and their constituent services.

In order to search for services, service requesters have to transform their requirements specifications into the optimal core language. Based on the transformed requirements specification, a broker uses a special composition software to generate corresponding specifications of a service composition and its constituent services. Service providers have to transform specifications of their provided services into the optimal core language, too. Based on these specifications, the operations `Match a service`, `Discover a service`, `Compose a service`, and `Analyze a service` (see Figure 1.2) are performed.

Brokers become primary users of the LOpt approach. Having an optimal core language created with LOpt, a broker works with specifications, which optimally support the operation `Match a service` with respect to the matching efficiency and effectiveness. As a result, the operation `Discover a service`, which uses the operation `Match a service` to find suitable services, and the operation `Compose a service`, which uses the operation `Discover a service` for discovering the constituent services for a service composition, are facilitated regarding the service matching. Thus, a broker gains an advantage over other brokers, who do not use an optimal core language in an OTF market, because the broker creates composed services more effectively and efficiently.

A broker needs to select the relevant market properties to formalize the OTF market and their strategy. Then, the configuration procedure runs to configure the optimal core language. Finally, this optimal core language is delivered to the broker for use in the OTF market.

For example, the broker sets the market property `MarketSize` based on the amount of provided services in the market. If the search is performed on all services of the market, then the broker cannot influence the value of this property as the amount of services in the market depends on such market mechanisms as supply and demand. In comparison, the broker can assign the market property `SensitiveData` based on her strategy in the market. If the broker knows that specifications of sensitive data are unreliable in the given OTF market, then the broker can set this property to `FALSE`. In this case, specifications of the service privacy will not be covered by the optimal core language and, thus, will not be considered during the service matching. For the matching services output as a result, the broker can check the usage of sensitive data manually.

4.3 Evaluation

This section presents the evaluation of the LOpt approach. Section 4.3.1 presents the tool support realizing LOpt. Section 4.3.2 continues with the application of LOpt to different OTF markets, whose optimal core languages configured with LOpt are evaluated. Section 4.3.3 compares the results with the requirements stated in Section 3.1.1.

4.3.1 Tool Support

The tool support realizing the LOpt approach is a part of the tool suite called SeSAME. SeSAME stands for **Se**rvice **S**pecification, **A**nalysis and **M**atching **E**nvironment [10]. This environment supports specification of services and service compositions, service matching, functional analysis of service compositions based on protocols and pre-/postconditions, and quality analysis of service compositions with respect to performance and scalability.

Figure 4.19 gives an overview over the tool support. For simplification, components for functional and non-functional analysis are omitted.

`SeSAME` and its components are realized as Eclipse plugins using the Eclipse Modeling Framework (EMF) that follows the principles of the model-driven software development [140]. The component `SSE` (**S**ervice **S**pecification **E**nvironment) [10] realizes the concept of the comprehensive core language introduced in Section 4.2.2. `SSE` uses and extends the Palladio Component Model (PCM) [119] with new service properties and language operations, e.g., view building. The component `MatchBox` [117, 22] realizes the operation of service matching defined for specifications created in SSE. The component `LM configurator` realizes the language configuration explained in Section 4.2.3. `LM configurator` also performs the configuration of the service matching defined in `MatchBox`, in order to configure the optimal service matching for the optimal core language.

The realization of the comprehensive core language in SSE contains language packages needed to cover all service features presented in Figure 4.16. Each

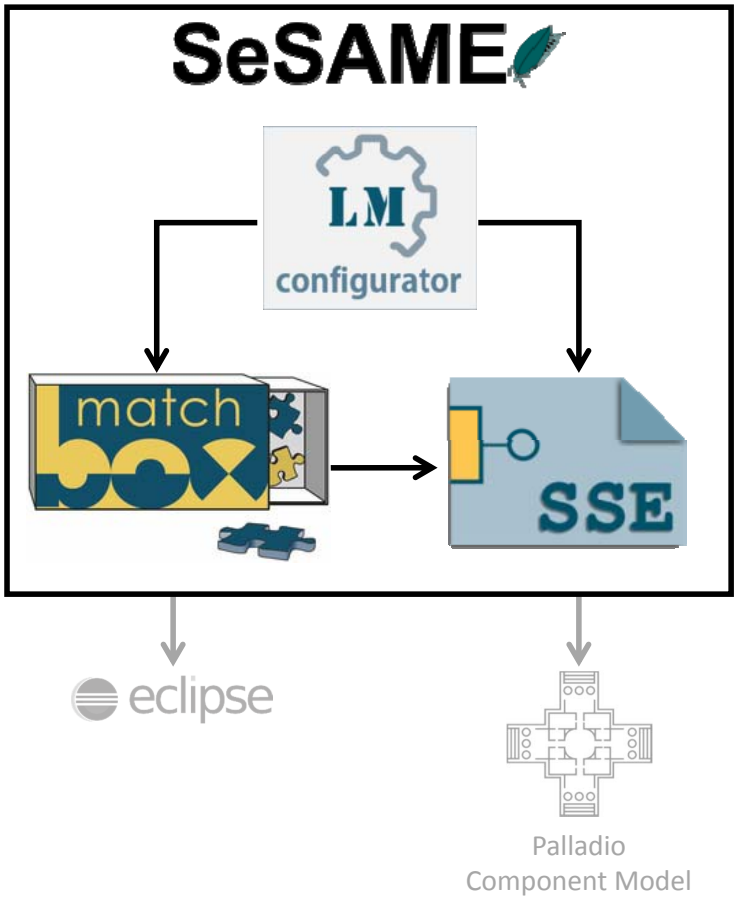


Figure 4.19: Overview of the tool support for LOpt

language package is realized as a metamodel. Those language packages, which extend the metamodels of PCM, are realized as decorator metamodels. Editors for creating specifications as instances of these metamodels are generated using the corresponding EMF mechanisms. Tree editors, textual editors, and graphical editors were generated for service specification in SSE.

Figure 4.20 presents the graphical editor for the specification of service interfaces and their operations in SSE. Each interfaces is shown as a rectangle having a compartment for the interface name (e.g., `RequestForReservationService`) and a compartment with the operations provided by this interface (e.g., `search` and `reserve`). For each operation, the specification contains its name, the list of its input and output parameters, and its exceptions.

Figure 4.21 shows the specification of pre-/postconditions created in a textual editor. The specification describes pre-/postconditions for the operation `simpleSearchRoom` of `simpleReservationService` introduced in Figure 4.20. This editor enables the checking of the syntax and semantics of the specified pre-/postconditions. The syntax check is performed against the language de-

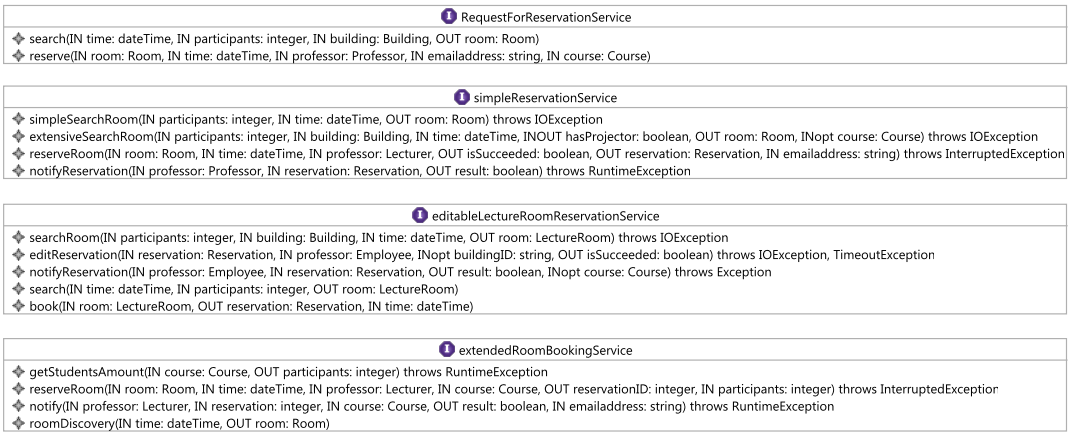


Figure 4.20: Specification of service interfaces and operations in SSE

finition. The semantics check uses the underlying ontology modeling an OTF market. In particular, it checks whether all the specified predicates really exist in the ontology. Another check ensures that the parameters and free variables used in the predicates are compatible with their definitions in the ontology.

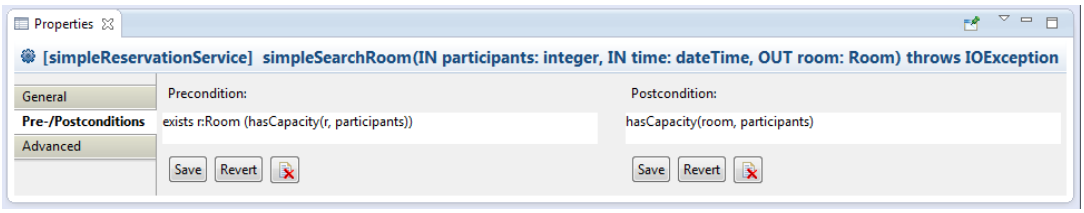


Figure 4.21: Specification of pre-/postconditions in SSE

The component `LM configurator` realizes the configuration of the comprehensive core language. It allows the specification of market properties and the configuration logic in the form of configuration rules. Figure 4.22 shows a specification of market properties created in `LM configurator`. This specification is created in a tree editor that allows to specify all properties and their values introduced in Figure 4.14. For example, the market property `Standardization` is set to `FALSE` and the market property `MarketSize` is set to `SMALL`.

Figure 4.23 presents a specification of several configuration rules from Section 4.2.3.4. In this specification, the rules of both types are shown. As a part of the rule, a market property with a concrete value exist, e.g., `Standardization TRUE`. Then, the application of the `Select Operator` to the service feature `OperationSignatures` is specified.

MatchBox realizes the service matching for specifications written in the comprehensive core language. MatchBox contains different matching approaches, which are defined for service specifications being instances of metamodels realizing the corresponding language packages. MatchBox enables to use these

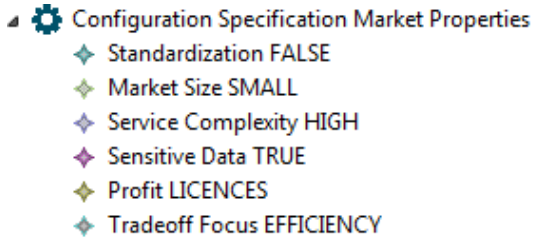


Figure 4.22: Specification of market properties in `LM configurator`

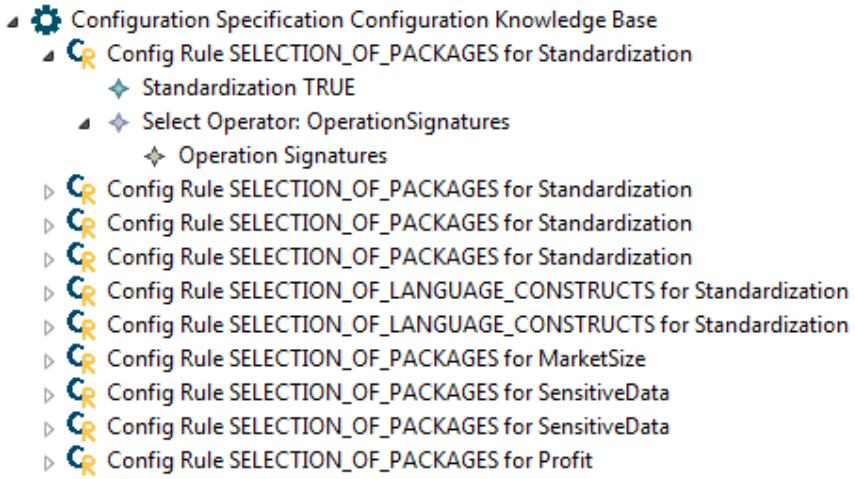


Figure 4.23: Specification of configuration rules in `LM configurator`

matching approaches as steps within executable matching processes, which realize a stepwise matching of comprehensive service specifications. The steps have a certain order in a matching process. Based on the matching results calculated for the steps, the aggregated matching result is computed according to a certain aggregation strategy, e.g., a weighted aggregation strategy, which considers each step result according to its given weight. Additionally, the parameters of the matching approaches can be configured. For example, different step results can be weighted differently in the aggregated result. Since this PhD thesis focuses on the specification of services, MatchBox is not shown in detail here.

4.3.2 Evaluation on Case Studies

The evaluation of LOpt is performed for the OTF markets of tourism, university management, and water net optimization. The goal of the evaluation is to show that the LOpt approach outputs an optimal solution with respect to the trade-off between effectiveness and efficiency illustrated in Figure 4.3.

This section starts with explaining the evaluation procedure in Section 4.3.2.1. Section 4.3.2.2 describes the setting of market properties for the considered service markets. Section 4.3.2.3 shows the optimal core languages, which were

configured by LOpt based on the specified market properties. Section 4.3.2.4 illustrates the test collections used to measure the matching efficiency and effectiveness in the considered OTF markets. Section 4.3.2.5 presents the evaluation of the optimal core languages based on the matching efficiency and effectiveness measured for the introduced test collections.

4.3.2.1 Evaluation Procedure

The evaluation procedure of the LOpt approach contains four steps:

Step 1 Domain experts set the values of market properties for the given service markets. Thereby, the markets are described as well as a certain broker strategy is considered. For the evaluation, domain experts from the CRC 901 created the specifications of market properties for the considered markets relying on their expertise. The value assigned to the market properties are presented in Section 4.3.2.2.

Step 2 The LOpt approach is executed to obtain optimal core languages for the specified market properties. The resulting core languages are shown in Section 4.3.2.3.

Step 3 Domain experts build test collections for the considered service market, which are used to show the optimality of the obtained core languages. These test collections contain representative services for each market. Domain experts may also use existing service and requirements specifications. In this case, they have to be transformed into the optimal core language. The build test collections are introduced in Section 4.3.2.4.

Step 4 Finally, the optimality of the obtained core languages is shown based on the metrics for the matching efficiency and effectiveness measured for the presented test collections. These measurements are presented in Section 4.3.2.5. As a result, it is shown that the LOpt approach produces optimal core languages by design.

4.3.2.2 Setting the Values of Market Properties

Table 4.2 presents the values assigned for the market properties describing the OTF markets from the case studies. The case studies are chosen in a way that they cover all range values for each market property at least once. As a result, each configuration rule from the configuration knowledge base applies.

OTF market for tourism The OTF market for tourism trade services for trip reservation including, in particular, the reservation of hotels, flights, and restaurants. It is an established large market, where service providers offer many services (`Market size LARGE`). The terminology in this market is standardized as properties of hotels or flights are common worldwide. Furthermore,

Market property value	Tourism	University management	Water net optimization
Market size	LARGE	SMALL	MEDIUM
Standardization	TRUE	FALSE	FALSE
Sensitive data	TRUE	TRUE	FALSE
Service complexity	LOW	MEDIUM	HIGH
Profit	ADS	LICENSES	LICENSES
Trade-off focus	EFFICIENCY	EFFICIENCY	EFFECTIVENESS

Table 4.2: Assignment of market properties in the case studies

the processes of booking a hotel or a flight are mostly identical. Thus, the market is considered to be have both standardized terminology and processes (`Standardization TRUE`).

Many services in this market work on sensitive data, e.g., customer name, address, or credit card needed for the reservation of a hotel or flight. Thus, it is feasible for a broker to consider this property (`Sensitive data TRUE`). The services provide mostly a simple functionality in the OTF market (`Service complexity LOW`). Such services usually provide one interface having one operation realizing the service functionality. The service matching for such services is more efficient in comparison to complex services, for which the expensive protocol matching has to be performed. The broker knows that most of the services are offered for free in the tourism market (`Profit ADS`).

The broker sets the trade-off focus to efficiency as the broker wants to serve the requesters as fast as possible (`Trade-off focus EFFICIENCY`). The configuration of LOpt will provide the broker with the best effectiveness regarding the given setting of market properties.

OTF market for university management The OTF market for university management trades services providing functionality for different tasks at a university, e.g., exam reservation, room booking, or progress reporting. The organization of universities worldwide differs significantly. The first reason is the fact that most university management systems are proprietary software used internally. The second reason is the lack of standardization in the domain that causes the reluctance in offering services because of their rare compatibility to other universities. Therefore, this market is considered to have neither standardized terminology nor standardized processes (`Standardization FALSE`). The university management market is a relatively young and, therefore, is a small market having only few provided services (`Market size SMALL`).

Services in the university management market work on sensitive data, e.g., the information about students, their grades, and exams. The broker wants to consider this property for the service matching (`Sensitive data TRUE`). Most

services provided in this market have a medium complexity as they provide their functionality by single complex operations (`Service complexity MEDIUM`).

Most service providers in the university management market make profit by selling licenses for their services. Thus, the broker wants to consider this service property for the matching (`Profit LICENSES`). Similar to the tourism market, the broker sets the trade-off focus on efficiency, in order to serve the coming requesters as fast as possible (`Trade-off focus EFFICIENCY`).

OTF market for water net optimization The OTF market for water net optimization trades services for optimization of water supply systems [38]. Examples of such services are solvers like Gurobi or CPLEX, or services reducing parallel pipes and chains in a water net. The market has a medium size (`Market size MEDIUM`), and is not standardized as different service providers use their own terminology and processes for the services (`Standardization FALSE`). Services for water net optimization do not involve data about individuals as these services operate on accumulated data and profiles (`Sensitive data FALSE`).

The complexity of the services in this market is high because services implement complex algorithms operating on graphs representing water nets (`Service complexity HIGH`). Most service providers make profit in the market by selling licenses for their services, which provide a broad functionality necessary for the water net optimization. Therefore, the broker want to discover and compose these services (`Profit LICENSES`). The broker sets `Trade-off focus` to `EFFECTIVENESS` because getting a suitable optimization service is more important for service requesters than getting the service fast.

4.3.2.3 Obtaining the Optimal Core Languages with LOpt

In the second step of the evaluation procedure, LOpt runs using the market properties specified in Section 4.3.2.2 as an input. The optimal core languages are obtained by configuring the feature models presented in Figures 4.16–4.17.

OTF market for tourism Figure 4.24 illustrates the optimal core language. It contains two coarse-grained features `Operation signatures` and `Privacy`, and all fine-grained features of the feature `Operation signatures`.

The configuration procedure of LOpt produced this configuration by applying the rules `cr1`, `cr7`, `cr10`, `cr14`, `cr16`, and `cr17` from the configuration knowledge base described in Section 4.2.3.4. Section 4.2.3.5 explains that the configuration procedure applies the rules with the select-operator first. Thus, the rules `cr1` and `cr10` select the coarse-grained features `Operation signatures` and `Privacy`. As a part of this selection, all fined-grained features of the feature `Operation signatures` are selected as well. Since the trade-off focus is set to efficiency, then the rule `cr17` prioritizes the no-select operator over the select operator. As a result, the rules `cr7`, `cr14`, and `cr16` omit the features

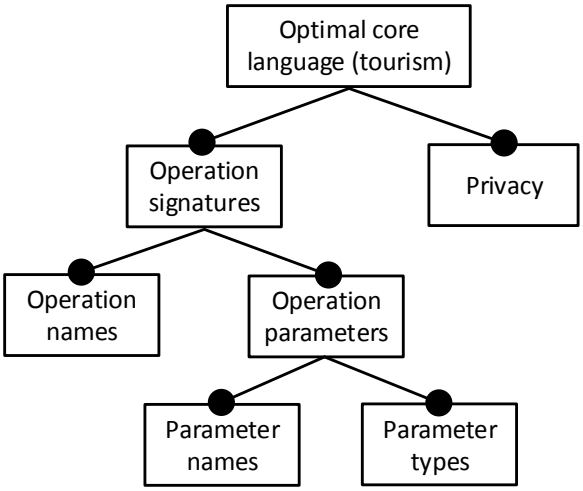


Figure 4.24: Optimal core language for the OTF market for tourism

`Pre-/Postconditions`, `Protocols`, `Reputation`, and `Price`. Since these features were not selected during the application, the built configuration remains.

Figure 4.25 presents the view type corresponding to this configuration. The view type contains placeholders for all packages and language constructs from the comprehensive core language realizing the features from Figure 4.24. The package placeholders `repository` and `privacy` realize the coarse-grained features `Operation signatures` and `Privacy` correspondingly. These placeholders reference the concrete packages `repository` and `privacy` of the comprehensive core language. Both package placeholders contain all metamodel classes, their attributes and their references, which belong to the original packages. In the following, the package placeholder `repository` is explained in detail.

For the realization of the fine-grained features, the configuration procedure used the mapping presented in Table 4.1. According to this mapping, the classifier placeholders for classes `OperationSignature`, `Parameter` and `DataType` realize the features `Operation signatures`, `Operation parameters`, and `Parameter types`. The attribute placeholders for attributes `entityName` in the placeholder `OperationSignature` and `parameterName` in `Parameter` realize the fine-grained features `Operation names` and `Parameter names`. The classifier placeholders for metamodel classes `Repository` and `OperationInterface` serve as a container for service operations and data types used in a specification. The reference placeholders for the association `dataType_parameter` from `Parameter` to `DataType` and for the containment `parameters_OperationSignature` from `OperationSignature` to `Parameter` exist in the view type as well.

OTF market for university management Figure 4.26 illustrates the configuration of the comprehensive core language representing the optimal core language for the OTF market for university management. The optimal core

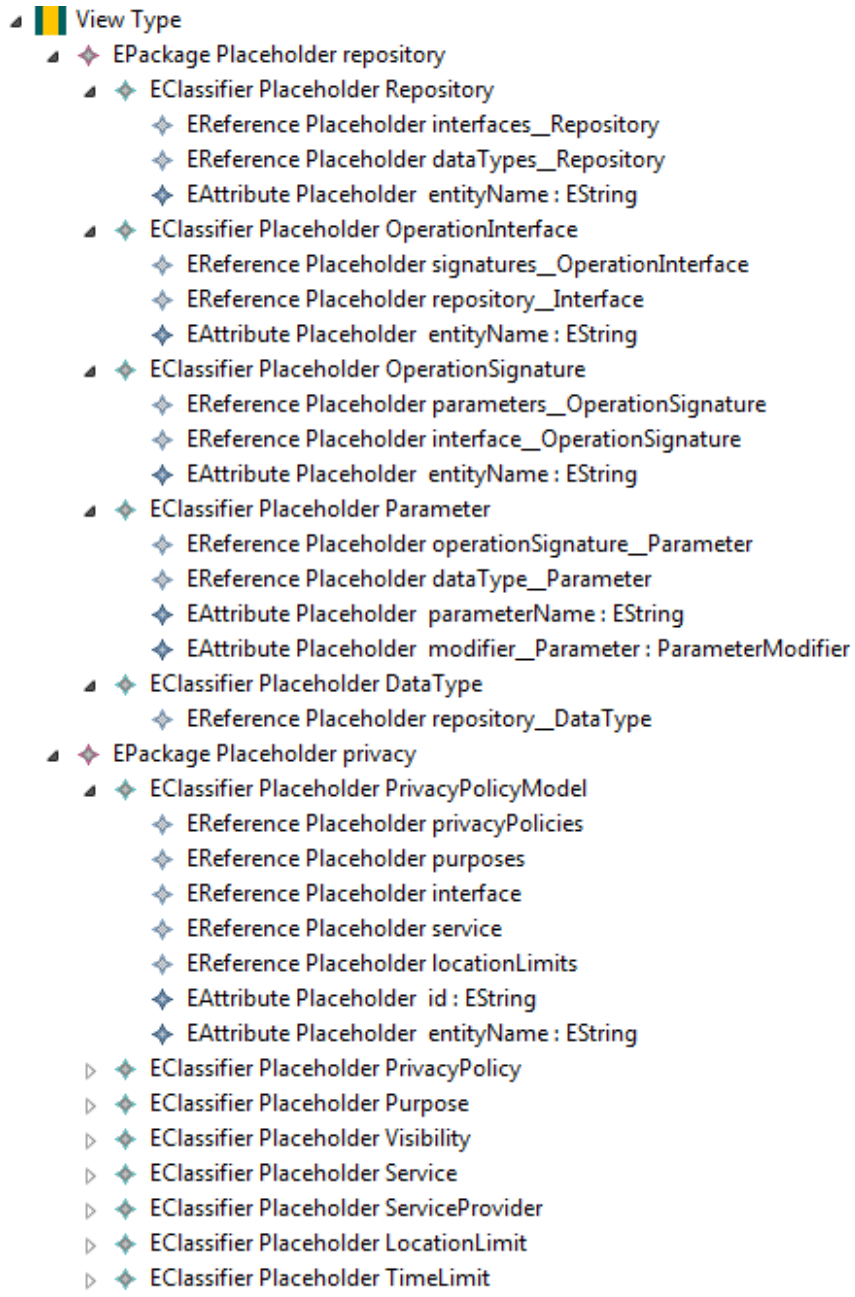


Figure 4.25: Optimal core language for tourism as a view type

language consists of six coarse-grained features. The feature of `Operation signatures` is reduced by the fine-granular features `Operation names` and `Parameter names`.

The configuration procedure of LOpt obtains this configuration by applying the configuration rules `cr4`, `cr6`, `cr9`, `cr10`, `cr13`, `cr15`, and `cr17` from the configuration knowledge base described in Section 4.2.3.4. The rules with

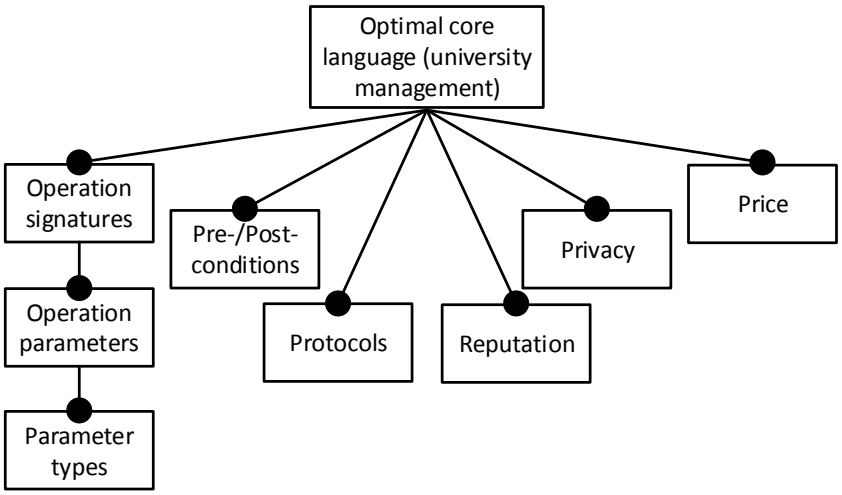


Figure 4.26: Optimal core language for the OTF market for university management

the select-operator `cr4`, `cr9`, `cr10`, `cr13`, and `cr15` apply first. These rules select the coarse-grained features `Operation signatures`, `Pre/Postconditions`, `Protocols`, `Reputation`, `Privacy`, and `Price`. Since the trade-off focus is put on efficiency, the rule `cr17` prioritizes the no-select operator over the select operator. This leads to the application of the rule `cr6` that omits the fine-grained features `Operation names` and `Parameter names`.

Figure 4.27 illustrates the view type representing the optimal core language for the OTF market for university management.

Similarly, to the view type for tourism, the package placeholder `repository` realizes the feature `Operation signatures`. It contains the classifier placeholders `Repository`, `OperationInterface`, `OperationSignature`, `Parameter`, and `DataType`. The classifier placeholder `OperationSignature` does not have the attribute placeholder `entityName` because the feature `Operation names` was omitted from the configuration. Since the feature `Parameter names` was omitted as well, the attribute placeholder `parameterName` of `Parameter` does not belong to the view type either.

The feature `Pre-/Postconditions` is realized by a set of language constructs from the package `structure`. This set contains the classifier placeholders `Precondition` and `Postcondition`. This feature is realized in the same package as the SSE extension of the PCM operation signatures that explains the name of the package. Package placeholder `protocols` stands for the package realizing the feature `Protocols`. Similar to tourism, the package placeholder `privacy` realizes the feature `Privacy`. The feature `Price` is modeled by the language constructs in the package placeholder `metadata`, which contains different kinds of metainformation about services. The package placeholder `trustmodel` stands for the package realizing the feature `Reputation`.

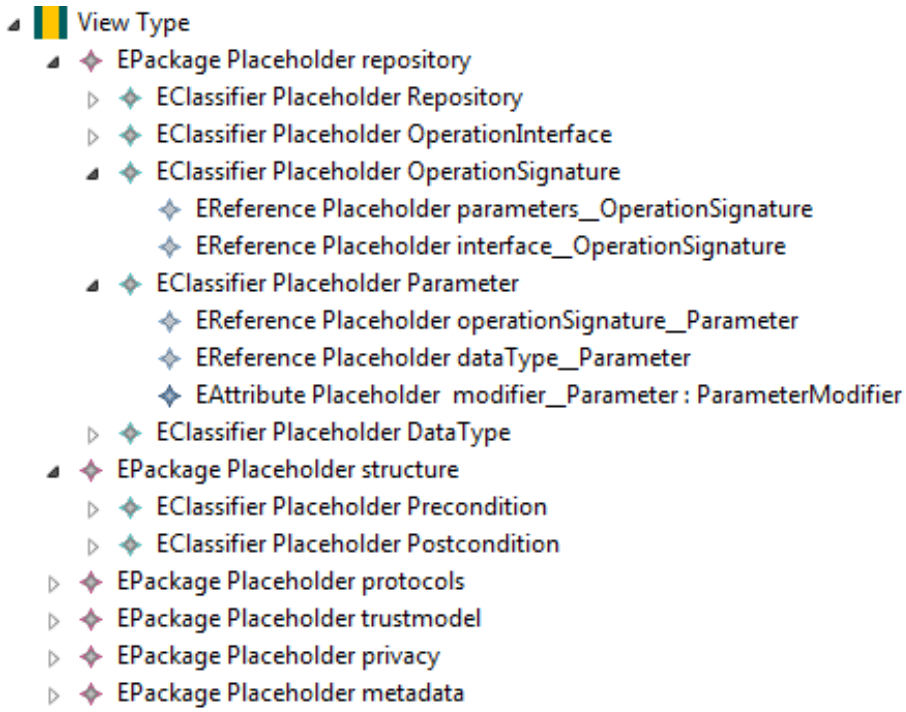


Figure 4.27: Optimal core language for university management as a view type

OTF market for water net optimization Figure 4.28 illustrates the configuration of the optimal core language for the OTF market for water net optimization. The configuration consists of four coarse-grained features. Similar to the optimal core language for university management, the feature `Operation signatures` is reduced by the fine-grained features `Operation names` and `Parameter names`.

In order to obtain the presented configuration, the configuration procedure applies the rules `cr4`, `cr6`, `cr8`, `cr11`, `cr12`, `cr15`, and `cr18` from the configuration knowledge base described in Section 4.2.3.4. The rules `cr4`, `cr12`, and `cr15` having the select-operator are executed first. These rules select the features `Operation signatures`, `Pre/Postconditions`, `Protocols`, and `Price`. All fine-grained features of `Operation signatures` are selected as well. Since the trade-off is put on effectiveness, the rule `cr18` sets to ignore all rules with the no-select operator. Thus, the rules `cr6`, `8`, and `11` are not applied, and the built configuration remains.

Figure 4.29 shows the view type for this optimal core language. The view type contains the package placeholders `repository`, `structure`, `protocols`, and `metadata` represent the coarse-grained features `Operation signatures`, `Pre/Postconditions`, `Protocols`, and `Price` correspondingly. Classifier placeholders for all classifiers from the referenced packages of the comprehensive core language are added to the view type. All placeholders for attributes and references of these classifiers are added to the view type as well.

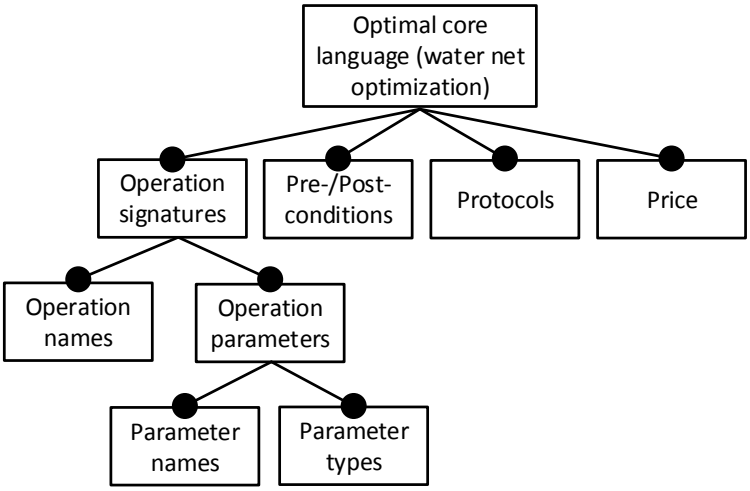


Figure 4.28: Optimal core language for the OTF market for water net optimization

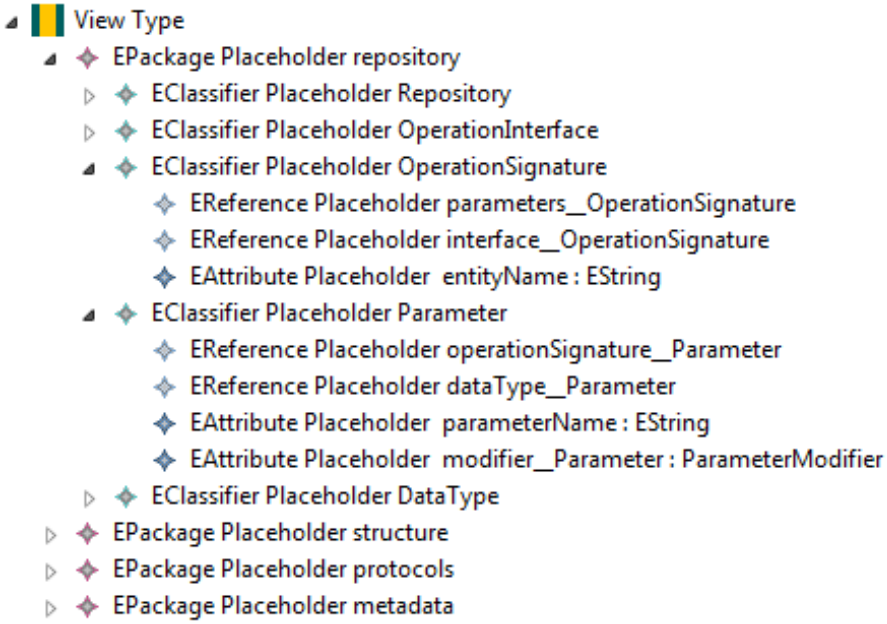


Figure 4.29: Optimal core language for water net optimization as a view type

4.3.2.4 Create Test Collections

In the third step of the evaluation, domain experts of CRC 901 build test collections for the OTF markets from the case studies. These test collections are built according to Definition 4 and contain a set of pairs each including a service specification, a requirements specification, and their expected matching result. The experts set the expected matching result, in order to state whether a described

service satisfies the given requirements specification. The domain experts describe services typically provided in the considered OTF market. They create their specifications in the optimal core language as they have enough expertise to write formal service specifications in this language.

For service matching of the created specifications, several matching steps are performed depending on the packages in the corresponding optimal core language. Matching results from these steps are aggregated into an aggregated matching result calculated for a pair. Based on the expected matching result from the test collection and the calculated aggregated matching result, the effectiveness of the matching is computed. The specifications in the test collection are created in SSE, while the pairs and their expected matching results are described using MatchBox (see Figure 4.19).

For the OTF market for tourism, the domain experts created a test collection having 100 pairs of service and requirements specifications. This test collection includes, in particular, the functionality for calculating a trip or for finding points of interest in a city. Since the optimal core language consists of two packages (see Figure 4.25), the experts had to create two specifications (one for signatures and one for privacy), which are interconnected with each other by references between the packages. The corresponding matching process consists of two matching steps (one for signatures and one for privacy correspondingly).

For the OTF market for university management, the domain experts created a test collection consisting of 100 pairs as well. The specifications from this test collection describe, in particular, services for reserving a room, for registering for an exam, and for printing an overview of grades. For these services, the experts describe their interfaces and operations, their behavior in the form of pre-/postconditions and protocols as well as such non-functional properties as privacy, reputation, and price. The matching process for the core language consists of six steps each corresponding to one of the listed properties.

The test collection for the OTF market for water net optimization consists of 10 pairs. The test collection describes services that realize the optimization of water nets by reducing parallel segments or chains in it. The experts described the interfaces and operations of these services, their behavior in the form of pre-/postconditions and protocols, and their price. The matching process contains four matching steps each corresponding to one of the listed properties.

4.3.2.5 Evaluation of the Optimal Core Languages

After the experts have created the test collections, as a final step in the evaluation, the matching runs on specifications from these test collections. The values for the matching efficiency and effectiveness are measured as defined in Definitions 6 and 15. The metrics of precision and recall are measured, in order to calculate the matching effectiveness. The mean runtime per pair is measured in milliseconds for 100 iterations of the matching executed on the test collection. Then, the efficiency as reversed runtime is calculated.

In order to check the optimality of the core languages obtained with the LOpt approach, the domain experts first decide, whether the values of the matching effectiveness and efficiency for these languages sufficiently satisfy the trade-off. In this case, it has to be shown that the obtained languages are pareto optimal as stated in Definition 16. For that purpose, the obtained core languages are slightly modified. Modifications aim at finding a core language that yields the same or higher effectiveness with the same efficiency. If such a core language can be found, then the core language obtained with LOpt is falsely considered optimal. Otherwise, the obtained core language is shown to be optimal for the given OTF market. As a result, it is shown that the LOpt approach configures optimal core language by design.

For the execution of the service matching, MatchBox provides several matching strategies. For this case study, the matching strategy with the focus on the aggregation result was chosen. The aggregation result indicates how good a service matches which the requirements considering all its specified properties. According to this matching strategy, all matching steps in a matching process are executed independently of the results from the previous steps. Thus, each matching process runs completely and executes all its constituent steps.

Based on the chosen matching strategy, adding a new language package to the optimal core language results in adding an additional step to the matching process defined for this language. Since the matching process executes all its steps, an additional step has to run for specifications of the modified core language. Thus, the mean runtime of this matching process increases by the runtime of the added step, thus, decreasing the efficiency. Therefore, core languages having more language packages result in lower values for the efficiency. As a result, it is infeasible to add new language packages to the core languages.

Alternatively, removing language packages from the core language as well as removing language constructs from these packages results in a lower runtime of the corresponding matching steps. The reason is the fact that less packages or language constructs have to be matched. Therefore, the efficiency of the matching process increases. Simultaneously, if the effectiveness remains the same after the reduction, then the current core language is not optimal.

OTF market for tourism Table 4.3 shows the results measured for the optimal core language in the OTF market for tourism.

For the core language calculated by LOpt, the effectiveness equals to `1.00` that is the highest possible value that can be reached. The runtime per pair equals to 26 ms. According to the experts' opinion, the effectiveness and efficiency obtained with this core language address the trade-off sufficiency. Therefore, the experiments to improve these values are performed.

Since adding a new language package would automatically decrease the efficiency, no new packages can be added. In order to improve the efficiency, the packages and language construct realizing the selected features shown in Fi-

Core language	Precision	Recall	Runtime	Effectiveness	Efficiency
Configured by LOpt	1.00	1.00	0.026	1.00	39
No privacy	0.56	0.75	0.017	0.64	59
No operation names	1.00	0.81	0.025	0.90	40
No parameter names	0.60	0.91	0.020	0.72	50
No parameter types	0.54	0.83	0.019	0.65	53

Table 4.3: Matching effectiveness and efficiency for the OTF market for tourism

gure 4.24 are omitted. For the core language without privacy, the effectiveness decreases by `0.36` with regard to the core language of LOpt, while the efficiency increased significantly. If the feature `Operation signatures` is reduced by different fine-grained features, then the effectiveness of such core languages decreases as well. For the core language without operation names, the effectiveness decreases by `0.10`, without parameter names by `0.28` and without parameter types by `0.35`. For all the modified languages, the efficiency increases.

As a result, the efficiency for all these variations of the core language increased while the effectiveness always decreased. This shows that the same or higher effectiveness cannot be achieved with a better efficiency (see Definition 16). Thus, the core language obtained with LOpt is optimal for the OTF market for tourism.

OTF market for university management Table 4.4 shows the evaluation of the core language configured by LOpt for the university management market.

Core language	Precision	Recall	Runtime	Effectiveness	Efficiency
Configured by LOpt	1.00	1.00	0.100	1.00	10
No privacy	0.83	0.95	0.085	0.89	12
No price	1.00	0.69	0.081	0.82	12
No reputation	0.79	0.75	0.084	0.77	12
No protocols	0.50	0.26	0.093	0.34	11
No conditions	0.39	0.25	0.066	0.31	15
No signatures	0.33	0.07	0.060	0.12	17
No parameter types	0.03	0.04	0.025	0.03	40

Table 4.4: Matching effectiveness and efficiency for the OTF market for university management

The core language configured with LOpt has the highest possible effectiveness `1.0` for the given test collection. The obtained values address the trade-off

between the effectiveness and efficiency sufficiently, therefore, possibilities to improve the calculated values are investigated.

For that, different coarse-grained features comprising the core language illustrated in Figure 4.26 are omitted first. For each reduced core language, the effectiveness decreases from `0.11` to `0.88` in comparison to the core language configured by LOpt. The efficiency of the reduced core language increases but not significantly. For the reduction, dependencies between the matching steps have to be taken into account. In particular, reducing the core language by signatures requires the reduction by conditions, protocols, and privacy due to their dependency to the signature matching result. Due to the amount of omitted steps, the matching effectiveness for the core language without signatures decreases so significant. Since the reduction of the core language did not improve the effectiveness, the optimal core language has to contain all the coarse-grained features chosen by LOpt.

Afterwards, the coarse-grained feature `Operation signatures` is reduced with regard to its fine-grained feature `Parameter types`. In this case, the effectiveness drops to `0.60` that is a significant decrease analogously to the reduction by signatures. Since the reduction by parameter types decreases the effectiveness, the core language obtained by LOpt is pareto optimal according to Definition 16 and also optimal for this OTF market.

OTF market for water net optimization Table 4.5 shows the evaluation results for the core language in the OTF market for water net optimization.

Core language	Precision	Recall	Runtime	Effectiveness	Efficiency
Configured by LOpt	1.00	1.00	0.032	1.00	31
No conditions	1.00	0.86	0.018	0.92	56
No price	1.00	0.57	0.029	0.73	34
No protocols	0.57	1.00	0.029	0.73	34
No signatures	0.43	1.00	0.010	0.60	100
No parameter types	1.00	0.43	0.010	0.60	100

Table 4.5: Matching effectiveness and efficiency for the OTF market for water net optimization

The efficiency and effectiveness obtained with the core language configured by LOpt sufficiently satisfy the trade-off. The effectiveness of the reduced core languages decreases from `0.08` to `0.40`. The efficiency of the reduced core languages increases but not significantly except for the reduction by signatures and parameter types. In this case, dependencies between the matching steps have to be taken into account. Reducing the core language by signatures requires the reduction by conditions, protocols, and privacy due to their dependency to

the signature matching result. Due to the amount of omitted steps, the matching becomes so efficient. The reduction of the core language by coarse- or fine-grained features does not reach the same effectiveness with a better efficiency. Therefore, the core language configured by LOpt is optimal.

The presented evaluation shows that the core languages obtained with LOpt are optimal for the given OTF markets. Therefore, it is shown that the LOpt approach configures optimal core languages for OTF markets by design based on its configuration knowledge base and configuration procedure.

4.3.3 Evaluation of the Requirements

This section explains how the requirements stated in Section 3.1.1 are fulfilled by the presented solution approach LOpt.

R.1.1.1 – LOpt configures an optimal core language as a view type of the comprehensive core language (see Section 4.2.3). The abstract syntax of the comprehensive core language is defined formally in the form of a metamodel (see Section 4.2.2). As a result of the view building (see Section 4.2.2.5), the abstract syntax of the optimal core language is defined formally as a metamodel as well.

R.1.1.2 – How the semantics of the comprehensive core language can be formally defined is described in Section 4.2.2.3. As a view type of the comprehensive core language, the optimal core language contains a subset of its packages and language constructs (see Section 4.2.2.5). Thus, the semantics of the language constructs of the optimal core language is described in the formal semantics definition of the comprehensive core language.

R.1.1.3 – The comprehensive core language allows the specification of structural, behavioral, and non-functional service properties, which are relevant for the automated execution of market operations in all OTF markets (see Section 4.2.2). After the configuration, an optimal core language contains those properties, which are relevant for the automated execution of market operations in a given OTF market (see Section 4.2.3).

R.1.1.4 – The configuration procedure of LOpt configures the comprehensive core language in a way to obtain an optimal core language for the automated market operations in a given OTF market (see Section 4.2.3). The configuration knowledge base contains the configuration logic customizing the comprehensive core language based on the properties of an OTF market. The evaluation in Section 4.3.2 shows that the core languages created with LOpt are really optimal for the considered markets.

R.1.1.5 – This PhD thesis considers the operation of the service matching for the design of an optimal core language. Section 4.1.1 defines the service

matching, while Section 4.3.1 describes how the service matching for the comprehensive core language is realized in the tool support. As a result of the configuration, a matching process realizing the service matching for the configured optimal core language is created. Thus, the operation of the service matching can be executed on service specifications written in the optimal core language.

R.1.1.6 – The comprehensive core language has a package structure, which facilitates the reuse and maintainability of its parts as well as supports its configuration (see Section 4.2.2). As a view type, an optimal core language partially repeats the package structure of the comprehensive core language (see Section 4.2.2.5). Thus, the optimal core language is modular.

R.1.1.7 – The comprehensive core language is designed to be orthogonal as explained in Section 4.2.2.3. For that, the integration of established service specification languages is performed, in order to eliminate redundancies in the comprehensive core language. As a view of the comprehensive core language (see Section 4.2.2.5), an optimal core language does not contain redundancies and, thus, is orthogonal as well.

R.1.1.8 – The design of the comprehensive core language includes the integration of existing service specification languages that enables their reuse (see Section 4.2.2). As a view type of the comprehensive core language (see Section 4.2.2.5), an optimal core language integrates and, thus, reuses existing established service specification languages of market actors as well.

R.1.1.9 – The comprehensive core language has a package structure, in which each package is realized by a metamodel created by reusing existing service specification languages (see Section 4.2.2). According to the MDSD paradigm followed in this PhD thesis, different model transformation techniques defined for metamodels already exist. Furthermore, the reuse of existing languages facilitates the transformation, because common language constructs are used in the comprehensive core language and existing specification languages. As a result, an optimal core language as a view type of the comprehensive core language (see Section 4.2.2.5) facilitates the transformation from other service specification languages.

R.1.2.1 – The design approach LOpt creates an optimal core language by customizing a comprehensive core language based on the given formalized properties of an OTF market (see Section 4.2). Thereby, LOpt provides guidelines for a language engineer to develop the comprehensive core language systematically. Furthermore, the configuration approach of LOpt consists of a deterministic automated configuration procedure, which works upon a configuration knowledge base containing formalized configuration logic. Thus, the design approach LOpt becomes systematic.

As described above, LOpt also creates an optimal core language satisfying the requirements *R.1.1.1*–*R.1.1.9*.

R.1.2.2 – The design approach LOpt is automated in the application phase, when a user specifies the market properties manually, and the configuration approach outputs an optimal core language fully automatically (see Section 4.2.3.1). In the design phase, when the comprehensive core language has to be created and the configuration knowledge base has to be built, a manual effort from a language engineer is required (see Section 4.2.2 and Section 4.2.3.5).

R.1.2.3 – The design approach LOpt formalizes the properties of OTF markets relevant for the configuration (see Section 4.2.3.2). LOpt leverages the formalized properties of OTF markets, in order to find an optimal core language for the given OTF market (see Section 4.2.3.1).

R.1.2.4 – The design approach LOpt uses a measurable definition of the language optimality presented in Section 4.1.2. The configuration logic of the configuration knowledge base is developed regarding this definition (see Section 4.2.3.4). Section 4.3.2 uses this measurable definition of the language optimality to show that the core languages obtained with LOpt are really optimal for the presented case studies.

R.1.2.5 – The design approach LOpt adheres to the guidelines for the design of specification languages. LOpt builds modular core languages and reuses existing specification languages (see Section 4.2.2.2). The configuration procedure of LOpt uses the view building to create core languages (see Section 4.2.2.5). In order to avoid redundancies in the core languages, the orthogonality of the comprehensive core language is guaranteed by design due to the performed language integration (see Section 4.2.2.3). The configuration procedure follows the principle of building a family of domain-specific languages through language customization, where all possible optimal core languages build a language family by customizing the comprehensive core language for different OTF markets (see Section 4.2.3.4).

R.1.2.6 – The design approach LOpt handles conflicts arising during the integration of existing languages into the comprehensive core language and during the configuration of an optimal core language. During the integration, conflicts are solved by prioritizing languages to integrate and by guidelines in the integration procedure (see Section 4.2.2.2 and Section 4.2.2.3). During the configuration, the market property `Trade-off focus` sets the focus on either matching efficiency or matching effectiveness. Using the rules based on this property solves possible conflicts in the application of configuration rules (see Sections 4.2.3.2–4.2.3.5).

5 User-Friendly Language Transformation

Chapter 4 introduces concepts to design an optimal core language for an OTF market. In order to enable the reuse of existing service specifications, the concept of language transformation into the optimal core language is tackled in this chapter. This chapter introduces the mtbe approach for a user-friendly language transformation. This approach provides a solution for the problem statement presented in Section 1.2.2. The solution satisfies the requirements on a solution of this problem statement introduced in Section 3.1.2.

This chapter presents the mtbe approach in Section 5.1 and describes the evaluation of the mtbe approach on various case studies in Section 5.2 .

5.1 The Approach mtbe

In an OTF market, the concept of language transformation is used to transform a specification written in an arbitrary service specification language used in this market into a specification in the optimal core language. It applies to languages having a definition of their abstract syntax in the form of a metamodel, and preserves the semantics of the original specification in the transformed one. If both languages have a formal semantics definition, then the semantic equivalence of their specifications can be checked using techniques described in [67].

The operation of language transformation is based on the artifact of a model transformation. A model transformation contains the transformation logic, based on which the language transformation is performed [5]. A model transformation consists of a set of transformation rules and a control structure defined for these rules. The control structure defines the control flow, in which the rules of the model transformation have to be executed. Each transformation rule has a left-hand side and a right-hand side. The left-hand side defines which language constructs of an input specification have to be transformed, while the right-hand side states how they have to be transformed.

This section presents the mtbe approach that automatically generates a model transformation, in order to define the operation of the language transformation in a user-friendly manner. Providers and requesters in an OTF market use mtbe to transform their existing service or requirements specifications into the optimal core language for the service matching. They apply mtbe for those languages, for which no mapping to the optimal core language is defined yet.

The approach mtbe applies genetic algorithms to the problem of generating model transformations from a set of examples. mtbe takes as an input a set of model mappings between example specifications written in a language of a market actor and the optimal core language. mtbe outputs a directly executable model transformation preserving the semantics of the languages based on the specified model mappings. mtbe derives the model transformation using the metaheuristic approach of genetic algorithms. Genetic algorithms allow to effectively and efficiently explore the solution space of possible model transformations, in order to find the most suitable model transformation.

Section 5.1.1 gives an overview of the mtbe approach and describes its realization with genetic algorithms. Section 5.1.2 explains how market actors create source and target models as well as the model mappings between them. Sections 5.1.3–5.1.7 introduce different genetic operators realizing the genetic algorithm of mtbe. Section 5.1.3 describes the genetic operator encoding possible solutions. Section 5.1.4 presents the genetic operator generating a model transformation based on a given encoding. Section 5.1.5 shows the genetic operator evaluating model transformations and computing their fitness. Section 5.1.6 describes the genetic operator selecting model transformations having a high fitness. Section 5.1.7 introduces the genetic operator modifying the selected model transformations, in order to create new ones with a better fitness. Section 5.1.8 discusses the quality of model transformations created by mtbe.

5.1.1 Overview of mtbe

Figure 5.1 shows the overview of the mtbe. This approach uses the technique of Model Transformation By-Example (see *R.2.2.1* in Section 3.1.2).

mtbe starts with a service provider or a service requester defining model mappings between specifications written in her language and in the optimal core language. A model mapping connects two specifications, which are semantically equivalent in the language of the market actor and the optimal core language. mtbe works on formally defined specification languages and formal specifications written in these languages created by market actors.

Service providers and service requesters know the concrete syntax and semantics of the languages they use in an OTF market. These languages should have a formal language definition (*R.2.2.2* in Section 3.1.2). In order to create model mappings, market actors have to learn the concrete syntax and the semantics of the optimal core language of their OTF market. They can learn it by using a formal or informal specification of the optimal core language or by using market-specific sets of example specifications written in the optimal core language. Such example specifications must represent certain usage scenarios of the optimal core language in the considered OTF market.

The created specifications as well as the mappings between them serve as an input for the derivation approach of mtbe. The mappings between example models have to be leveraged for the derivation (see *R.2.2.5* in Section 3.1.2).

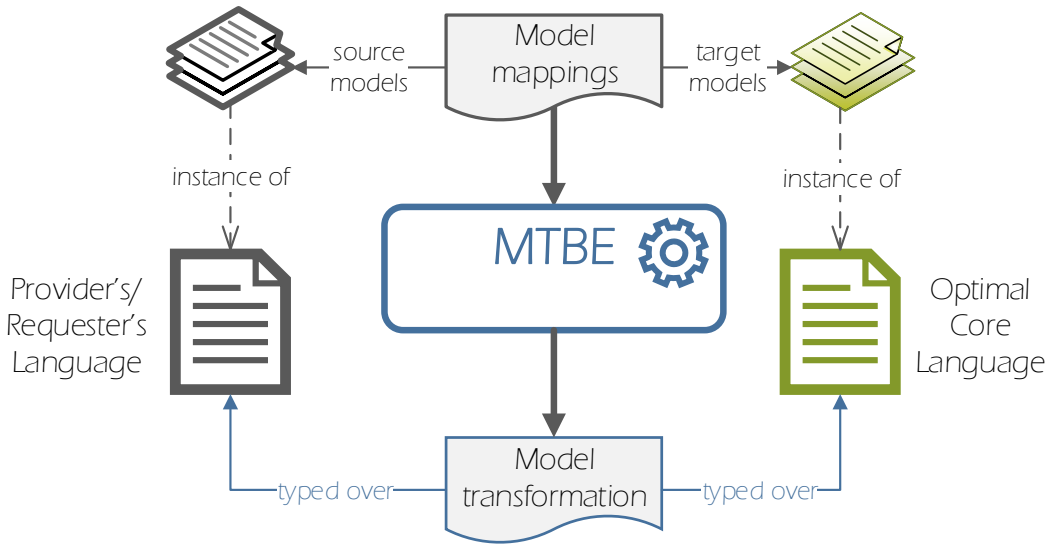


Figure 5.1: Overview of the mtbe approach

Based on these model mappings, mtbe automatically derives and outputs a model transformation between the languages of the input specifications. The resulting transformation is typed over the source language of a provider or requester and the optimal core language serving as a target language.

mtbe uses a metaheuristic for the derivation of model transformations (see *R.2.2.6* in Section 3.1.2). Several algorithms belong to the category of metaheuristic approaches [78]. Koza et al. [83] presents the characteristics of optimization problems, which the application of genetic algorithms is suitable for. Based on these characteristics, genetic algorithms were chosen as a well-suitable metaheuristic approach for the mtbe approach.

The first characteristic is the need to discover the size and the shape of the solution for the given optimization problem. In the mtbe approach, the amount of rules in the model transformation indicating its size as well as the left- and right-hand sides building the shape of the rules have to be found.

The second characteristic is the reuse of substructures. Reusable substructures in a model transformation can be found in rule sides. Promising substructures can be reused for the resulting model transformation.

The third characteristic is the need to maintain syntactic validity of a resulting model transformation. This is an important requirement, since the resulting model transformation has to be represented in a certain model transformation language. In addition, the model transformation has to be directly executable (see *R.2.1.2* in Section 3.1.2). Thus, it has to conform to the syntax of the chosen model transformation language.

The fourth important characteristic is the need to discover the "type of substructures", e.g., subroutines, iterations, or loops. Since the rules in the model

transformation have to be executed in a given control flow, finding the suitable control structure is an important task of the mtbe derivation approach.

Figure 5.2 presents the derivation approach of mtbe. The derivation approach imitates evolution and runs for a given number of generations set in advance. In each evolution run, a set of genetic operators are applied in a certain order. The approach is realized based on the concepts of the framework Opt4J introduced by Lukasiewycz et al. [88]. This modular framework for meta-heuristic optimization is chosen, because the authors enable the efficient design and development of solutions for complex optimization problems by providing such properties as strict decoupling and flexibility.

The derivation starts with the execution of the operator `Creator` (1), which creates a population of solutions. The size of the population defines the number of candidate solutions, which can be tried out in one generation. The size influences the convergence of the genetic algorithm. At the beginning of the evolution run, a solution contains a genotype only. A genotype is an encoding of a model transformation, which is independent from any model transformation language that is required in *R.2.2.8* from Section 3.1.2. The genotype contains all information necessary for the generation of a model transformation in an arbitrary model transformation language. The `Creator` generates genotypes randomly but considers the information from the model mappings.

A population containing genotypes serves as an input for the next operator called `Decoder` (2). For each genotype, the `Decoder` generates a representation of the solution (phenotype), which this genotype encodes. At this point of the evolution run, each solution in the population contains a phenotype as well. A phenotype is specific for a certain model transformation language. Thus, the `Decoder` is an operator implemented specifically for a certain model transformation language, in which the generated model transformation is described. The idea to separate the genetic representation (genotype) from the representation of the solution (phenotype) originates from Lukasiewycz et al. [88].

In the next step, the operator `Evaluator` (3) computes the fitness for the phenotypes in a population (see *R.2.2.9* in Section 3.1.2). The fitness is computed based on the predefined objectives implemented as a part of the `Evaluator`. The calculated fitness values are added to the solutions in the population.

Then, the operator `Selector` (4) picks the fittest solutions and filters out the weakest ones based on the fitness of the phenotypes. Depending on the selection strategy, the `Selector` applies different filtering techniques. Based on the amount of evolution runs to be executed in the derivation approach, the `Selector` can either transfer the selected solutions to the next genetic operator or output the solution with the highest fitness as a result.

If the evolution run continues, then the selected phenotypes are changed with the goal to improve their fitness. For this purpose, the operator `Mutator` (5) performs changes on the genotypes of the corresponding phenotypes according to a certain mutation strategy. According to the authors of [50], a crossover is

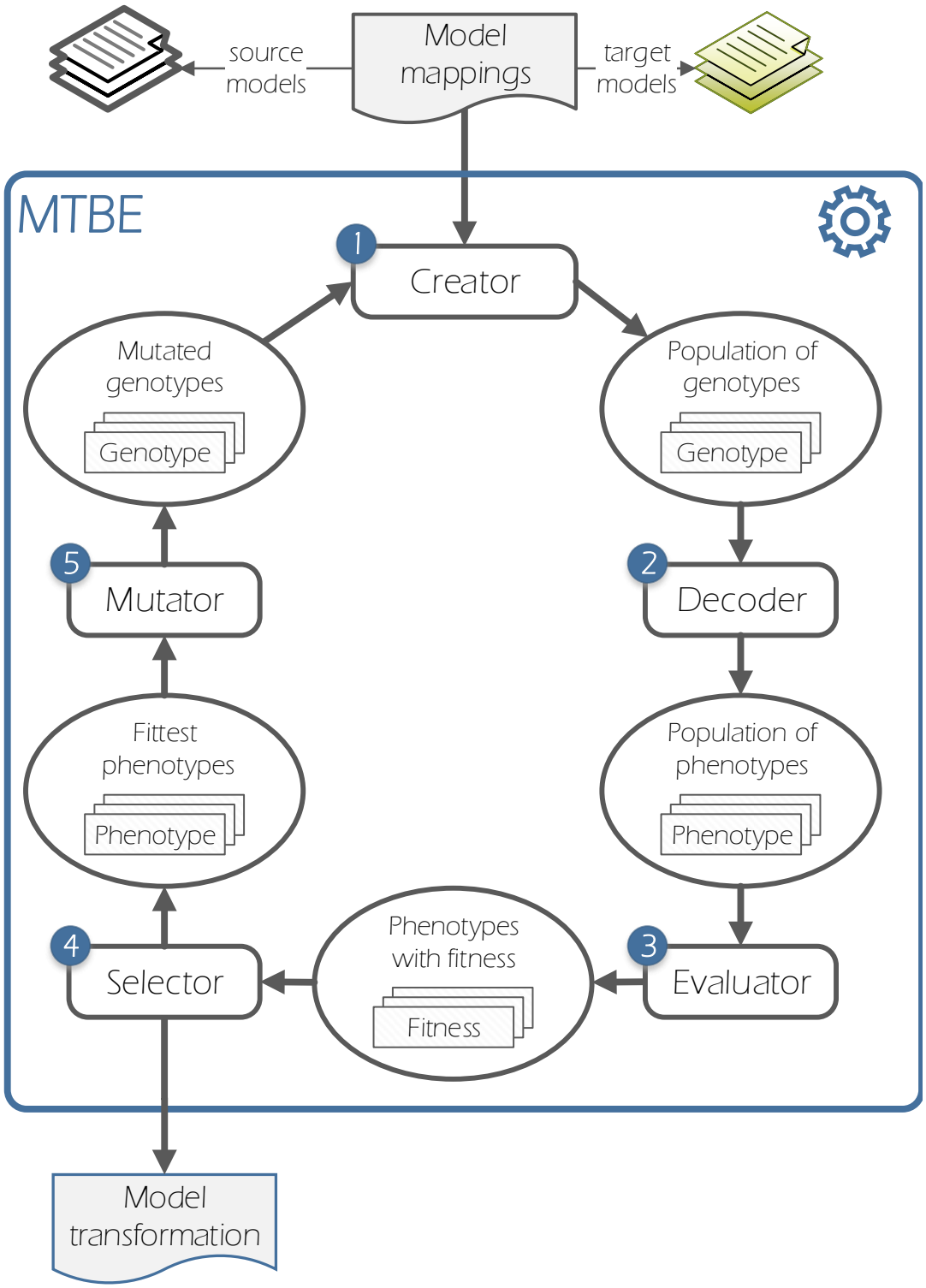


Figure 5.2: Overview of the derivation approach of mtbe

a special kind of mutation that performs several mutations steps at once. mtbe contains no special operator realizing a crossover because all suitable mutations are defined in `Mutator` with no further distinction. The mutated genotypes serve as an input for the `Creator`, which creates the new population of the next generation. In the new population, the `Creator` puts the mutated genotypes as well as creates some random ones.

As a result of the mtbe approach, the `Selector` outputs a model transformation with the highest possible fitness achieved over all generations. If the mtbe approach converges to an optimum, then several model transformations with the same highest fitness usually exist in the population of the final generations. In this case, the `Selector` outputs one such model transformation.

5.1.2 Creation of Model Mappings

Service requesters and service providers create model mappings, when they need to transform their specifications into the core language. A market actor creates specifications, i.e., models, in the corresponding editors of her language (the source language) and the optimal core language (the target language). The goal of the created model mappings is to show typical transformation scenarios between the source and the target languages in the given OTF market. A mapping exists between two models, if they are semantically equivalent but syntactically described in different languages. The market actor creating the mappings decides on the semantic equivalence of the models based on her experience with the source and target languages.

Creation process Figure 5.3 illustrates the process of the mapping creation.

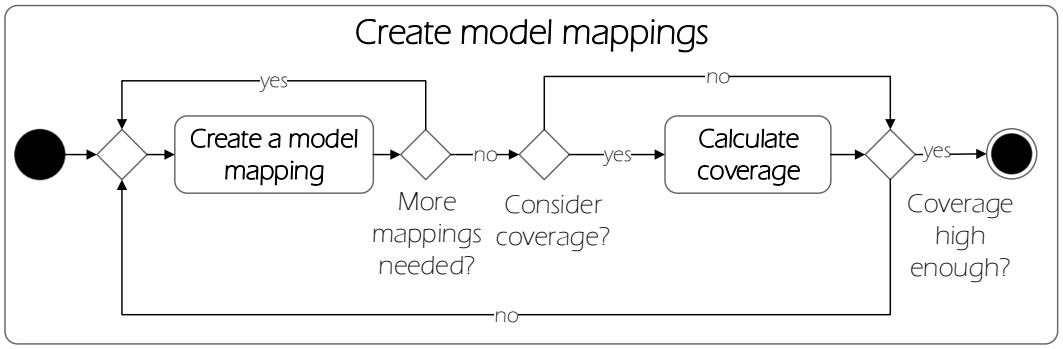


Figure 5.3: Creation of model mappings

In the first step `Create a model mapping`, a market actor creates a single model mapping. This step repeats until the market actor decides that no more model mappings are needed. For the creation of model mappings, market actors can either reuse existing specifications or create new ones from scratch.

A set of typical requirements or service specifications might already exist. Such specifications represent typical usage scenarios of a language in the considered OTF market. Market actors can directly use these specifications as a part of the model mappings. Furthermore, based on these specifications, market actors can learn the semantics of the language. If market actors decide to create models from scratch, they can start with creating simple models and continue with more complex ones. This technique is similar to the guidelines for the creation of test cases by Soltenborn et al. [133].

Since the model mappings serve as a basis for the learning process of mtbe, their quality directly influences the quality of the derived transformation. This PhD thesis assumes that market actors create semantically correct mappings between models. Since the models are qualitative per design by the market actor, the models determine the parts of the language relevant for the model transformation. In this case, the corresponding language is very expressive and the market actor uses only a subset of all its language constructs. A model transformation is derived for these language constructs only. Thus, the models do not cover the whole language but its relevant parts only.

For the evaluation of the quality of models, Fleurey et al. [49] proposes techniques for calculating the coverage. The authors define coverage criteria to evaluate a set of models. The criteria are defined for languages having the abstract syntax defined in the form of a metamodel. In this case, the step of coverage calculation together with a creation of new model mappings or modifying existing ones are performed until the obtained coverage value is high enough. After the market actor decides that the obtained coverage is sufficient, then the creation process finishes.

Criteria used for calculating the coverage are listed in the following. The criterion of the *class coverage* demands that each class of the abstract syntax is instantiated at least once in the given set of models. The *attribute coverage* demands that each attribute in the classes of the abstract syntax is instantiated with a set of representative values. According to the *association coverage*, each association has to be instantiated in the set of models with its representative multiplicities. The authors also define further coverage criteria that allow to calculate coverage values for different combinations of classes, attribute values, and association multiplicities.

Example creation of model mappings In order to illustrate how model mappings are defined, the broadly-used example of a transformation between UML class diagrams and relational data bases (RDB) is chosen as a running example for this chapter. The example is based on [154] and involves toy languages describing very simplified versions of the mentioned languages.

Model mappings Figure 5.4 shows the metamodel describing the abstract syntax of the first toy language for UML class diagrams. According to this

metamodel, a UML specification consists of classes (see `classes`), which have superclasses (see `parent`). A class contains several attributes (see `attrs`) having a type represented by `Class`. A specification also contains associations that connect source and target classes (see `src` and `dst`).

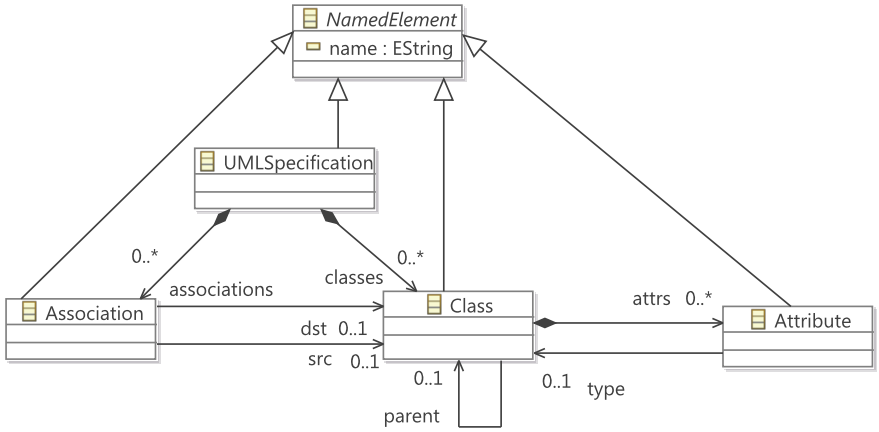


Figure 5.4: The toy language for UML class diagrams based on [154]

Figure 5.5 shows the metamodel of the toy language for relational data bases. According to it, `RDBSpec` contains modeling elements to describe a specification of a relational data base. A specification consists of tables (see `tables`) having columns and foreign keys (see `tcols` and `fkeys`). A table references one column as a primary key (see `pkey`). A foreign key references one column that contains its identifier (see `cref`) and columns of the current table (see `kcols`).

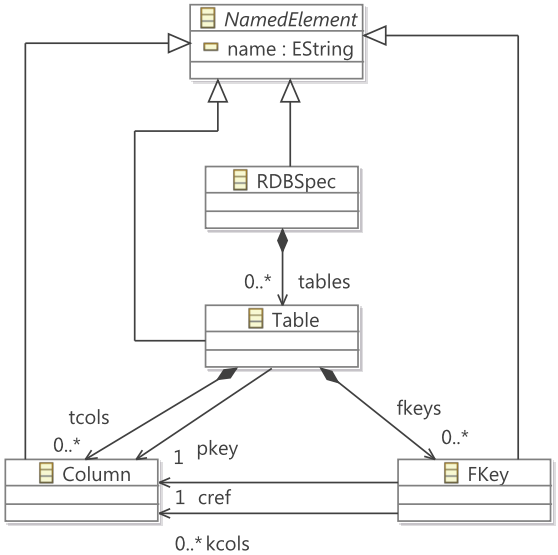


Figure 5.5: The toy language for relational data bases based on [154]

Figure 5.6 and Figure 5.7 introduce two model mappings for the presented toy languages. A market actor specifies these model mappings in the iterative process illustrated in Figure 5.3.

Figure 5.6 describes a mapping of an UML class with an attribute typed over another class. As a correspondence of this source model, the target model contains two tables one referencing another and each corresponding to a class from the source model. The foreign key models the attribute.

Figure 5.6: Model mapping 1

Figure 5.7 describes, how two UML classes connected by an association are mapped. The corresponding target model contains two tables corresponding to the classes from the source model. The corresponding target model consists of an object of type `RDBSpec`. These tables have columns containing identifiers serving as primary keys and the association is represented as a foreign key.

Figure 5.7: Model mapping 2

Coverage calculation In the following, an example calculation of coverage for the toy language for UML class diagrams is shown. The coverage for the toy language for relational data bases was calculated in the same way.

Figure 5.8 shows an overview of the coverage calculation for the toy language for UML class diagrams. During the coverage calculation, four coverage criteria were considered: class coverage, class combination coverage, reference coverage, and attribute coverage. The coverage values for these criteria and their computation are explained in detail in the following.

Figure 5.9 introduces the results of the computed class coverage. As shown in Figure 5.4, the metamodel of the toy language for UML class diagrams has 5 metamodel classes. Each of these classes is covered by one of the source models

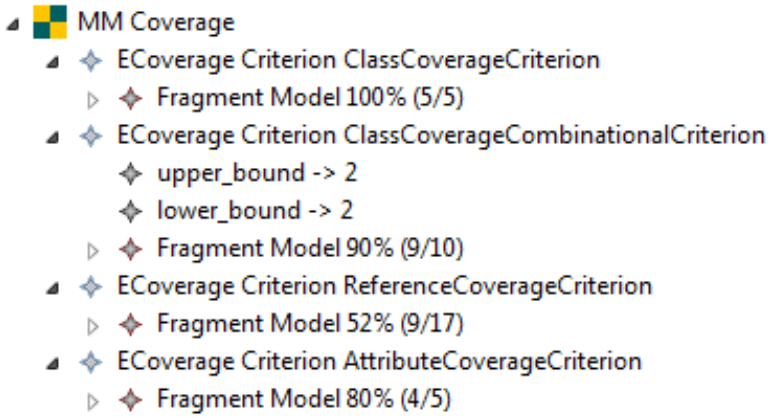


Figure 5.8: Coverage of the toy language for UML class diagrams

from the given model mappings in Figure 5.6 and Figure 5.7. Thus, the class coverage equals to 100%.

Figure 5.10 shows the results of the computed class combination coverage. This coverage criterion builds the combinations of different classes based on the given lower and upper bounds. In Figure 5.10, the bounds are set to 2. This means that all possible pairs of classes of the toy language (i.e., all combinations of two classes) are checked for being covered by the models from the mappings. As illustrated in Figure 5.4, the toy language has five metamodel classes that results in 10 pairs to cover by the models. The given models cover 9 of 10 pairs excluding the case, when the objects of types `Attribute` and `Association` appear simultaneously in one model. Thus, the coverage results in 90%.

Figure 5.11 introduces the results of the computed reference coverage.

This coverage criterion evaluates how good the metamodel references with their different cardinalities are covered in models. For the coverage calculation, different ranges of cardinalities are considered depending on the original cardi-

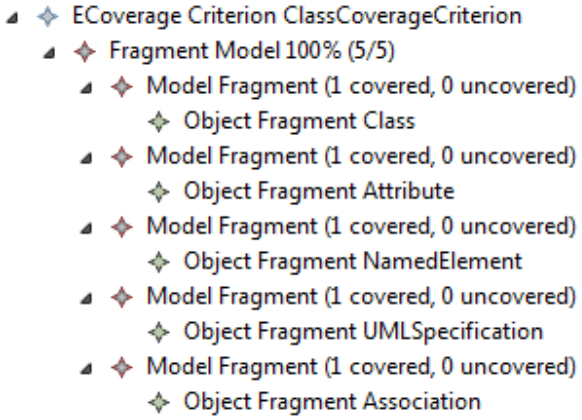


Figure 5.9: Class coverage of the toy language for UML class diagrams

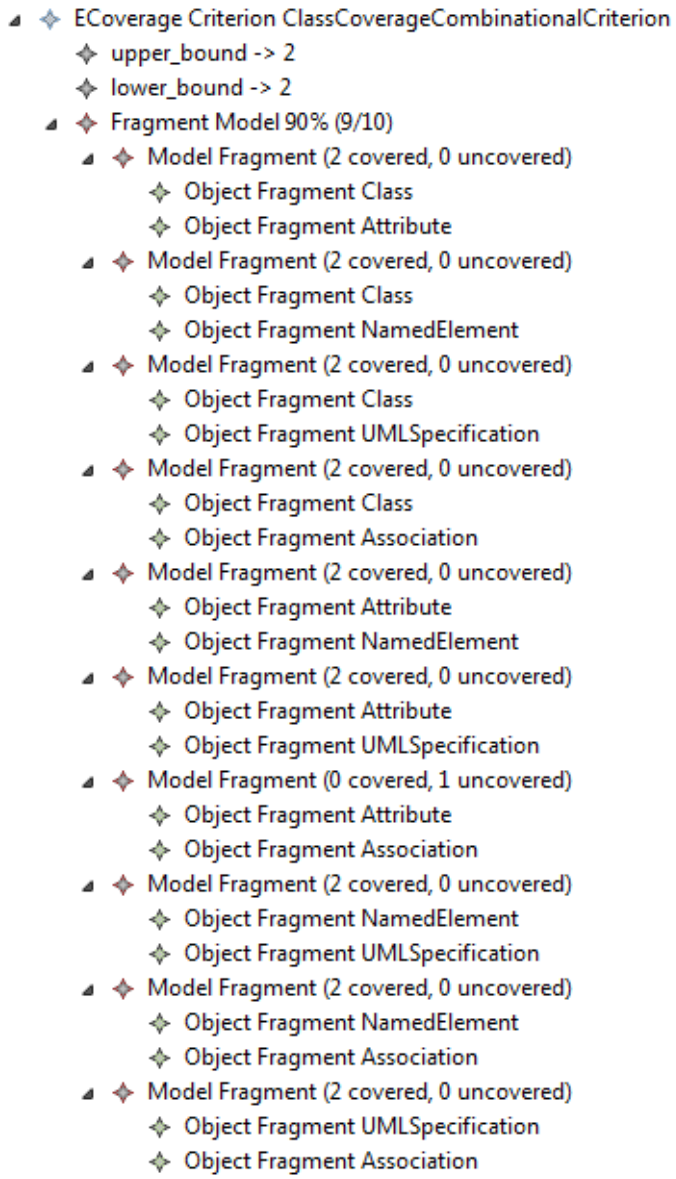


Figure 5.10: Class combination coverage of the toy language for UML class diagrams

nalities of the references and the maximal considered cardinality. To calculate the reference coverage for the running example, the maximal considered cardinality is set to 2. Thus, all references have to be instantiated with cardinalities from lower bound to upper bound in the maximal range from 0 to 2. Based on the cardinalities of the references in the metamodel illustrated in Figure 5.4, the amount of checked instantiations equals to 17. Models from the mappings cover 9 out of 17 instantiation possibilities of these references. Thus, the calculation of the reference coverage results in the value of 52%.

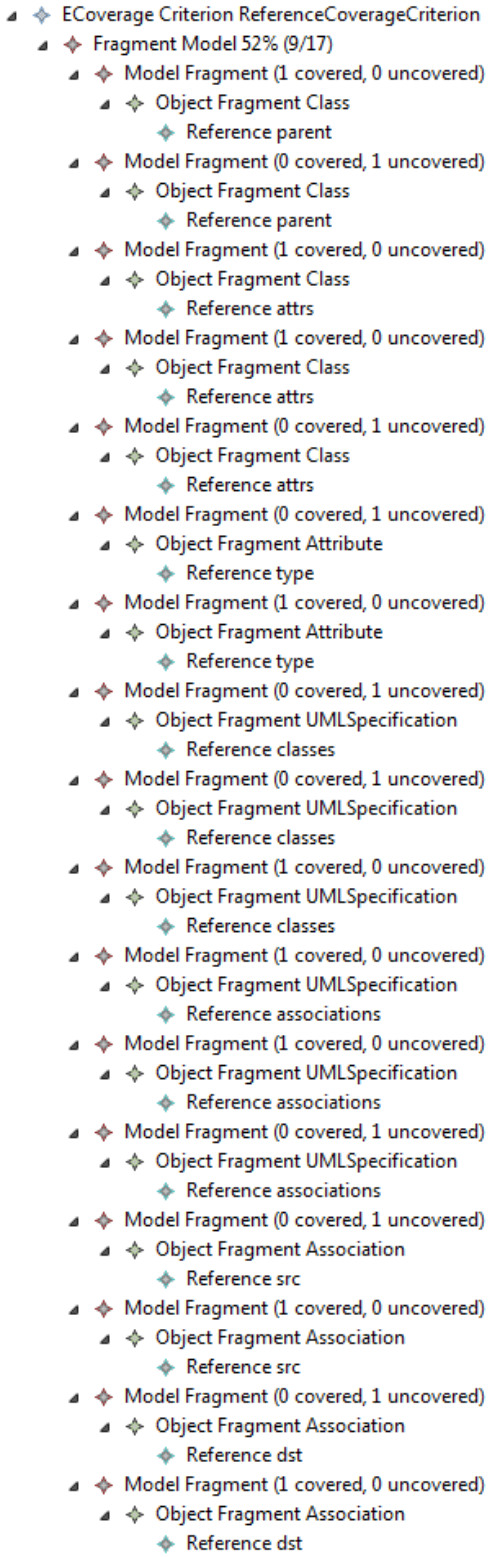


Figure 5.11: Reference coverage of the toy language for UML class diagrams

Figure 5.12 shows the results of the computed attribute coverage.

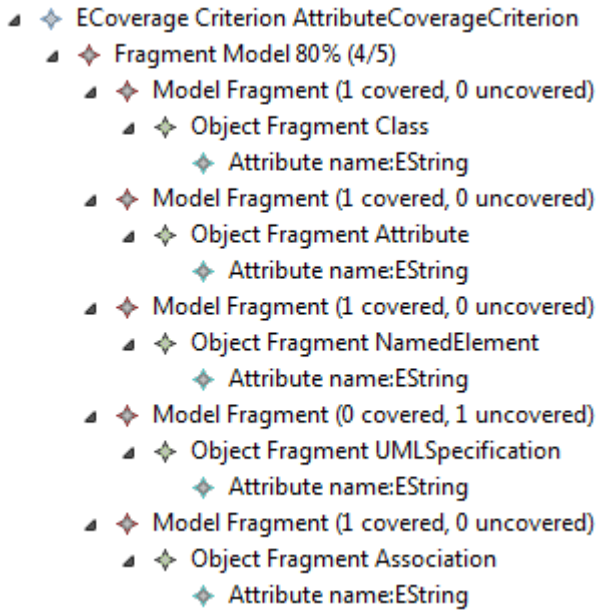


Figure 5.12: Attribute coverage of the toy language for UML class diagrams

This coverage criterion checks that all attributes of the metamodel classes are covered by the models from the mappings. This coverage criterion considers different ranges to cover for each attribute depending on its data type. In the running example, only attributes of type `EString` are considered. For this type, a corresponding attribute has to be instantiated with a arbitrary String-value at least in one of the models. Based on the metamodel in Figure 5.4, five attributes are considered (each of five classes has or inherits the attribute `name:EString`). Source models from the mappings cover 4 of 5 attributes because no objects of type `UMLSpecification` instantiates this attribute. Thus, the attribute coverage equals to 80% for the given model mappings.

Summary To sum up, this section describes a process how market actors can create model mappings serving as an input for the `Creator` operator described in the next section. The presented process allows to evaluate the quality of the model mappings by calculating the coverage of the languages by these models. Thus, the market actors can improve the quality of their model mappings and, thus, increase the quality of the model transformations built based on them.

5.1.3 Creator

The genetic operator `Creator` is responsible for creating the population of solutions for each new evolution run (see Figure 5.2). Each solution contains the genetic representation called *genotype*. The genotype is an encoding of a model

transformation in a language-independent manner (i.e. independent from a concrete model transformation language), which can be decoded in several model transformation languages. Based on the genotype, the genetic operator `Decoder` generates a model transformation in a concrete model transformation language.

Genotype The genotype used in mtbe is defined for graph-based model transformation languages with the orientation on the model transformation language Henshin [5], which is used in the `Decoder`. The genotype consists of an ordered list of model transformation rules and contains the control structure of either independent control or priority control. Independent control defines that rules are executed in an arbitrary order. After a rule is found, whose application was successful, the execution stops. Priority control defines that rules are executed according to their given priorities. After a rule applies successfully to the source model, all following rules with lower priorities are not executed.

Creator Figure 5.13 gives an overview of the `Creator`'s logic.

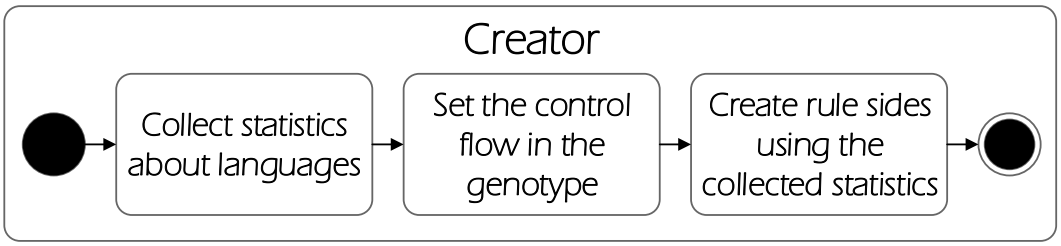


Figure 5.13: Overview of the `Creator`'s logic

In order to generate a genotype, the `Creator` leverages the knowledge from the model mappings to limit the search space for the genetic algorithm. For that, in the step `Collect statistics about languages`, different kinds of statistics are collected from the given example models and their mappings. An example of such statistics data are the probabilities of occurrence of different language constructs in the input models. The `Creator` generates rule sides with random sizes using language constructs with respect to their descending occurrence probabilities. Then, the control flow of the genotype is set in the second step. Afterwards, the rule sides are created in the final third step using the collected statistics.

Collect statistics about languages Listing 5.1 presents the statistics collected from model mappings.

The data is collected in an object called `MTBEProblem`, which contains all necessary data to describe the optimization problem of model transformation by-example. `MTBEProblem` contains the set `ModelPairs` describing the model mappings given as an input. The data structure `ModelPair` describes a single

mapping and contains a resource storing the source model (`sourceModel`) and a resource storing the target model (`targetModel`). `MTBEProblem` also contains a description of the properties of the source and target languages, which the source and target models conform to. The properties of the languages are described by the data structure `LanguageElements`.

Listing 5.1: Statistics collected from the model mappings

```
MTBEProblem:
    Set<ModelPair> modelPairs
        ModelPair:
            Resource sourceModel
            Resource targetModel
    LanguageElements sourceLanguage
    LanguageElements targetLanguage
        LanguageElements:
            Map<EClass, Double> classes2probabilities
            Integer numberOfClasses
            Map<EClass, Map<EAttribute, List<Object>>
              classes2attributes2values
            Map<EClass, Map<EAttribute, Double>
              classes2attributes2probabilities
            Map<EReference, Double> references2probabilities
            Integer maxModelSize
            EClass root
    Map<EClass, List<EClass>> classMappings
```

Since the input models may cover only a part of the language, metamodel classes, which are instantiated in the models, have to be identified. Languages considered by `mtbe` are defined using the Ecore modeling language, whose abstract syntax is shown in Figure 4.7. `EClass` represents metamodel classes. Objects instantiating classes are represented by the class `EObject` shown in Figure 5.14. Instantiated classes are determined based on the types of the objects in the models. The method `eClass()` of `EObject` is used for that.

For the instantiated classes, probabilities of their occurrence in the models are calculated. The `Creator` uses this information to add classes in the rules of the genotype according to their descending probabilities. For the calculation of probabilities, the number of models, in which a class occurs, is divided by the number of all models. This results in the occurrence probability of a class in one model (`classes2probabilities`). The number of all instantiated classes is also stored (`numberOfClasses`). Based on the instantiated classes, their instantiated attributes are identified.

For the instantiated attributes, their values assigned in the models are collected as well, in order to find out, which attributes from the source and target models may correspond to each other. The map `classes2attributes2values` stores each instantiated metamodel class with its instantiated attributes and their values, which can be found in the models. The Ecore class `EAttribute` represents attributes of metamodel classes `EClass` as illustrated in Figure 4.7.

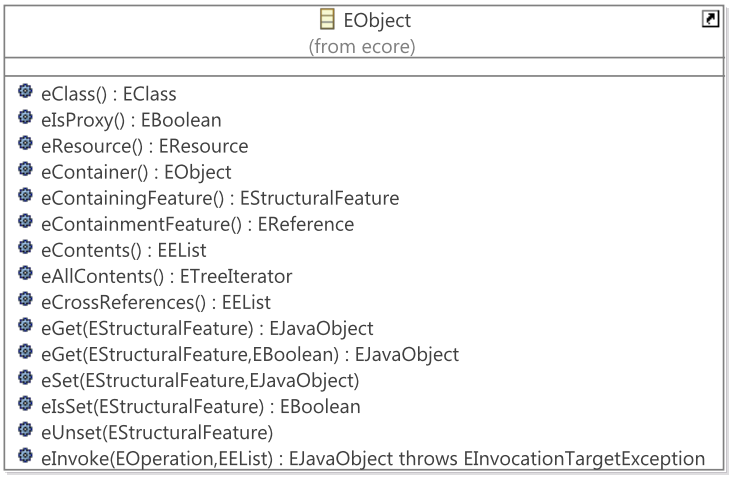


Figure 5.14: Ecore class EObject

The method `eIsSet` of `EObject` (see Figure 5.14) is invoked on an object with its attribute as an input, in order to determine, whether this attribute is instantiated in this object. The method `eGet` is invoked to get the value of the instantiated attribute.

For each attribute of the instantiated classes, its probability of occurrence in these classes is calculated. The `Creator` uses this information, in order to add attributes to the classes of the rule sides according to their descending probabilities. For each attribute of the collected classes, the number of its instantiations in these classes is divided by the number of the occurrences of these classes over all models (`classes2attributes2probabilities`).

Since the `Creator` needs to add references in the rule sides of the genotype, the information about their occurrence probabilities is required as well. For each reference contained in the collected classes, the number of occurrences of classes containing this reference as instantiated is divided by the number of occurrences of the classes containing this reference according to the metamodel. The number of occurrences are calculated over all models. The map `references2probabilities` stores the correspondence between the references and their occurrence probabilities. The Ecore class `EReference` stands for references between metamodel classes as shown in Figure 4.7. Similar to attributes, the method `eIsSet` of the class `EObject` is invoked with a reference as an input to determine, whether this reference is instantiated for a given object.

In order to limit the sizes of the rule sides created in the genotype, the information about the maximum model size is needed (`maxModelSize`). Putting no limitation on the sizes of rule sides would unlimitedly expand the search space for the derivation approach of `mtbe`. This would complicate the evaluation, which is the most expensive part of the derivation approach, and might also hinder the convergence of the genetic algorithm.

When creating rule sides of the genotype, the tree structure in metamodels described in Ecore has to be considered. In particular, the `Creator` should not add two metamodel roots into one rule side. In this case, a rule cannot apply to a model correctly as each model has a tree structure corresponding to its metamodel and, therefore, only one root. `MTBEProblem` stores the information about the root of the metamodel (see attribute `root`). For each class, it is checked, whether a containment with a target at either this class or its superclass exists. A class serving as no target for any containment is the root.

The final data of `MTBEProblem` are mappings between the source classes and the target classes corresponding to them (`classMappings`). For each instantiated class in the source models, this map stores all classes from those target models, which correspond to the source models containing objects of this class. Using these mappings, the `Creator` can limit the combinations of classes to those, which really occur in the given example models.

The language statistics calculated for the running example is illustrated in the following. Afterwards, the creation of a genotype based on this information is described.

Example of language statistics The language statistics is collected for the toy languages and model mappings introduced in Section 5.1.2. Based on the given model pairs, the toy languages can be described using the data structure `LanguageElements`. The presented model pairs are stored in the set `modelPairs` of an object of type `MTBEProblem` described in Listing 5.1.

The toy language for UML class diagrams serves as a source language for the current instance of `MTBEProblem`. The toy language for relational data bases serves as the target language. For these languages, the map `classes2probabilities` contains 4 elements, which correspond to all concrete classes of its metamodel. Thus, `numberOfClasses` equals to 4.

The probabilities of language classes are shown in Table 5.1. `UMLSpecification` and `Class` occur in each model, while `Attribute` and `Association` occur only in one model of two. Values from the map `classes2attributes2values` are also shown in Table 5.1. No attribute is added for the class `UMLSpecification` because no attribute is set for its objects. The map `classes2attributes2probabilities` contains the attribute `name` with the probability 1 because it occurs in every object of these types.

Table 5.2 presents the map `references2probabilities`, which contains the occurrence probabilities of the instantiated references. Containments with probability 1 are instantiated for each object of the classes containing them. For example, the containment `attrs` has the probability 0.25 as it is instantiated in one object of the type `Class` out of four objects (see `Model mapping 1`). The containment `associations` has the probability 0.5 as it is set for one object of the type `UMLSpecification` out of two (see `Model mapping 2`). The association `parent` is not covered by the model pairs. In this case, the modeler might

Table 5.1: Excerpt of language statistics for UML class diagrams

Language class	Class occurrence probability	Class attribute	Attribute value	Attribute occurrence probability
UMLSpecification	1	-	-	-
Class	1	name	Clerk, Manager	1
Attribute	0.5	name	Boss	1
Association	0.5	name	hasEmployee	1

left out this association deliberately because it is out of scope for the derivation of a model transformation (see Section 5.1.2).

Table 5.2: References and their probabilities for UML class diagrams

Reference	Occurrence probability
classes, dst, src, type	1
associations	0.5
attrs	0.25
parent	-

The integer `maxModelSize` is set to 4. The root is set to `UMLSpecification`.

In this example, class mappings contains all objects from each source model mapped to all objects of the corresponding target model. Therefore, in this example, no advantage of the collect mappings is gained.

The language statistics for the toy language for relational data bases is calculated in the same way.

Create the genotype Listing 5.2 presents the data structure realizing the genotype in mtbe described in Section 5.1.3. The mtbe approach is realized upon the framework Opt4J of Lukasiewycz et al. [88] and uses Opt4J data structures to realize the genotype.

Listing 5.2: Genotype in the mtbe approach

```
1 MTBEGenotype extends
      CompositeGenotype<Integer ,ListGenotype<List<EObject>>>
2 String controlUnit
```

`MTBEGenotype` extends the Opt4J class `CompositeGenotype`, which consists of multiple simple genotypes modeled by the Opt4J class `Genotype`. `CompositeGenotype` contains a map of two integers, each standing for the source or target language correspondingly. The Opt4J class `ListGenotype` defines a genotype in the form of a list of objects `EObjects` representing rules sides. Each rule side is modeled as a list of `EObjects` that define, what classes are instantiated in the

rule sides and what attribute values these classes have. The control unit used in the genotype is defined by the variable `controlUnit` of type `String`, which can be either independent or priority control unit.

The creation process starts with creating a new object of type `MTBEGenotype` and choosing the control unit. The process continues by creating rule sides. For left-hand sides of rules, a `ListGenotype` based on the source language is created and set as a first element of the `CompositeGenotype` comprising the `MTBEGenotype`. For right-hand sides, a `ListGenotype` is created for the target language and is set as the second element of the `MTBEGenotype`. To determine the amount of rules (i.e., the size of each `ListGenotype<List<EObject>>`), a random number from 1 to a given ceiling `maxAmountOfRules` is taken.

The sizes of each rule side are defined by the amount of its objects. The sizes are stored in two lists of integers: `LHS_Sizes` and `RHS_Sizes`. For each rule, the size of its rule side is set to a random number between 1 and the double number of instantiated classes of the input language (`2·number_of_language_classes`). This is a heuristic to get reasonable sizes for rule side. Any other upper bound, which is equals or higher than the number of instantiated classes, can be used as well. If the sum of all generated sizes is less than `number_of_language_classes`, then some classes instantiated in the given example models do not occur in the rules. Therefore, arbitrary elements of the rule side sizes are increased by 1 until the sum becomes equal or larger than the number of instantiated classes.

Figure 5.15 shows the process to create rule sides. The process is performed until a rule side is created for each rule number. Rule sides are created by instantiating classes of the corresponding language, their attributes and references according to their probabilities collected as a language statistics.

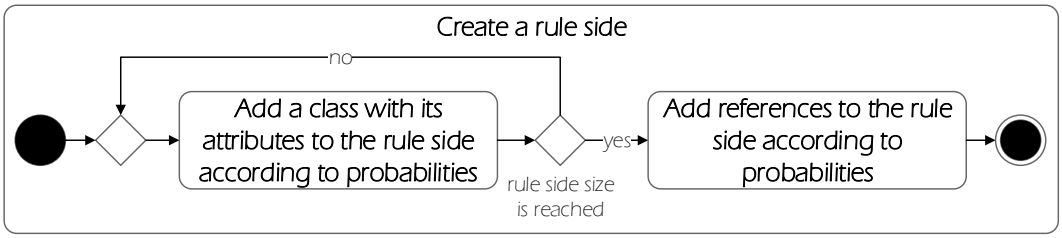


Figure 5.15: Overview of creating rule sides

To create a rule side, the probability of instantiated classes collected in the map `classes2probabilities` (see Listing 5.1) is checked to determine whether objects of these classes should be added to the rule side. If the probability of the class is 1, i.e., the class occurs in every input example model, then an object of this class with its attributes should be added to the rule side. If the class is also the root of the metamodel of the input language, then this class is added only if the rule side does not contain a root yet. This is important to preserve the tree structure of models conformed to Ecore metamodels.

After all the classes with the occurrence probability 1 are checked, the rule side size might be less than the defined size. In this case, the rule side has to be filled with objects until the size of the rule side is reached. For that, a random class from the set of instantiated classes is taken, and its occurrence probability is checked to be greater than a random probability obtained as a random real number between 0 and 1. In this case, this class is added to the rule side.

For the target language, the genotype for left rule sides is already built. Thus, only those classes of the target language should be used in the right-hand side, which correspond to the classes instantiated in the left-hand side of that rule according to the given example models (see `classMappings` in Listing 5.1). This prevents using classes in the right-hand side, which do not have a mapping to the classes from the left-hand side.

After rule sides are filled with objects typed over classes of the given metamodels, attributes of these classes have to be checked. To add attributes to objects, for each attribute of a language class, its occurrence probability from the map `classes2attributes2probabilities` (see Listing 5.1) is checked. If the occurrence probability is greater than a random probability, then the corresponding attribute is set in the object. The value of the object attribute is set to a random values from the list of all possible values of this attribute from the map `classes2attributes2values` (see Listing 5.1).

After objects and their attributes are added to the rule sides, references between these objects have to be added as well. For that, all references instantiated in example models and stored in the map `references2probabilities` (see Listing 5.1) are considered. For each reference from this set, all its possible source objects and all its possible target objects in the rule side are collected in sets `sources` and `targets`. Afterwards, depending on the type of the references, they are added differently between the collected source and target objects.

If the reference is a containment, then each target object of this reference has to be contained in some source object. Occurrence probabilities are not considered for containments because they help to ensure the correct tree structure of the rule sides. For each target object from the set `targets`, a random source object (`randomSource`) from the set `sources` is chosen. If the upper bound of the reference's cardinalities is less than the amount of targets, which the reference of `randomSource` already has, then the reference can be set from `randomSource` to an object from the set `targets`. Otherwise, if the upper bound of the reference's cardinalities is reached, then the reference cannot be added. In this case, the set `sources` is searched for an object, in which the current target can be contained. So, for each object from `sources`, it is checked, whether the upper bound of the reference is less than the amount of its targets for the current source object. If the upper bound is not reached, then the reference is added between this source and the corresponding target.

If a reference is not a containment and its occurrence probability equals 1, then the reference is added from the corresponding target to each source object, because it occurs for every class instantiated in example models. So, for each

source object from `sources`, if the considered reference is set for this source object, then the object is skipped. Otherwise, the reference is added from the source object to the target object arbitrary chosen from the set `targets`.

If a reference is not a containment and its occurrence probability is less than 1, then its occurrence probability is compared with a random probability obtained as a random number between 0 and 1. If the occurrence probability of the reference is greater, then a random source object is chosen from `sources` and a random target object is chosen from `targets`. Then the reference is added between these objects. As a result, references are added between the source and target objects in the given rule side according to their probabilities.

Example genotype Listing 5.3 shows an example `MTBEGenotype` built by the presented `Creator`.

Listing 5.3: Example genotype

```
1 MTBEGenotype:
2   (0, < <1:UMLSpecification(classes(2:Class), classes(3:Class)),
      2:Class(name="Clerk", attrs(4:Attribute)),
      3:Class(name="Manager", attrs(5:Attribute)),
      4:Attribute(name="boss", type(3:Class)),
      5:Attribute(name="boss", type(2:Class))>,
3         <1:UMLSpecification(classes(2:Class), classes(3:Class)),
      2:Class(name="Clerk"), 3:Class(name="Manager")> >),
4
5   (1, < <1:RDBSpec>,
6         <1:RDBSpec(tables(2:Table)), 2:Table(name="Manager",
      tcols(3:Column), pkey(3:Column), fkeys(4:FKey)),
      3:Column(name="MngrId"), 4:FKey(name="hasEmployee",
      cref(3:Column), kcols(3:Column))> >)
7
8   PriorityControl
```

This genotype is based on the model mappings shown in Figures 5.6 and 5.7. The representation of the genotype is based on the definition in Listing 5.2. In the example genotype, `PriorityControl` was arbitrary chosen from the list of control units. The amount of rules in the genotype is randomly determined as 2. Thus, the `ListGenotypes` in both map entries contain two lists. The lists sizes are determined randomly as 5 and 3 for the left-hand sides and as 1 and 4 for the right-hand sides.

The lists modeling left-hand sides and right-hand sides are generated based on the language definitions in Figure 5.4 and Figure 5.5. To create rule sides, the statistics about the languages and the model mappings presented in Section 5.1.3 was used. For example, the first list contains five objects. The objects `2:Class` and `3:Class` both have the attribute `name` arbitrary set with the values `"Clerk"` and `"Manager"` correspondingly. The two objects are connected as targets with the source `1:UMLSpecification` by the containment `classes`. The reference `type` is set to an arbitrary chosen object of type `Class`.

Summary To sum up, this section introduces the genotype used in the mtbe approach and the genetic operator `Creator`, which generates such genotypes. The genotype encodes a model transformation including the left-hand sides and right-hand sides of its rules and its control flow. The `Creator` leverages the knowledge from the given mappings between example models for the creation of a genotype. Based on this knowledge, the introduced methods are designed to create a genotype that encodes a possibly correct model transformation, thus, leading to a faster convergence of the genetic algorithm.

5.1.4 Decoder

This section describes the genetic operator `Decoder` (see Figure 5.2). This operator generates a representation of a solution called phenotype based on a genotype. The phenotype in mtbe is a model transformation. The `Decoder` is specific for a certain language used to describe phenotypes. In this PhD thesis, the `Decoder` is defined for the language Henshin [5].

Transformation language Henshin Henshin is a transformation language with its origin in algebraic graph transformations [5]. Model transformations of Henshin are written for languages defined as metamodels in the Ecore modeling language. These transformations are directly applicable on instances of these metamodels. The underlying formalism of Henshin allows the validation of model transformations. Existing approaches define sufficient conditions for the termination of model transformations based on algebraic graph transformations and show the confluence of such model transformations [144]. Approaches to check the syntactical and semantic correctness of transformation results are presented in Section 5.1.8.

Figure 5.16 shows an excerpt form the abstract syntax of Henshin in the form of an Ecore metamodel. A more detailed description of Henshin is given in [51].

The metamodel class `Module` represents a Henshin model transformation. It consists of a set of units represented by the abstract class `Unit`. The class `Unit` stands for the control flow as well as for the transformation rules (`Rule`). The class `Rule` represents rules for the transformation of source objects into target objects. The class `MultiUnit` serves as a superclass for three concrete units modeling the control flow of a model transformation. The concrete units `InpedendentUnit` and `PriorityUnit` have the same meaning as the control structures of the genotype introduced in Section 5.1.3. Further units like `SequentialUnit` are considered as future work.

A unit also contains a set of parameters, which aim to model the data flow between units. Parameters allows to parametrize complex model transformations and control the data flow. A parameter is characterized by its name and its type modeled as the Ecore class `EClassifier` (see Figure 4.7). In order to model the correspondence between the values of source parameters propagated

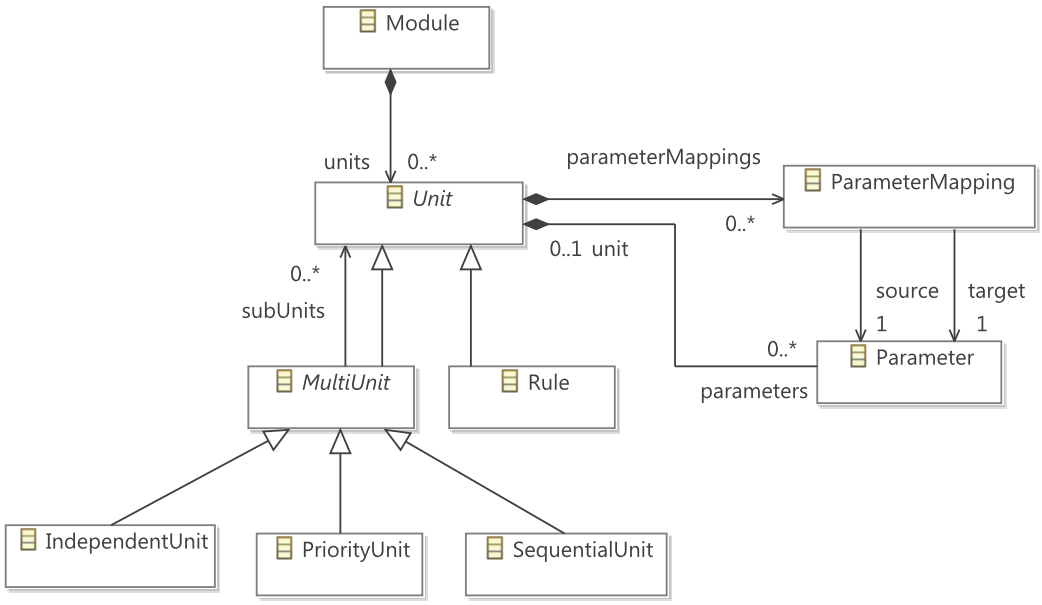


Figure 5.16: Excerpt from the Henshin abstract syntax describing units

to certain target parameters, the class `ParameterMappings` is introduced. Parameter mappings belong to a unit and store the information about the source and target parameters mapped onto each other.

Figure 5.17 presents an example model transformation written in Henshin. In this example, a priority unit determines the control flow of this model transformation. The subunits of the priority unit are two transformation rules `rule1` and `rule2` shown later in this section. The priority unit propagates the value of its parameter `name:EString` to the parameter `name:EString` of `rule1`. In order to define the data flow between these parameters, a mapping is defined from the parameter `name` of the priority unit to the parameter `rule1.name` from `rule1` (`Parameter Mapping name -> rule1.name`).

Figure 5.18 introduces an excerpt of the Henshin abstract syntax describing the concept of transformation rules.

A rule in Henshin consists of a left-hand side and a right-hand side both represented as an attributed typed graph (see containments `lhs` and `rhs`). A graph is typed over one or several metamodels, which represent the source and target languages. A graph consists of nodes connected by edges (see containments `nodes` and `edges`). A node might have no or many incoming and outgoing edges, while an edge always has one source node and one target node.

Nodes represent instances of metamodel classes, while edges represent references between them. Both classes and references come from the metamodels, which this graph is typed over. A node has a type represented by the Ecore class `EClass`, while an edge has a type represented by the Ecore class `EReference` (see Figure 4.7). A node also contains a set of attributes, which model the att-

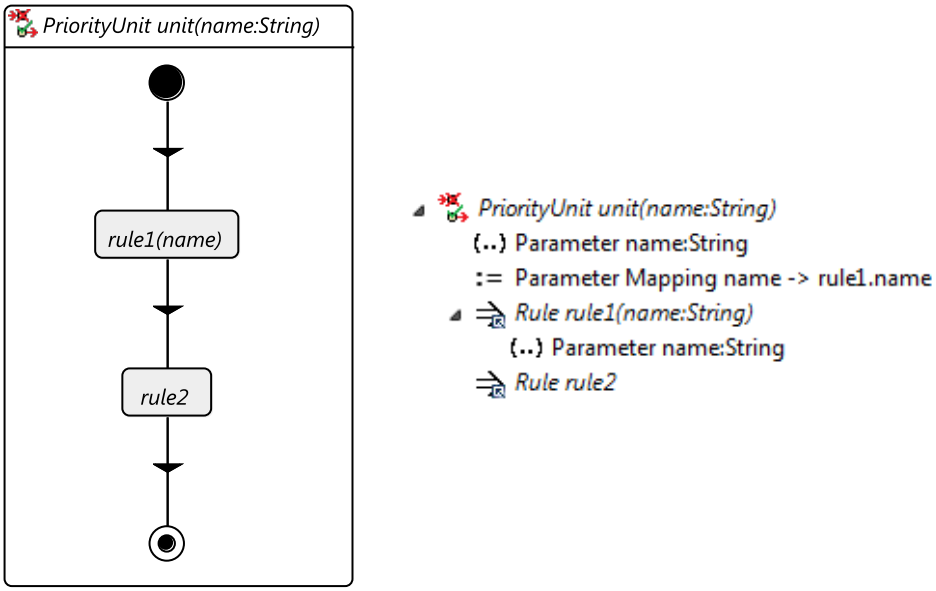


Figure 5.17: Example priority unit in Henshin

ributes of the objects being instances of metamodel classes. An attribute has a type represented by the Ecore class `EAttribute`.

The classes `Node`, `Edge`, and `Attribute` inherit from the abstract class `GraphElement` containing the attribute `action:Action`. This attribute indicates the type of action that is performed on nodes during the execution of the model transformation. The action types, which are considered for the `Decoder` are `<<preserve>>`, `<<create>>`, and `<<delete>>`. The action type of a certain node is realized by the class `Mapping` contained in the class `Rule`. This class aims at storing the correspondences between nodes in the left-hand side (`origin`) and right-hand side (`image`) of a rule.

The action type `<<preserve>>` stands for nodes, which exist in the left-hand side as well as in the right-hand side of the rule. For these nodes, the mappings between the rule sides exist and indicates their preservation. These nodes are usually marked as gray in the concrete syntax representation of Henshin rules. The type `<<create>>` stands for new nodes, which are added to the preserved nodes as a result of the rule execution. These nodes exist only in the right-hand side of the rule. Thus, these nodes do not have any correspondences in the left-hand side and no mappings for them exist. Nodes of this type are marked as green in the concrete syntax of Henshin. The type `<<delete>>` stands for nodes from the left-hand side, which are to be deleted after the rule execution. Similar to the newly created nodes, such nodes exist in the left-hand side only. Thus, no mappings for them exist. These nodes are marked as red.

Figure 5.19 presents an example of two transformation rules used as subunits in the priority unit from Figure 5.17. The graphs of these rules are mapped over the toy language for simplified UML class diagrams presented in Figure 5.4.

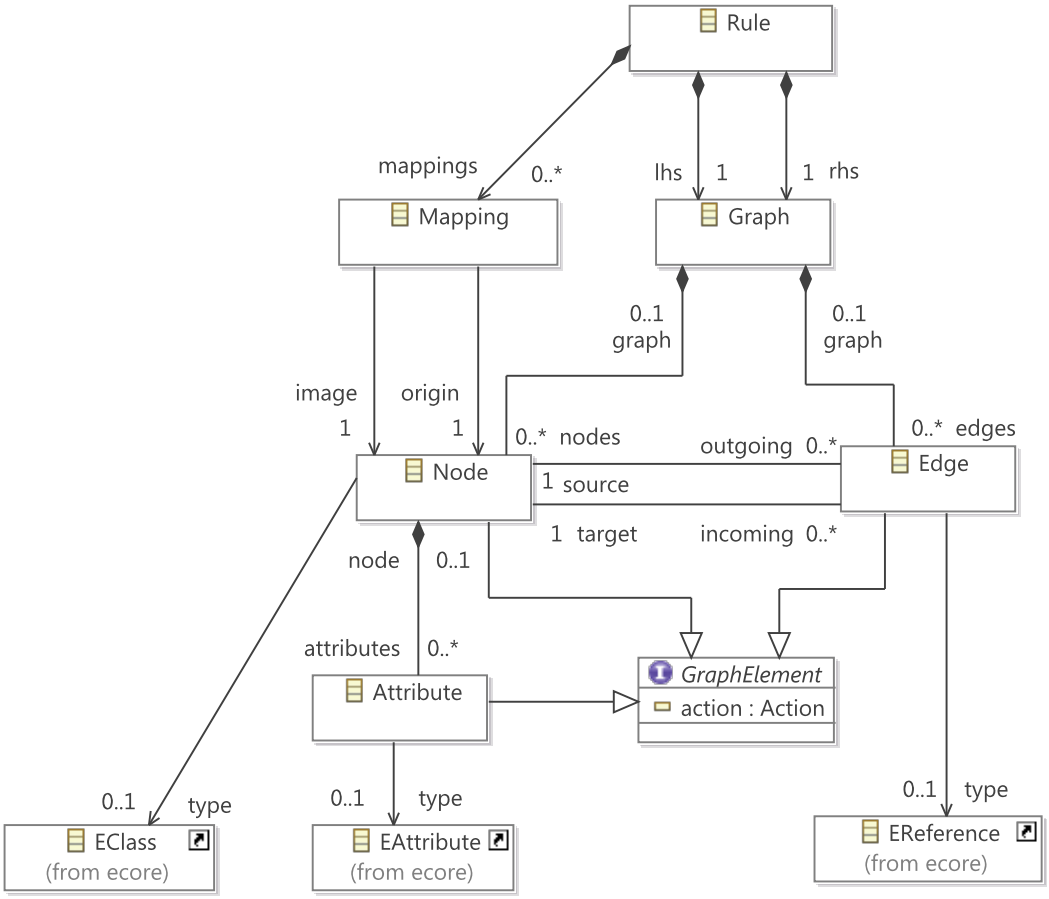


Figure 5.18: Excerpt of the Henshin abstract syntax describing a rule

The rules have nodes with types `UMLSpecification`, `Class`, and `Attribute` from the simplified UML class diagrams. The nodes are connected by the edges representing references existing between the corresponding classes. For example, the edge `classes` represent the containment between the classes `UMLSpecification` and `Class`. The node of type `Attribute` contains the node attribute `name` assigned to the value of the input parameter `name:EString` (see `name=name`).

Create the phenotype Figure 5.20 gives an overview of the logic of the `Decoder`.

The `Decoder` creates a phenotype containing a model transformation written in Henshin, which is based on the genotype built by the `Creator` in the previous step. The `Decoder` considers all information encoded in the genotype and provides a deterministic decoding algorithm. It starts by creating a new object of type `Module` (see Figure 5.16). In the next step, based on the control unit of the genotype, a corresponding unit in Henshin is created (`PriorityUnit` for priority control, `IndependentUnit` for independent control). After all the rules are created, the method creates a new phenotype `MTBEPhenotype` (see Listing 5.4).

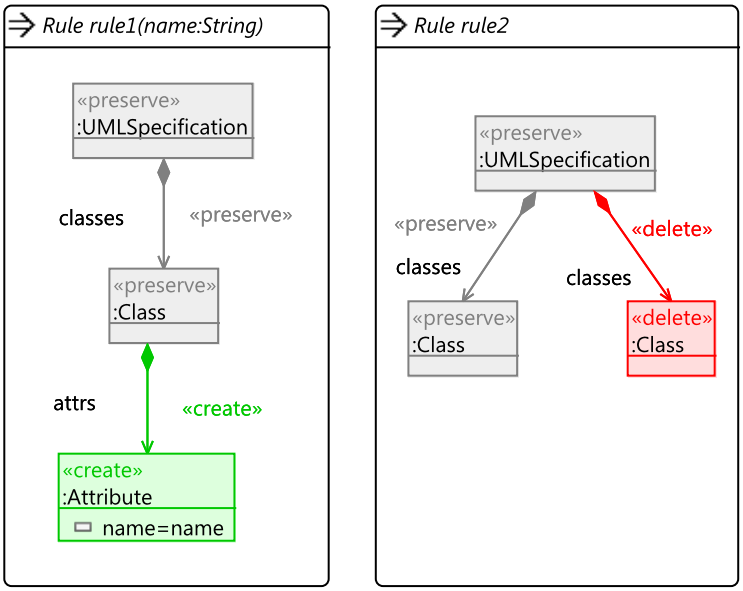


Figure 5.19: Example transformation rules in Henshin

For each new rule, nodes are added to the left-hand side or right-hand side of this rule. To the left-hand side, the objects from the corresponding rule side typed over the source language are added. Analogously, the objects from the corresponding rule side typed over the target language are added to the right-hand side. For each object from a rule side, a new node is created. This node has the type of the corresponding object and its name.

After nodes are created, edges are added between the nodes already existing in the graphs. Each object from the left- or right-hand side is considered as a source for possible edges. Then, the instantiated references of each object are checked. For each target object, an edge is created from the node corresponding to the source object to the node corresponding to the target object. The type of the edge is set to the current reference. The action of the edge is set to the action performed on its source node. By design source and target nodes have the same action type, otherwise, an edge cannot be added.

Attributes are added to nodes, which are already contained in the rule graphs.

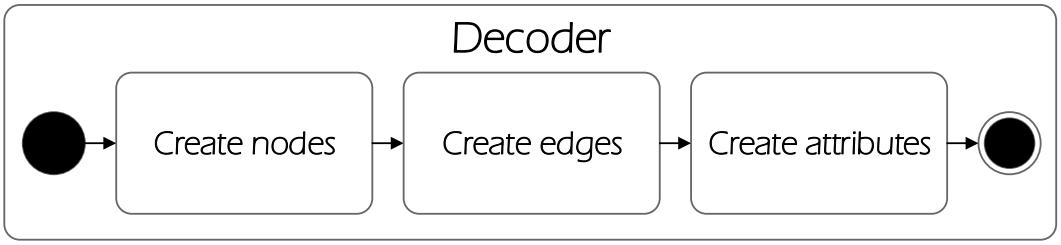


Figure 5.20: Overview of the `Decoder`'s logic

For each object, if its attribute is set, then a new object of the Henshin type `Attribute` is created (`newAttribute`), and the attribute is added to the node corresponding to the current object. The action of `newAttribute` is set to the action of its node.

The value of `newAttribute` can be in Henshin either a parameter name or a concrete attribute value [5]. The `Decoder` creates an attribute value always set to a parameter. For that reason, an existing attribute in the rule that has the same type and the same value as `newAttribute` is searched. If such an attribute can be found, then the value of that attribute is taken for `newAttribute`. Otherwise, a new parameter is created. The new parameter has the type of the object attribute. Finally, the value of the new attribute is set to the parameter's name. This means that, during the rule application, the value of `newAttribute` is set to the value of this parameter.

Example phenotype Figure 5.21 shows the example phenotype using the concrete syntax of Henshin. The phenotype is built by the presented `Decoder` based on the example genotype presented in Listing 5.3. The example genotype encodes a model transformation containing two rules (`rule1` and `rule2`) and the priority control (`prio`). Each rule is invoked with the parameters `par_0` to `par_3` mapped from the priority unit to the corresponding rule.

For each object from the lists of the example genotype, a node is created in the corresponding rule side. The left-hand side of `rule1` consists of five nodes typed over `UMLSpecification`, `Class`, and `Attribute` connected by the edges `classes` and `attrs` representing containments and the edge `type` representing references. The nodes have `<<delete>>` as action because these nodes exist in the graph of the left-hand side only. The right-hand side of `rule1` contains one node of type `RDBSpec`. It has `<<create>>` as action because it exists in the graph of the right-hand side only. Base on the instantiated attributes of the objects in the genotype, the node attribute `name` of type `EString` is created. The values of this attribute are set to the parameters `par_0` to `par_3`. In the left-hand side of `rule2`, since no attributes `name` with the same values `"Clerk"` and `"Manager"` already exist in the node, new parameters `par_0` and `par_1` are created. The attributes of the nodes of type `Class` are assigned to these parameters.

Summary To sum up, the presented genetic operator `Decoder` generates model transformations described using the graph-based model transformation language Henshin. The `Decoder` can represent the explicit control flow chosen in the genotype as well as the generated rule sides. The chosen language Henshin is close to the presented definition of the genotype that gives an advantage of a rather straight-forward generation of the phenotype. Henshin also supports the data flow between different objects that allows a stronger mapping of the object from the left- and right-hand side of a rule. Section 5.1.8 describes existing approaches to verify such properties as termination, confluence, syntactical and

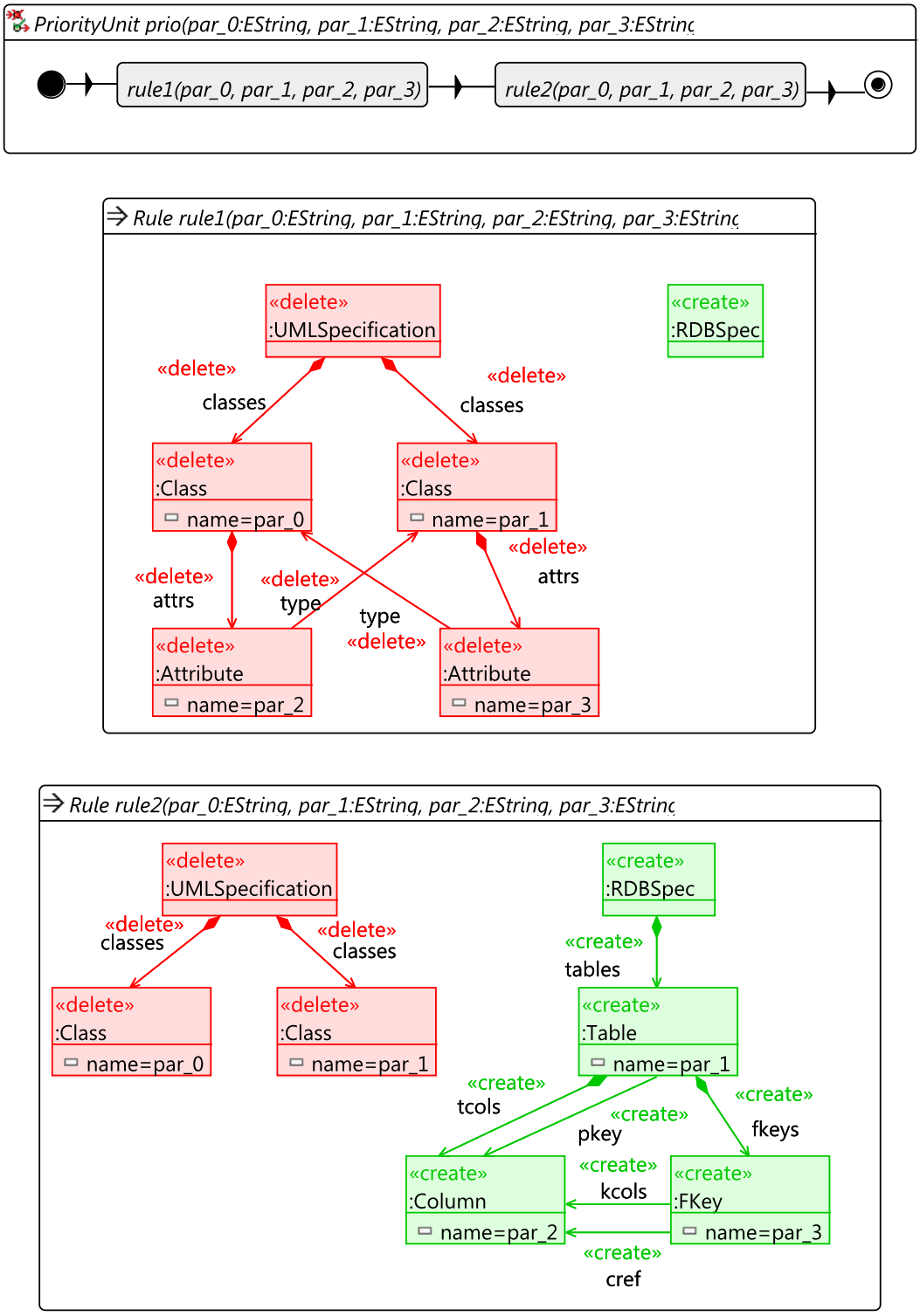


Figure 5.21: Example phenotype

semantic correctness for model transformations written in Henshin, allows to check and increase the quality of the resulting model transformation.

5.1.5 Evaluator

The genetic operator `Evaluator` evaluates the fitness of phenotypes (see Figure 5.2). As a result of this evaluation, each phenotype gets a fitness value characterizing its suitability as a solution for the given MTBE problem. For that purpose, the `Evaluator` defines objectives for the evaluation of phenotypes and metrics to measure these objectives. The `Evaluator` also sets the optimization goals for each objective, e.g., whether the value of an objective has to be minimized or maximized. Measurements for the objectives are defined depending on the model transformation language chosen in the `Decoder`. The `Evaluator` utilizes the specifics of Henshin using its graph matching.

Phenotype fitness Listing 5.4 shows the definition of the phenotype fitness.

Listing 5.4: Fitness of the phenotype in the mtbe approach

```
1  MTBEPhenotype:
2      Module module
3      double maxModuleCoverage
4      Map<Rule, List<BestRuleModelMatch>> resultingMatches
5      BestRuleModelMatch:
6          Rule rule
7          URI model
8          PartialMatchInfo partialMatchInfo
9          double ruleModelCoverage
10     Map<Rule, Map<URI, Double>> rule2model2maxCoverage
11     Map<Rule, Map<ModelPair, Double>> rule2modelPair2maxCoverage
```

`MTBEPhenotype` contains the model transformation generated by the `Decoder` (`Module module`). The objective used in the mtbe approach is the coverage by model transformation (`double maxModuleCoverage`). This objective defines, how good the resulting model transformation covers the model pairs from the mappings provided by the `mtbe` user. The coverage is a double value between 0 and 1 that has to be maximized. The coverage of 1 means that, each source model from the input model pairs can be transformed into the corresponding target model from the mappings. In other cases, the coverage is lower. The module coverage is evaluated using the graph matching of Henshin. The graph matching determines matches of the model transformation to a given model. The best matches per rule are stored in the map `resultingMatches`.

The information about a single match is stored in the data structure `BestRuleModelMatch`. It contains a rule (`Rule rule`) and a model (`URI model`) for the match. The information about the best match of this pair of rule and model is represented using the data structure `PartialMatchInfo` explained later in this section. The coverage by a rule of a model described in the match is set to

`ruleModelCoverage`. Based on the resulting matches, two maps `rule2model2maxCoverage` and `rule2modelPair2maxCoverage` are computed. The first map stores the maximal coverage achieved by a certain rule for a certain model. The second map stores the maximal coverage achieved by a certain rule for a certain model pair. Finally, the module coverage is computed based on these data.

Evaluate the phenotype Figure 5.22 shows an overview of the `Evaluator`.

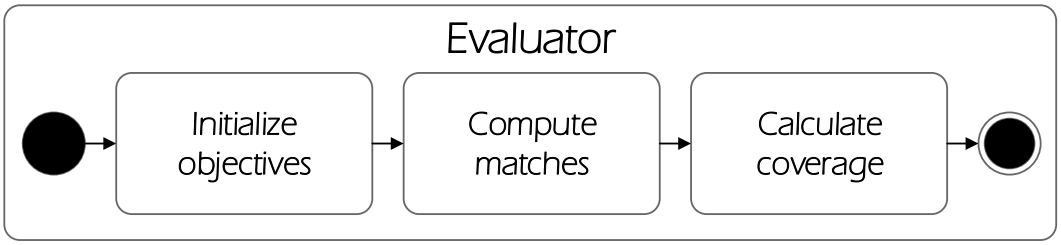


Figure 5.22: Overview of the `Evaluator`'s logic

The `Evaluator` starts by initializing objectives describing the phenotype fitness, e.g., `maxModuleCoverage`. Afterwards, the graph matching of Henshin is used to compute the matches of the model transformation and the models from the input mappings (step `Compute matches`). Finally, the coverage is calculated as an objective based on the computed matches (step `Calculate coverage`). In the current version of `mtbe`, the `Evaluator` uses one objective to evaluate the fitness of the population phenotypes. Multi-objective optimization in the `mtbe` approach is considered as future work.

For the computation of matches, the method `findAllMatches` in the Henshin class `InterpreterUtil` exists [51]. The method `findAllMatches` searches matches for a transformation rule and a model. A match contains correspondences between the nodes of the left-hand side of a rule applicable to the given model to the objects of the model. The Henshin graph matching considers each node in the left-hand side as a variable in a constraint-satisfaction problem [149]. For the variables, their solution spaces are created from the model objects serving as possible matches.

The original graph matching of Henshin finds *complete* matches. For a complete match, all nodes from the left-hand side must have a target node in the source model. In other words, the left-hand side has to be either the same as the source model or its subgraph. If a left-hand side is larger than the source model, then Henshin does not output any matches for this case as such rule is not applicable to the model.

Since the model transformations generated by the `Decoder` are subjects of genetic evolution, their coverage might be rather low at the beginning but increase during the improvement of the model transformations. As a result, the fact that Henshin always outputs complete matches only is a strong limitation for

the mtbe derivation approach. This limitation would lead to many phenotypes evaluated as having no matches, while the left-hand sides of their rules might match to the source models, if unsuitable nodes would be removed. Therefore, in order to extent the evaluation possibilities using Henshin, its graph matching was extended by the author of this PhD thesis.

The method `findAndReportMaximalPartialMatch` represents this extension. It aims at finding partial matches of a rule to a model. This covers the case, when the left-hand side is larger than the model but its subgraph yields a complete match with this model. The extension tries to find a complete match for a rule first. If a complete match for this rule does not exist, then the rule is iteratively reduced by one node producing a set of new rules. If a rule consists of N nodes, then, after the first reduction, N reduced rules are created having $N-1$ nodes each. The method continues by trying to find a complete match for the reduced rules. If a complete match exists, then this match is output as a partial match for the original rule. Otherwise, the reduction procedure recursively repeats for the already reduced rules. This procedure continues until either a complete match is found or the original rule becomes empty.

The worst run time complexity of `findAllMatches` depends on the worst run time complexity of the underlying graph matching. For Henshin, this complexity equals to the worst run time complexity for solving the constraint satisfaction problem, which the graph matching problem is transformed into. This problem is knows to be NP complete [122]. As a result, the worst case run time complexity is exponential to the size of the left-hand side of the input rule. This is also the run time complexity of the method `findPartialMatchesPerRule`.

The results of the execution of graph matching are stored in a report `PartialMatchReport` that stores information about complete or partial matches computed for different rules and models. Based on this report, found matches are added to the resulting matches of the phenotype (see Listing 5.4).

Listing 5.5 presents the data structure of `PartialMatchReport`. It contains a mapping between a rule and the information about its match, which has the best coverage over all considered models. The mappings are stored in `rule2info`.

Listing 5.5: Report of matching results

```
1  PartialMatchReport:
2      Map<Rule, PartialMatchInfo> rule2info
3      PartialMatchInfo:
4          Match match
5          Rule originalRule
6          ModelURI model
7          Graph delta
8          double coverage
9          double modelCoverage
10         double matchCoverage
```

`PartialMatchInfo` stores the match and the original rule (in the case of a partial match). If a computed match is partial, then this match is complete for

a reduced rule and contains a reference to the reduced rule only. The original rule is also the rule used in the map `rule2info`. The URI of the model `URI model` is needed to know, which model this match was computed for.

For the genetic operator `Mutator`, the information about the difference between the match and the original rule is needed. This information is used for steering the modification of genotypes with the goal to find an optimal solution. This difference is stored as a graph `delta`.

The coverage of the model by the rule is stored as a number between 0 and 1 (`double coverage`). The value of `coverage` is computed based on two further coverage values: `modelCoverage` and `matchCoverage`.

The value of `modelCoverage` indicates how many nodes of the model are covered by the match. Since the graph matching outputs complete matches, the information about how much of a model can be transformed based on this match is needed. The formula for calculating `modelCoverage` divides the number of nodes in the match (`match.rule.lhs.nodes.size`) by the number of nodes in the model (`graph.size`).

The value of `matchCoverage` indicates the difference between the reduced rule and the original rule. This value shows how many nodes and edges of the original rule are covered in the match. In the case of a partial match, the information about how good the original rule match to the model is needed for the evaluation. The formula used for the calculation of `matchCoverage` divides the sum of nodes and edges of the match by the sum of the nodes and edges of the original rule. The final coverage is calculated as an arithmetic mean of the values for `modelCoverage` and `matchCoverage`.

As the third step `Calculate coverage` (see Figure 5.22), the coverage for the whole model transformation can be calculated based on the resulting matches in the phenotype. It starts by creating a new instance of type `Objective` representing the coverage. For the calculation, the following information is collected first. A map `rule2model2maxCoverage` is created that stores the information about the model, which each rule covers the best. For the calculation of the module coverage, the map `rule2modelPair2maxCoverage` stores model pairs with the maximal coverage by each rule. This coverage is computed by obtaining the best average coverage of a model pair based on values from `rule2model2maxCoverage`.

Based on the constructed map, the module coverage can be calculated. The coverage calculation depends on the type of the control flow used in the module. For the priority control, the coverage of each model pair by the highest priority rule, which has a match for this model pair, is computed. These coverage values are summed up for all model pairs and divided by their number. Finally, the value of the objective is set to the sum of all coverages in `moduleCoverage` divided by the number of model pairs in `MTBEProblem.modelPairs`.

Evaluation of the example phenotype In the following, the fitness of the phenotype described in Section 5.1.4 is shown.

For each model pair, maximal (partial) matches with each rule of the model transformation are computed. Based on this information, the best match for each model with each rule is found. For this example, 4 best matches are found for the combinations of 2 rules with 2 model pairs.

For `rule1` from Figure 5.21 and the `Model mapping 1` from Figure 5.6, the best matches are as follows. A partial match is found for the source model and the left-hand side of the rule by reducing the left-hand side by a node of type `Attribute`. Thus, the calculated match coverage equals to $8/11 \approx 0.73$. The model coverage of this partial match equals to $5/5 = 1$. As a result, the overall coverage for the source model equals to $(0.73 + 1)/2 \approx 0.87$. For the target model, a full match is found for the right-hand side of the rule leading to the match coverage of 1. The model coverage is low and equals to $1/7 \approx 0.14$. The overall coverage for the target model equals to $(1 + 0.14)/2 \approx 0.57$.

For `rule1` and the `Model mapping 2` from Figure 5.7, the best matches are as follows. A partial match is calculated for the source model by reducing the left-hand side by all nodes of type `Attribute`. This results in match coverage of $5/11 \approx 0.46$. The model coverage equals $3/4 \approx 0.75$. The overall coverage for this source model equals to $(0.46 + 0.75) \approx 0.61$. For the target model, the model coverage equals to 1 because of a full match. The model coverage is $1/7 \approx 0.14$. The overall coverage results in $(1 + 0.14) \approx 0.57$.

For `rule2` from Figure 5.21 and the `Model mapping 1` from Figure 5.6, the following best matches were found. For the source model, a full match is found resulting in match coverage of 1. The model coverage is $3/4 = 0.75$. The overall coverage for the source model results in $(1+0.75) \approx 0.88$. For the target model, a partial match was computed by reducing the right-hand side by the node of type `FKey`, which has faulty references to other objects. Thus, the match coverage is $6/10 = 0.6$. The model coverage equals to $3/7 \approx 0.43$. The overall coverage for the target model results in $(0.6 + 0.43)/2 \approx 0.52$.

For `rule2` and the `Model mapping 2` from Figure 5.7, the found best matches are as follows. For the source model, the match coverage equals to 1 because of the found full match. The model coverage is $3/4 = 0.75$. The overall coverage of the source model is $(1 + 0.75)/2 \approx 0.88$. For the target model, a partial match is found by reducing the right-hand side by the node `FKey`, which has faulty references to other objects. The match coverage results in $6/10 = 0.6$. The model coverage is $3/7 \approx 0.43$. The overall coverage for the target model equals to $(0.6 + 0.43)/2 \approx 0.52$.

Then, the coverage of the whole model transformation is obtained. For that, the coverages of each model pair by each rule are calculated first. This calculation is performed as an average of the coverage values for source and target models presented above. Thus, for `rule1` and the `Model mapping 1`, the coverage equals to $(0.87+0.57)/2 = 0.72$. For `rule1` and the `Model mapping 2`, the coverage is $(0.61 + 0.57)/2 = 0.59$. For `rule2` and the `Model mapping 1`, the

coverage equals to $(0.88 + 0.52)/2 = 0.7$. For `rule2` and the `Model mapping 2`, the coverage is $(0.88 + 0.52)/2 = 0.7$.

Based on these coverage values, the coverage of the example model transformation can be calculated. In this transformation, `rule1` has the highest priority. Therefore, the average coverage of model mapping by `rule1` is the coverage of the model transformation. This coverage equals to $(0.72+0.59)/2 \approx 0.66$. Thus, the fitness of the example phenotype is calculated as 0.66.

Summary To sum up, the genetic operator `Evaluator` calculates the fitness of the model transformations generated by the `Decoder`. For that, the `Evaluator` introduces the objective of the coverage of the input model mappings by the generated model transformation. For the coverage computation, the graph matching of Henshin is extended, in order to compute possible complete and partial matches of each transformation rule of the model transformation with each model from the model pairs. Using the information about the matches, the model transformation is evaluated based on the existence of a complete match (match coverage) but also based on the extent of their coverage of the models (model coverage). The `Evaluator` also collects the information about matches, which the `Mutator` uses to improve the model transformations.

5.1.6 Selector

After the fitness of phenotypes is evaluated, the genetic operator `Selector` chooses solutions with the fittest phenotypes according to a certain selection strategy (see Figure 5.2). Opt4J realizes the elitism selection strategies such as NSGA-II [36] and SPEA2 [164] for single- and multi-objective optimization. The elitism strategy retains the solutions with the best fitness unchanged in the next generation and, as a result, prevents the loss of good solutions in the course of mutation.

Select phenotypes When choosing a selection strategy for mtbe, several decisions have to be considered. As first, the decision has to be made, whether all solutions in the population or only their subset have to be mutated. In the latter case, the selector has to build the subset of solutions to be mutated. As second, the decision has to be made, whether new randomly generated solutions are added to the population. In this case, the selector has to built a set of solutions, which have to be preserved in the population.

The `Selector` of mtbe follows the elitism strategy and, thus, selects only a subset of solutions for the mutation. The approach benefits from the preservation of a subset of optimal solutions in the population. As a result, the final evolution run always contains the fittest solution.

The possibility to preserve a set of solutions in each population and fill the remaining part with newly generated solutions is considered. However, due to a

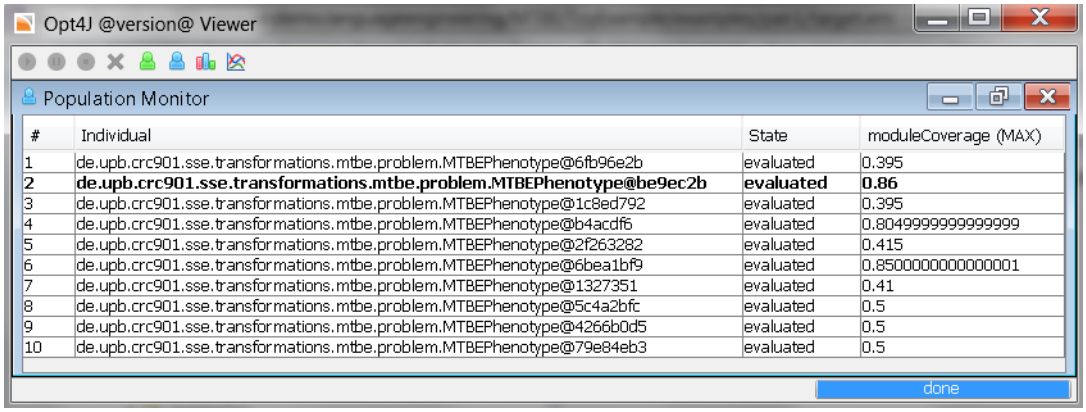
very large solution space of the MTBE optimization problem, a newly generated solution normally has a worse fitness than the mutated solutions. Furthermore, due to the specifics of the mtbe operator `Mutator`, the mutation is based on the information from the `Evaluator` and the language statistics. This fact increases the success of a mutation yielding a higher fitness that makes mutated solutions normally better than randomly generated ones.

In a general case, the choice of a suitable selection strategy also depends on the input languages and the given model mappings. As a reference for `mtbe`, the elitism selection strategy without adding new solutions to the population is recommended. However, if the fitness of obtained model transformations is not satisfying, the strategy could be changed to adding a small subset (e.g., 10% of population size) of newly generated solution to each population.

Thus, the `Selector` preserves a subset of the best solutions for the next population and selects a subset of the solutions for the mutation to fill the remaining part. As a modification, a subset of newly generated solutions is added to the population as well. In any case, the best solution is preserved over the generations till the final evolution run.

In `mtbe`, the `Selector` is also responsible for the selection of the resulting solution (the resulting model transformation) in the final evolution run of the genetic algorithm. The resulting solution has the best possible fitness over all phenotypes created in all runs during the `mtbe` derivation approach. If several solutions have the same best fitness, then one of them is arbitrary chosen.

Example selection Figure 5.23 shows an example population to illustrate the selection process for the running example.



#	Individual	State	moduleCoverage (MAX)
1	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@6fb96e2b	evaluated	0.395
2	**de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@be9ec2b**	**evaluated**	**0.86**
3	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@1c8ed792	evaluated	0.395
4	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@b4acdf6	evaluated	0.8049999999999999
5	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@2f263282	evaluated	0.415
6	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@6bea1bf9	evaluated	0.8500000000000001
7	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@1327351	evaluated	0.41
8	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@5c4a2bfc	evaluated	0.5
9	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@4266b0d5	evaluated	0.5
10	de.upb.crc901.sse.transformations.mtbe.problem.MTBEPhenotype@79e84eb3	evaluated	0.5

Figure 5.23: Solutions in an example Population

This view is the population monitor of Opt4J, which allows to monitor the current population and its development during the learning process. In the population monitor, each solution is represented by its phenotype, which is shown in the column `Individual`. All the presented phenotypes have been

evaluated, which is marked in the column `State` of the population monitor. The calculated coverage of the phenotypes is given in the last column representing the evaluated objective (`moduleCoverage`) that has to be maximized (`MAX`).

The population contains 10 individuals representing 10 solutions, each containing a genotype and an evaluated phenotype. Assume that the number of solutions, which shall remain unchanged in the next population, is set to 2. Thus, 20% of the solutions are added to the next population without a mutation. The remaining 80% are mutated and added to the next population too.

Based on the coverage, the selector identifies two best solutions: the solution `#2` with the coverage `0.86` and the solution `#6` with the coverage `0.85`. These solutions are added unchanged to the next population. The others are given as an input for the mutator, in order to improve their fitness. If the `Selector` needed to output the resulting model transformation after this evolution run, the solution `#2` highlighted bold in Figure 5.23 would be selected.

5.1.7 Mutator

The genetic operator `Mutator` is the final operator in the evolution run of the genetic algorithm of `mtbe` (see Figure 5.2). The `Mutator` modifies the underlying genotypes of the solutions from the current population with the goal to improve their fitness. The mutated genotypes serve as an input for the next run of the genetic algorithm.

In the following, the mutation of model transformations with priority control is described. It leverages the information obtained by `Evaluator` in `PartialMatchReport` (see Listing 5.5) and the maximal coverage of each model pair by each rule (`rule2model2maxCoverage`). Furthermore, the languages statistics extracted from the input model mappings is used (see Section 5.1.3). Depending on the maximal coverage, different kinds of mutations defined below are chosen.

The method starts by initializing two sets: `Set<Rule> coveringRules` storing the rules, which fully cover some model pair, and `Set<ModelPairs> coveredModelPairs` storing the model pairs, which are fully covered by some rule. In order to fill these sets, for each model pair from the set of model pairs `MTBEProblem.modelPairs` (see Listing 5.1) and for each transformation rule from the model transformation `phenotype.module`, the coverage of this model pair by this rule is checked for being 1. In this case, the current rule is added to the set `coveringRules`, and the current model pair is added to the set `coveredModelPairs`. Based on the sets `coveringRules` and `coveredModelPairs`, the kind of mutation is chosen as described below.

As first, if `coveredModelPairs` contains all given model pairs, it is checked, whether the model transformation contains more rules than the amount of covering rules. This case indicates that unused rules exist in the model transformation. The corresponding mutation of the model transformation is to remove the unused rules. For the rules in the given model transformation (`phenotype.module.units`), all those rules are removed from the genotype,

which are not contained in the set `coveringRules`. Since a rule is represented as a list of objects in `MTBEGenotype` (see Listing 5.2), the corresponding lists have to be removed from the `ListGenotype`s of the `CompositeGenotype`.

As second, a model transformation with priority control is checked. In this case, the module coverage (`phenotype.maxModuleCoverage`) can be lower than 1 even if all model pairs are covered. This indicates that rules covering model pairs have lower priority than other rules also applicable to these model pairs. The mutation for this case is to shuffle the rule order in the priority control. In the case of `MTBEGenotype` (see Listing 5.2), the elements of `ListGenotype` representing rules are shuffled so that their order changes but the correspondences between left-hand sides and right-hand sides remain.

In the alternative case, if `coveringRules` does not contain all rules of the model transformation, the module coverage of 1 is not reached by the model transformation yet. As a mutation, corresponding sides of the transformation rules are modified as follows.

Each rule of a model transformation, which does not cover a model pair completely yet, is modified according to a model pair, for which this rule has the best coverage. The idea is to use a model pair for the mutation of a rule, which covers it the best. This allows to faster improve the coverage of the whole model transformation. One model pair is used for mutation once, so that no two rules are optimized for the same model. The goal is to use different model pairs, in order to increase the amount of model pairs considered for the mutation and, thus, improve the coverage of several model pairs in one run.

If the rule covers the source model of a model pair with the coverage lower than 1, then its left-hand side is mutated. Similarly, if the rule covers the target model with the coverage lower than 1, its right-hand side is mutated.

For the mutation of a rule side, the information `PartialMatchInfo` stored in `MTBEPhenotype` (see Listing 5.5) is leveraged to perform goal-oriented mutations instead of completely arbitrary ones. The mutation kind depends on the values for `matchCoverage` (indicates the difference between the reduced rule and the original rule in case of a partial match) and `modelCoverage` (indicates how many nodes of the model are covered by a match). `Mutator` supports removing objects from the rule side based on `delta` (stores the difference between the match and the original rule) and based on the rule application by Henshin. Another kind of mutation is adding or deleting random objects in a genotype.

Remove objects by delta: If match coverage is less than 1, then the rule does not match to the model completely. In this case, both left- and right-hand sides are reduced by objects from the computed delta, the presence of which lead to a partial match. This method leverages information about partial matches of a rule and a model and the difference (delta) leading to a partial match.

For each node from the nodes of delta, the object of the rule side corresponding to this node has to be removed because it does not match with any object in the model and, thus, prohibits a complete match. Before the object is removed, its references are checked for containments, which are also contained in the set of

instantiated references. Then, all references pointing to this objects are removed from the rule side first. Then, the object is removed as well.

As a consequence of removing an object from the rule side, the rule side might lose its conformity to the structure of Ecore models. For example, some objects might now exist in the genotype, which are not contained in any others. This fact contradicts the principle of a tree structure, which models have in Ecore. As a consequence, the rule will not match to the input models. Therefore, all references of the rule side objects, which are contained in the set of instantiated references, are checked and added to the rule side as described in the logic of the `Creator`. This shows the interlocking of the genetic operators `Mutator` and `Creator`, where the `Mutator` uses the method of the `Creator` to create modified genotypes for the next population.

Remove objects by application: If the right-hand side has to be mutated, the results of the rule application to the model are used. Its goal is to check the values of the attributes resulting from the rule application. If the application results in objects having attribute values, which do not occur in the target model, then the corresponding objects have to be removed from the right-hand side of the rule. This mutation helps primarily to remove objects from the rule, which have attributes with incorrect values regarding the given target model.

The method starts with the application of the given input rule to the source model of the given model pair. The application is executed by the corresponding function of Henshin. For each object from the list of model objects resulted from the rule application, the method checks whether the current object exists in the target model of the given model pair. Those objects, which cannot be found in the object of the given target model, are removed from the rule side.

Add objects to model: This method applies in case, if match coverage is complete but model coverage is lower than 1, i.e., a complete match was achieved but it does not cover all objects in the model. As a mutation, new objects are added to the rule, in order to extend the complete match of a rule to the whole model instead of its part and, thus, to achieve the full model coverage.

The method starts with collecting the occurrences of language classes in the corresponding rule and model. If a class from the model does not occur in the rule or the amount of its occurrences in the model is higher than in the rule, then an object of this type is added to the rule side. For the newly added object, the corresponding logic of the `Creator` is used to add the references from and to the existing objects to the rule side (see Section 5.1.3).

Add or delete random objects: In addition to the mutation methods presented above, such mutations like adding or deleting a random object to or from a genotype can be used. The objects can be created by instantiating classes from the map `classes2probabilities` of `MTBEProblem` (see Listing 5.1). The methods to add classes, their attributes and references to a rule side are provided by the logic of the `Creator` (see Section 5.1.3).

Mutation of the example genotype In the following, an example mutation of the genotype shown in Section 5.1.3 is illustrated. This mutation is performed as describe above using the fitness of the example phenotype shown in Section 5.1.5.

The mutation starts by finding model pairs with the coverage 1. Since the rules of the example phenotype cover the given model pairs from Figure 5.6 and Figure 5.7 with lower coverage values, the sets of covered model pairs and covering rules remains empty. Thus, the rule mutation is performed.

An arbitrary model pair is chosen, which has not been covered or used for mutation yet. For the example mutation, the `Model mapping 2` from Figure 5.7 is chosen. For this model pair, the rule with the maximal coverage is selected. Based on the example evaluation in Section 5.1.5, `rule 2` has the higher coverage `0.7` in comparison to `rule 1` with the coverage `0.59`. Thus, `rule 2` illustrated in Figure 5.21 will be mutate. Since its coverage for both source and target models of the chosen model pair is lower than 1 (`0.88` and `0.52` correspondingly), both rule sides have to be modified.

As next, the values for match and model coverages are checked for the corresponding rule side. For the left-hand side of `rule 2` and the source model, the match coverage equals `1` and the model coverage equals `0.75`. These values correspond to the mutation `Add objects to model`. According to this mutation, two maps containing occurrences of classes in the rule side and in the corresponding model are built. For the running example, the first map contains the following entries: $< (UMLSpecification, 1), (Class, 2) >$. The second map contains the entries: $< (UMLSpecification, 1), (Class, 2), (Association, 1) >$. For `UMLSpecification` and `Class`, no changes in the rule side are required because the occurrences are the same in the rule and the model. For `Association`, a new object of this type has to be added to the rule side.

As a result, the second entry of the source part of the example genotype presented in Listing 5.3 is modified as illustrated in Listing 5.6. The source part of the genotype, which were mutated, is highlighted as green.

Listing 5.6: Example mutated genotype for the left-hand side of `rule 2`

```
1 MTBEGenotype:
2   (0, [..]
3   <1:UMLSpecification(classes(2:Class), classes(3:Class),
      associations(4:Association)), 2:Class(name="Clerk"),
      3:Class(name="Manager"), 4:Association(name="hasEmployee",
      dst(2:Class), src(2:Class))>>),
4   (1, [..]
5   <1:RDBSpec(tables(2:Table)), 2:Table(name="Manager",
      tcols(3:Column), pkey(3:Column), fkeys(4:FKey)),
      3:Column(name="MngrId"), 4:FKey(name="hasEmployee",
      cref(3:Column), kcols(3:Column))>>)
6   PriorityControl
```

As a result, an object `4:Association` is added with the attribute `name` and its value `"hasEmployee"` both having the occurrence probability 1. For this

object as a target, a containment `associations` is instantiated for the object `1:UMLSpecification`. The object `4:Association` also has two its associations `dst` and `src` having the occurrence probability 1. These associations are instantiated with the targets at objects of type `Class` chosen randomly. Thus, both added associations point at the same object `2:Class`.

For the right-hand side of `rule 2` and the target model, the match coverage equals to `0.6` and the model coverage equals to `0.43`. Since the value for the match coverage is lower than 1, the mutation `Remove objects by delta` applies. Furthermore, since the right-hand side is considered, the mutation `Remove objects by application` applies as well.

Listing 5.7 illustrates the resulting mutated part of the example genotype.

Listing 5.7: Example mutated genotype for `rule 2`

```
1 MTBEGenotype:
2   (0, [..]
3   <1:UMLSpecification(classes(2:Class), classes(3:Class),
      associations(4:Association)), 2:Class(name="Clerk"),
      3:Class(name="Manager"), 4:Association(name="hasEmployee",
      dst(2:Class), src(2:Class))>>),
4   (1, [..]
5   <1:RDBSpec(tables(2:Table)), 2:Table(name="Manager")>>)
6   PriorityControl
```

As a result of removing objects from the delta from the rule side, the object `4:FKey` including its attribute and references was removed. Furthermore, all references to this object were removed as well, e.g., `fkeys(4:FKey)` of `2:Table`. Since this object did not have any containments, no further references were added to the rule side.

For each object resulted from the application of `rule 2` (see Figure 5.21) to the source model of the `Model mapping 2`, it is checked whether this object can be found in its corresponding target model. For these objects, their data types and their attribute values are checked. For the right-hand side of `rule 2`, two objects of types `FKey` and `Column` are not found in the target model. The reason is the parameters `par_2` and `par_3`, which have no binding to any attribute value from the left-hand side. As a result, the attributes `name` of the objects `:Column` and `:FKey` are not assigned with any value. Thus, the objects `3:Column` and `4:FKey` together with references to them are removed.

As the mutation continues, the remaining `rule 1` is modified based on `Model mapping 1`. This mutation is performed similar as the one above illustrated for `rule 2` and `Model mapping 2`.

Summary To sum up, the genetic operator `Mutator` modifies the model transformations given by the `Selector`. Typically, approaches using genetic algorithms perform an arbitrary mutation realizing elementary changes on the genotype. Such elementary changes imitate the evolutionary process in nature and aim to converge to an optimal solution, if the process runs over a large amount

of generations. In mtbe, the `Mutator` is designed as a heavy-weighted operator realizing larger changes with the goal to foster the convergence of the genetic algorithm. For that, the `Mutator` leverages the information about the objects hampering a complete match of a rule to a model, about the objects producing incorrect attribute values, and about the model objects missing in the rule. Using this information, model transformations with a high coverage are obtained faster by the mtbe approach.

5.1.8 Quality of Generated Model Transformations

This section discusses the quality of the Henshin model transformations, which the mtbe approach outputs as a result. This section starts with properties concerning the execution result of a model transformation.

The first considered quality property is the syntactical correctness of the resulting model [144]. This property implies that the resulting model has to conform to the metamodel of the target language. The mtbe approach generates syntactically correct model transformations by design.

The next considered property is the semantic correctness of the resulting model to the corresponding source model [144]. In mtbe, the semantic correctness results from the semantic correctness of the model transformation. The semantic correctness of the model transformation can be evaluated using, for example, Dynamic Meta Model (DMM) [132]. Under the assumption that both source and target languages have a formal semantics definition using DMM, different quality properties can be checked on both source and target models evaluating their semantic correctness.

In order to improve the quality of a Henshin model transformation, approaches [23], [145], and [149] exist. Born et al. [23] propose to use critical pair analysis to identify potential conflicts and dependencies of rule applications in a transformation. Taentzer et al. [145] propose to perform different refactoring strategies, in order to improve such quality properties of a model transformation, like conciseness, changeability, and comprehensibility. Tichy et al. [149] presents the list of bad smells leading to performance problems of Henshin model transformations.

Taentzer [144] discusses further quality properties to verify model transformations. Termination of a model transformation is an important property, if loops are used in the control flow. Since the current version of mtbe does not generate transformations with loops, the check of this property is considered as future work. Furthermore, algebraic graph transformations provide means to verify the confluence of model transformations. Since the current version of mtbe generates transformations, which apply one rule only (independent and priority control), the property of confluence is also considered as future work.

Biermann [21] introduces the definition of consistent transformation rules, which allow to perform the analysis of critical pairs and confluence. A consistent transformation rule allows the deletion or creation of nodes only together with the deletion or creation of their containment edges. Another property of

a consistent transformation rule is that the deletion or creation of a containment edge follows by the deletion or creation of a content node or, if a content node already exists, then a containment edge for this node has to be created. Furthermore, no parallel edges between two nodes are allowed as well as cycle-capable containment edges have to be handled. The genetic operator `Creator` designed in mtbe creates genotypes, which are decoded exactly in consistent model transformations. Thus, the analysis of Biermann can be performed on resulting model transformations in Henshin.

5.2 Evaluation

This section presents the evaluation of the mtbe approach. Section 5.2.1 explains the tool support realizing mtbe. Section 5.2.2 continues with the application of mtbe to the service specification languages OWL-S and SAWSDL, for which the obtained model transformations are used for the evaluation. Section 5.2.3 compares the solution approach against the requirements stated in Section 3.1.2.

5.2.1 Tool Support

The tool support realizing the mtbe approach is developed as a part of the component `SSE` (**S**ervice **S**pecification **E**nvironment) [152] of the tool suite `SeSAME` (**Se**rvice **S**pecification, **A**nalysis and **M**atching **E**nvironment) presented in Section 4.3.1. The architecture of `SSE` is shown in Figure 5.24.

`SSE` realizes the concept of the comprehensive core language introduced in Section 4.2.2. The realization and editors of the comprehensive core language are already explained in Section 4.3.1. Besides the comprehensive core language, `SSE` realizes such language operations as view building and coverage calculation used in Section 5.1.2 in the description of the running example. The main focus for the evaluation in this section lies on the tool support for mtbe. mtbe uses two frameworks: `Opt4J` realizing different generic algorithms and `Henshin` realizing graph-based model transformations.

As shown in Figure 5.24, in addition to the mtbe approach, semantic-preserving hand-written model transformations are implemented. These model transformations are written from SAWSDL [45], OWL-S [91] and UML [108] to the comprehensive core language. These model transformations are implemented by experts, who learned the abstract syntax of the listed languages and their informal and formal semantics. The model transformations can be applied to single service specifications as well as to test collections.

In order to run the mtbe approach, its different parameters can be initialized to steer the learning process. Figure 5.25 shows the dialog to set such parameters. For starting the learning algorithm of `mtbe`, the concrete or maximum amount of rules in the resulting model transformation can be set. Furthermore, the concrete sizes of the left- and right-hand sides of rules in this model transformation

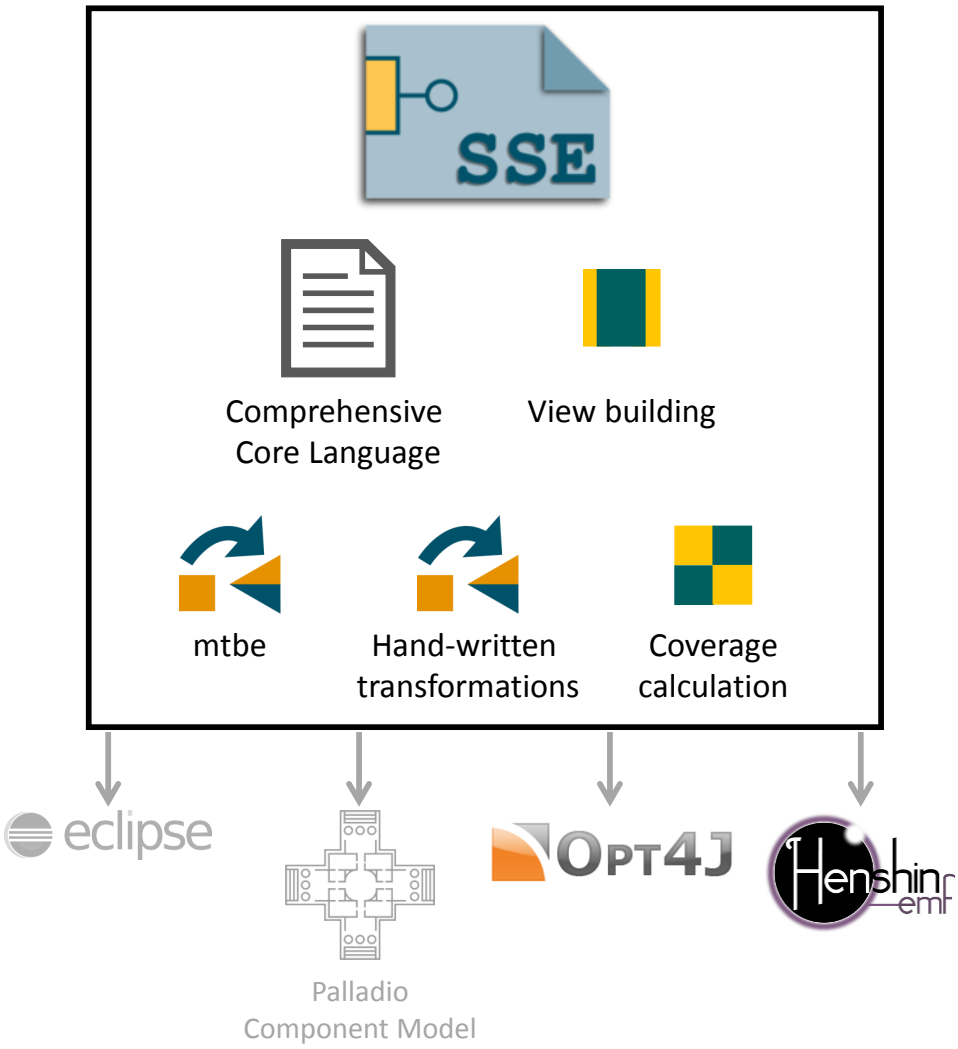


Figure 5.24: Architecture of the tool support for mtbe

can be given. The setting of the concrete amount of rules or the concrete rule sizes is the means to determine the form of the resulting model transformation and, thus, to restrict the search space of the mtbe learning algorithm.

As further parameters, the control flow of the resulting model transformation can be set (e.g., `Priority Unit`). The size of the population is set for genetic runs and the amount of offsprings are set for aligning the selection of phenotypes for the next genetic run (amount of offsprings determines the amount of genotypes to be mutated). The number of iterations determines the overall amount of runs of the genetic algorithm of mtbe. The objective, according to which solutions in a current population are evaluated, can be chosen as well.

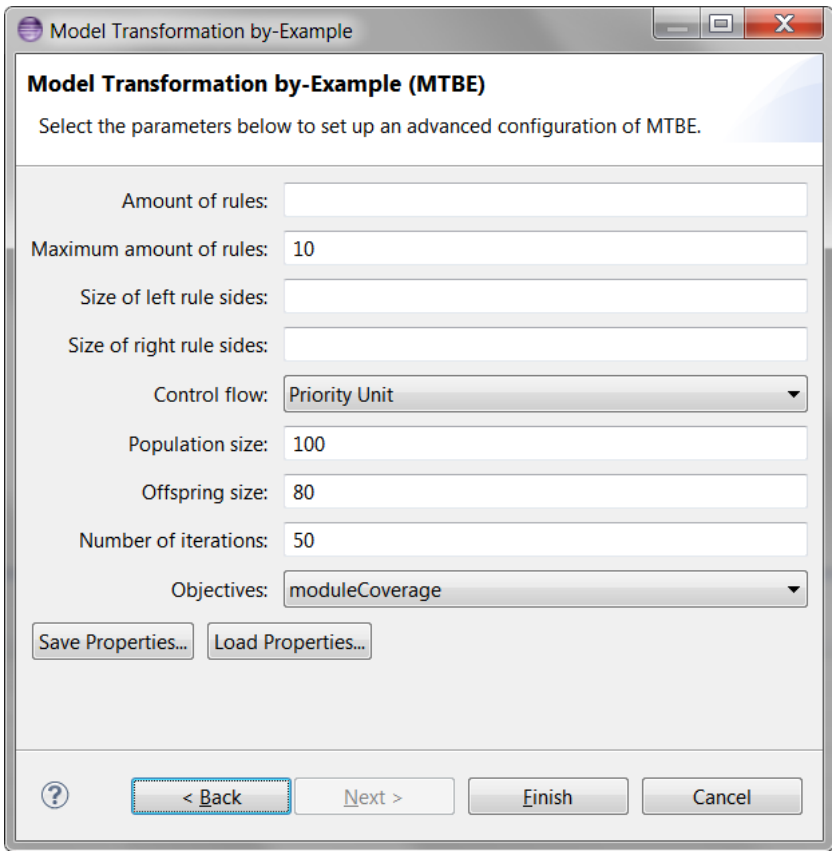


Figure 5.25: Dialog for setting mtbe parameters

5.2.2 Evaluation on a Case Study

The evaluation of mtbe is performed using a test collection of service specifications written in SAWSDL and the operation of service matching. The S3 Contest on Semantic Service Selection [123] provides this test collection.

The goal of the evaluation is to show that the matching effectiveness computed on specifications transformed in the comprehensive core language using the mtbe approach is comparable to the matching effectiveness computed on specifications obtained using the semantic-preserving hand-written transformations.

For this purpose, the test collection including pairs of specifications and their expected matching results are used to compute the matching effectiveness as introduced in Definition 15. The matching effectiveness is evaluated for specifications from the test collection transformed into the comprehensive core language using the hand-written transformations and mtbe. Since the specifications from the same test collection are used and their expected matching results are known in advance, the matching effectiveness can be objectively compared.

This section starts by introducing the evaluation procedure in Section 5.2.2.1. Section 5.2.2.2 presents the evaluation of the matching effectiveness for specifications of the test collection obtained using hand-written transformations.

Section 5.2.2.3 shows the evaluation of the matching effectiveness of the specification from the same test collection obtained using mtbe. Section 5.2.2.4 concludes the evaluation by comparing the values for the matching effectiveness presented in Sections 5.2.2.2 and 5.2.2.3.

5.2.2.1 Evaluation Procedure

Figure 5.26 shows an overview of the evaluation procedure for mtbe.

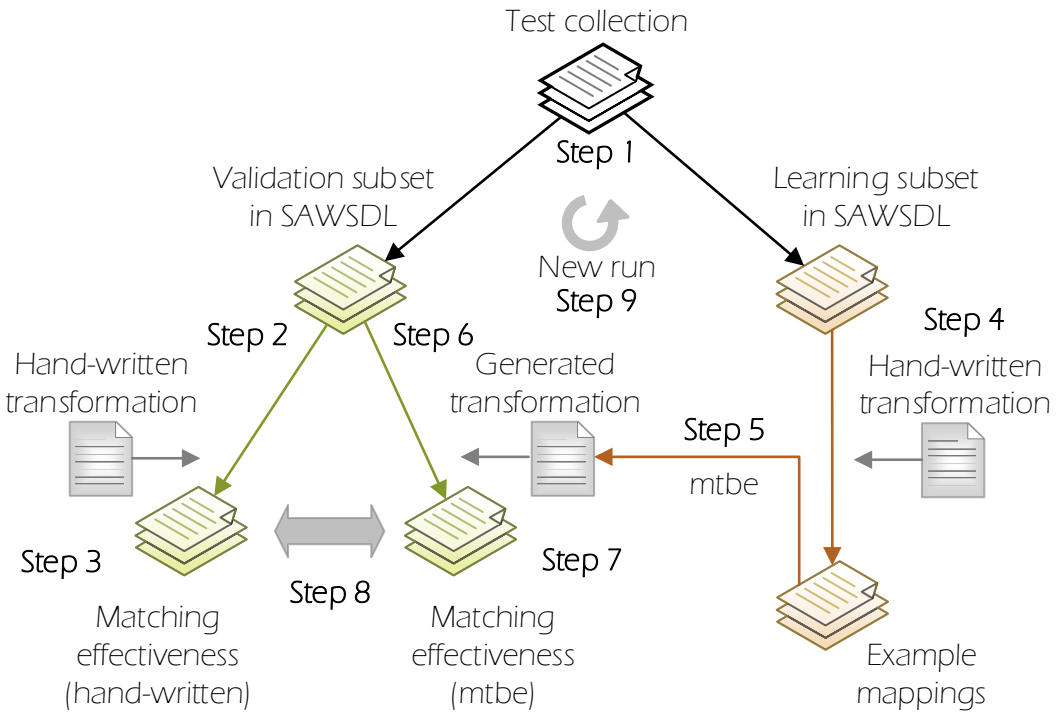


Figure 5.26: Overview of the evaluation procedure

The evaluation procedure for the mtbe approach contains the following steps:

Step 1 Two subsets are built from specification pairs of the test collection: a subset of specification pairs for validation and a subset of specification pairs for learning. Both subsets contain pairs of requirements and service specifications written in SAWSDL.

Step 2 The specifications of the validation subset are translated into the comprehensive core language using semantic-preserving hand-written model transformations from SAWSDL into the comprehensive core language implemented in SSE. As service properties in the comprehensive core language, operation signatures and data types are considered, because these are covered by SAWSDL.

Step 3 Using the service matching implemented in `MatchBox` introduced in Section 4.3.1, the matching effectiveness for the transformed specification pairs is evaluated. The matching effectiveness is evaluated according to Definition 15 presented in Section 4.1.2. For the evaluation procedure, expected matching results for these specification pairs given in the test collection are considered.

Step 4 A model transformation from SAWSDL and the comprehensive core language is generated using the mtbe approach based on example mappings. To obtain example mappings, the learning subset of specification pairs from the test collection is translated into the comprehensive core language using the hand-written model transformation. The created mappings contain semantically equal specifications in SAWSDL and the comprehensive core language, thus, serving as example mappings.

Step 5 Using the example mappings obtained in **Step 4** as an input, the learning algorithm of mtbe generates a model transformation. Since the set of specifications used for learning and validation are disjunctive, mtbe learns on different specifications than the resulting model transformation applies to. This fact increases the objectiveness of the evaluation.

Step 6 Using the generated model transformation, the specifications from the validation subset obtained in **Step 1** is transformed into the comprehensive core language.

Step 7 Using the service matching of `MatchBox` and the expected matching results from the test collection, matching effectiveness is evaluated for the specification pairs transformed using the generated model transformation. Expected matching results for these specification pairs are given in the test collection.

Step 8 The values for the matching effectiveness obtained in **Step 3** and **Step 7** are compared. A conclusion, whether the different of the values is acceptable, is drawn based on the expert knowledge in matching from the CRC 901.

Step 9 Steps 1 to 8 are repeated for different subsets of specification pairs of the test collection. Based on several evaluation runs, the final conclusion for the evaluation is drawn.

During the evaluation, different settings was tried out, which vary in the size of learning subset and the settings for mtbe, e.g., population size, selection strategy, or number of evolution runs. These parameters were changed, in order to investigate their influence on the evaluation results.

For each evaluation setting, several numbers of runs were performed on different subsets of specifications from the test collection. Based on these results,

the best parameter settings were identified experimentally. For these settings, the evaluation for two selection strategies (elitism and non-elitism) are shown in the next chapter.

5.2.2.2 Matching Effectiveness of Specifications Obtained using Hand-Written Transformations

This section describes, how **Steps 1–3** of the evaluation procedure from Figure 5.26 were performed.

For each evaluation setting, validation subsets of size 100 pairs were built randomly from the specification pairs of the SAWSDL test collection. Specifications from these subsets were transformed into the comprehensive core language using a semantics-preserving model transformation implemented manually in `SSE` using Java. Since specifications in SAWSDL cover operation signatures of services and data types used in them, only those service properties were considered in the comprehensive core language as well. For each of validation subsets, matching effectiveness according to Definition 15 was calculated.

Figure 5.27 shows an example pair of specifications used for validation. Each interface contains one operation `get_PUBLICATION` having one input and one output parameter.

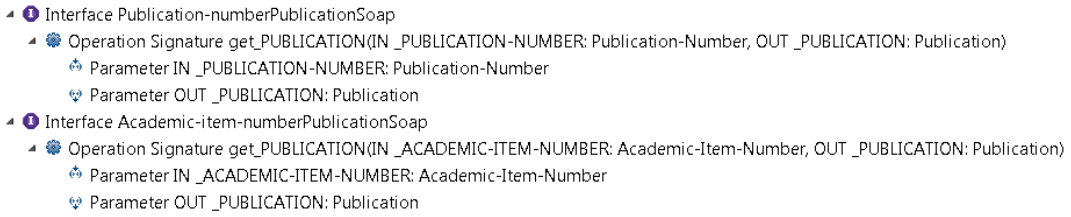


Figure 5.27: Example pair of specification used for validation

Service matching used in `MatchBox` is ontological signature matching. Its setting considered signature names, input and output parameters as well as their names and data types. Matching results obtained for each of those properties were integrated into a signature matching result, which got a value from the interval 0 to 1. For calculating the matching precision and recall (see Definitions 13 and 14), this matching result was compared with the expected matching results given as a number 0 or 1 (binary matching result in `MatchBox`).

For the example pair in Figure 5.27, Figure 5.28 shows the expected `Signature Matching Results: 0%` and the calculated `Continuous Matching Result` for `Operation Signature: 0%`. The calculated result is an aggregation of matching results for the operation name (100%), input parameter types (0%), output parameter types (100%), input parameter names (0%), and output parameter names (100%). These results are obtained by either comparing the string similarity (`Cosine Similarity`) or the ontological similarity (`Ontology Class`).

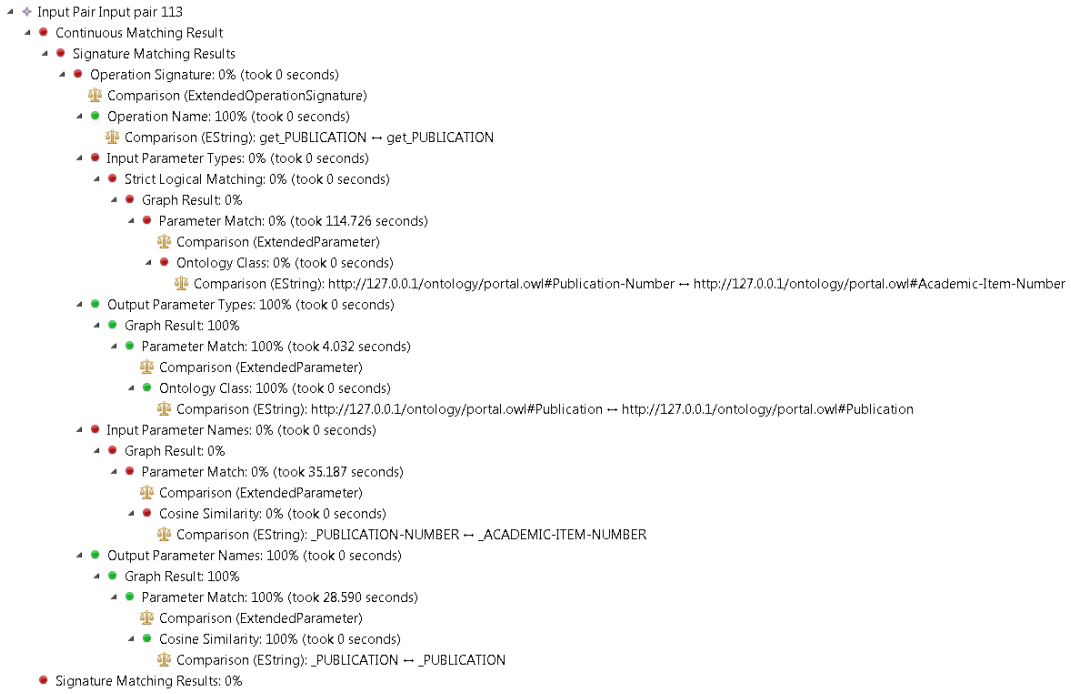


Figure 5.28: Matching results for the example pair

Figure 5.29 and Figure 5.30 show values for the matching effectiveness normalized on the interval from 0 to 1. These figures show 10 evaluation runs each. The matching effectiveness was calculated for different validation set of 100 specifications evaluated per run. The runs were executed for two selection strategies: an elitism and a non-elitism (see Section 5.1.6).

Figure 5.29 illustrates that the matching effectiveness in the case of the elitism strategy, in which no new individuals are added to the populations and the evolution is based on mutating the initial ones. The percentage of mutated individuals was 75% of the whole population, while the remained ones filled 25%. The values for the matching effectiveness vary from `0.41` to `0.68`; the average matching effectiveness equals to `0.54`. The difference in the matching effectiveness depends on the distribution of (true and false) positives and negatives in each validation set.

Figure 5.30 illustrates the case of the non-elitism strategy, according to which new individuals are added to the population in each generation. In the presented case study, the mutated individuals filled 50% of the population, while the sets of remained and new individuals filled 25% correspondingly. The values for the matching effectiveness vary from `0.4` to `0.62`; the average matching effectiveness equals to `0.53`.

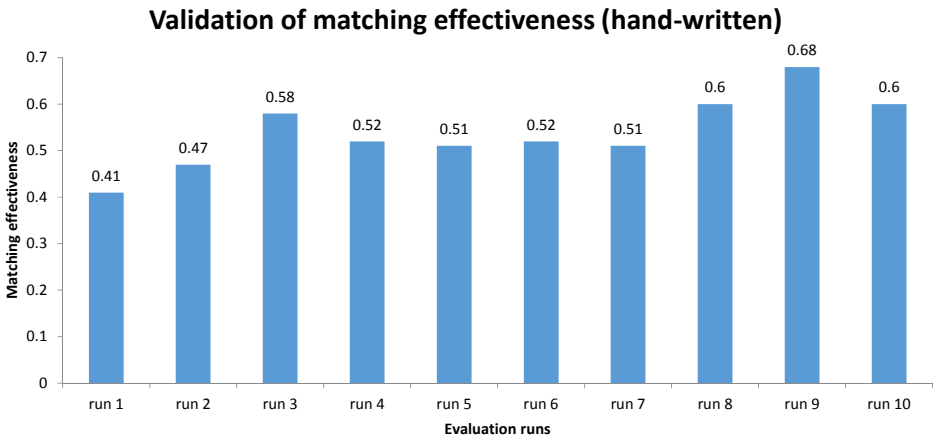


Figure 5.29: Matching effectiveness for the elitism strategy

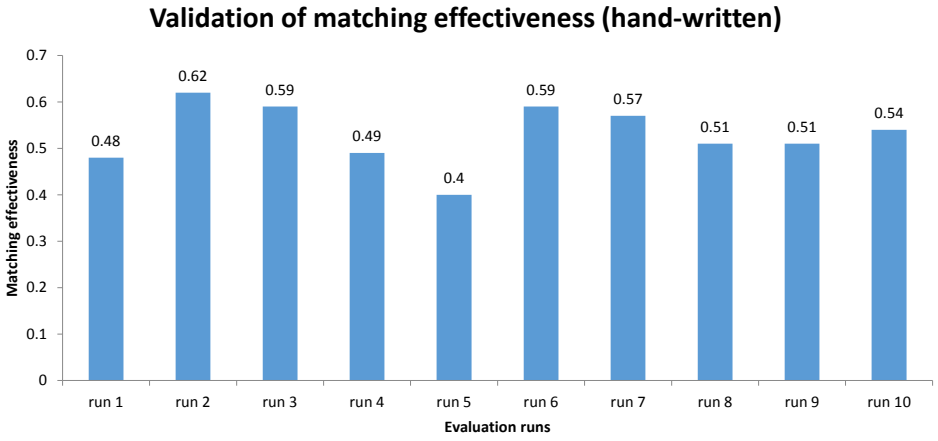


Figure 5.30: Matching effectiveness for the non-elitism strategy

5.2.2.3 Matching Effectiveness of Specifications Obtained using mtbe

This section describes how **Steps 4–7** of the evaluation procedure from Figure 5.26 were performed.

In **Step 4**, the predefined number of specifications written in SAWSDL were randomly selected from the test collection for obtaining specifications for the learning set. Specifications from the learning set were transformed into the comprehensive core language using the hand-written model transformation applied in **Step 2**. This evaluation was performed on a learning set of 20 pairs. During the experiments, starting with this size of the learning set, no positive influence of a higher size of the set was determined. In the case of the chosen test collection, it can be explained by the following two facts. Firstly, the models have a rather common structure. Secondly, the more possible values are there

for a certain parameter to choose for the generation of a genotype, the lower the probability of a match between the left and right side of the rules becomes.

Figure 5.31 and Figure 5.32 show an example mapping between two semantically equivalent specifications in SAWSDL and the comprehensive core language.

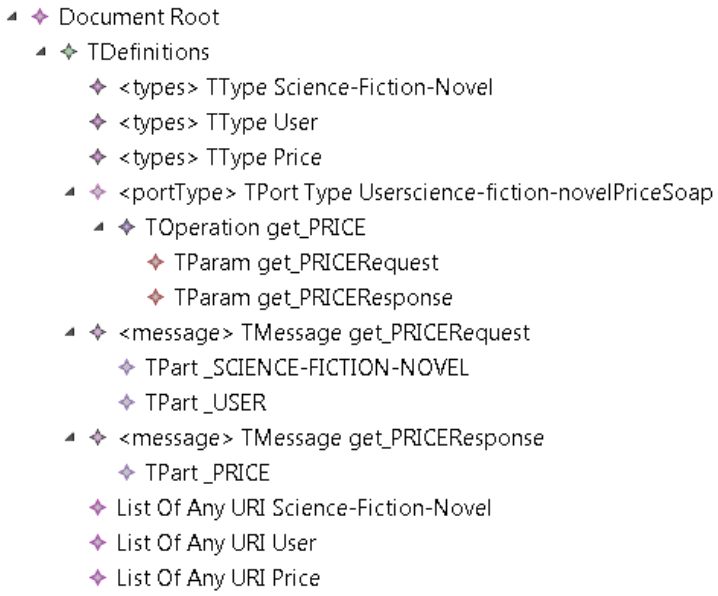


Figure 5.31: Specification in SAWSDL from the example mapping

The specifications describe a service `Userscience-fiction-novelPriceSoap` with an interface having an operation `get_PRICE`. This operation contains two input parameters (`_SCIENCE_FICTION_NOVEL` and `_USER`) and an output parameter (`_PRICE`), which are represented in SAWSDL by the language construct `TPart` contained in `TMessage`. Ontological data types referenced by the parameters are modelled as URIs, e.g., `http://127.0.0.1/ontology/books.owl#User`. Language constructs `TType` and `List Of Any URI` represent that in SAWSDL.

In **Step 5**, `mtbe` generated 20 model transformations, one per evaluation run.

Figure 5.33 and Figure 5.34 show the fitness of the generated model transformations for both elitism and non-elitism strategies. In the course of evaluation, a slightly better fitness was reached with the non-elitism strategy. The average

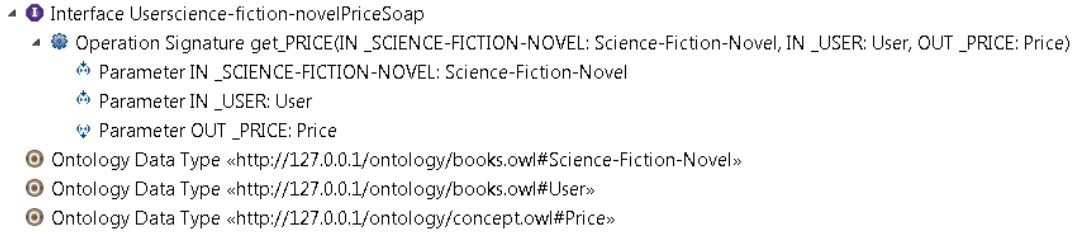


Figure 5.32: Specification in the core language from the example mapping

fitness in case of the elitism strategy equals to 0.89, while the average fitness in the case of the non-elitism strategy is 0.915. The minimum obtained fitness with the elitism strategy was 0.86, while the non-elitism strategy gained 0.89. For the maximum obtained fitness, the value of 0.96 was reached with the non-elitism strategy, while 0.94 was reached using the elitism strategy.

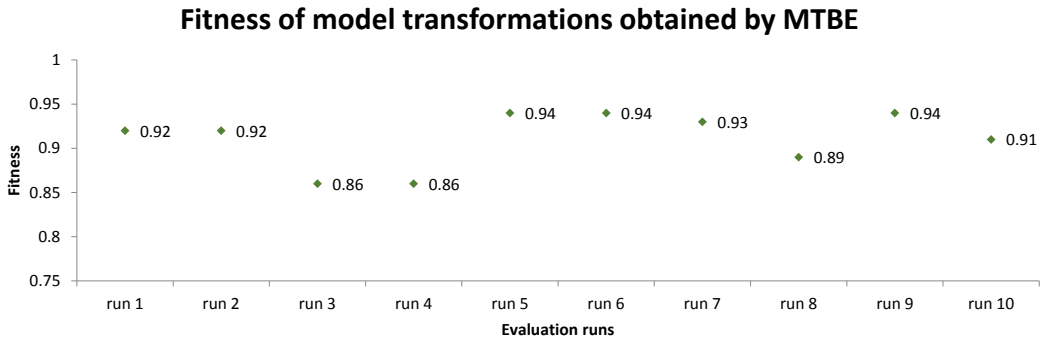


Figure 5.33: Fitness of the model transformations (elitism strategy)

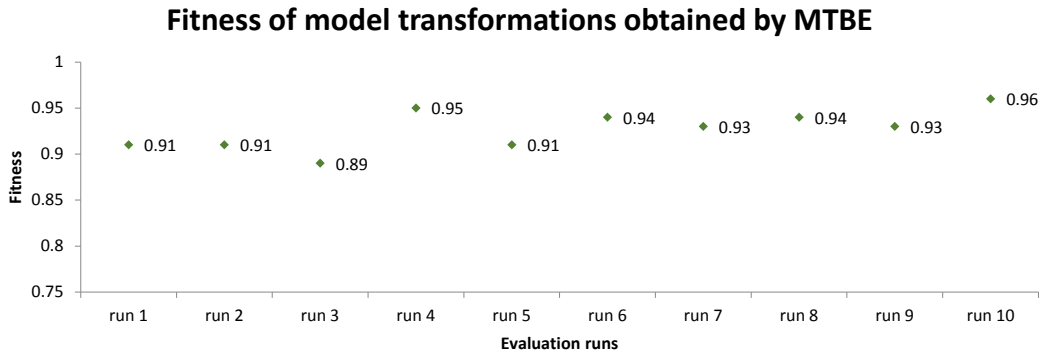


Figure 5.34: Fitness of the model transformations (non-elitism strategy)

In **Step 6**, specifications in SAWSDL from the validation subsets are transformed into the comprehensive core language using the corresponding generated model transformation. A generated model transformation can preserve the following semantic properties in service and requirements specifications: the operation name, the name of one or several operation parameters, the names of the data types for one or several operation parameters, the ontological data types of one or several operation parameters.

In **Step 7**, the matching effectiveness is calculated for the validation set transformed using the generated model transformation. Values for the matching effectiveness are presented in Figure 5.35 for the elitism strategy and in Figure 5.36 for the non-elitism strategy. The values vary between 0.33 and

0.53 in the case of the elitism strategy and between 0.27 and 0.47 for the non-elitism strategy. The average fitness in case of the elitism strategy equals to 0.89, while the average fitness in the case of the non-elitism strategy is 0.915.

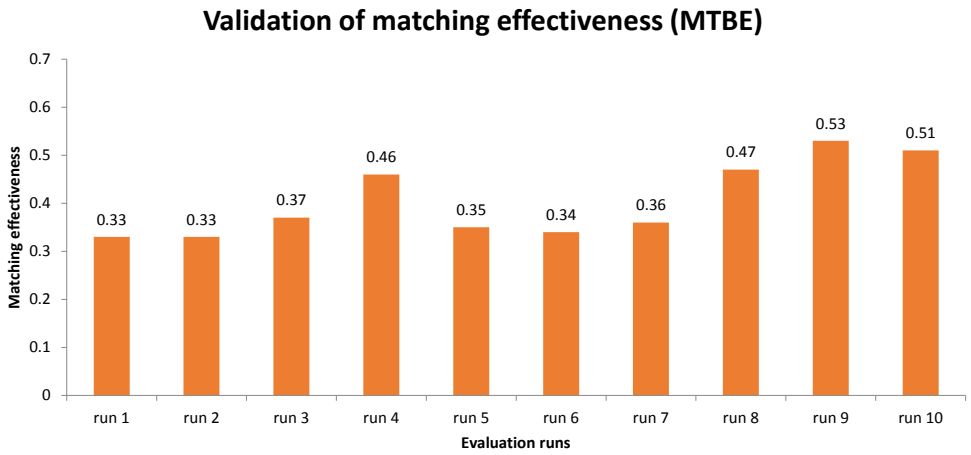


Figure 5.35: Matching effectiveness for the elitism strategy

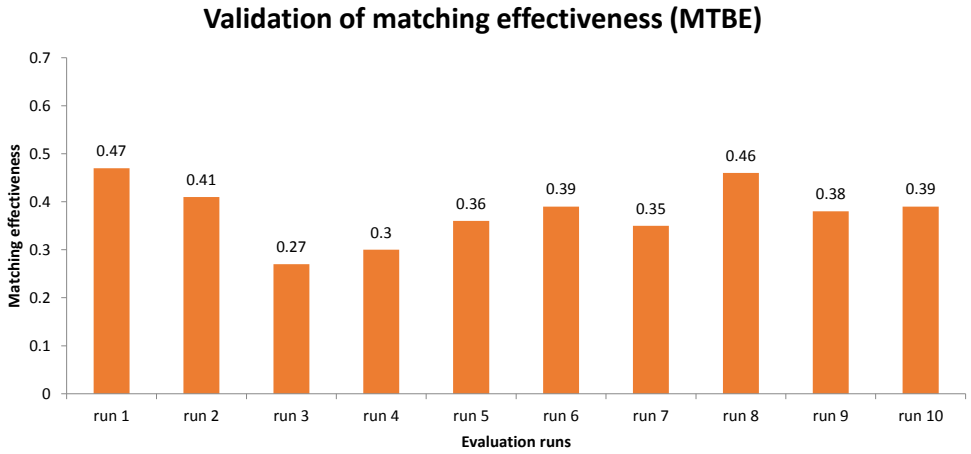


Figure 5.36: Matching effectiveness for the non-elitism strategy

5.2.2.4 Comparison of Matching Effectiveness

This section presents the comparison of the values for matching effectiveness obtained using hand-written model transformations (Figure 5.29, Figure 5.30) and model transformations generated with mtbe (Figure 5.35, Figure 5.36). This evaluation was performed for the elitism and non-elitism selection strategies.

In comparison to the matching effectiveness obtained using the hand-written model transformation, the average effectiveness obtained using the generated

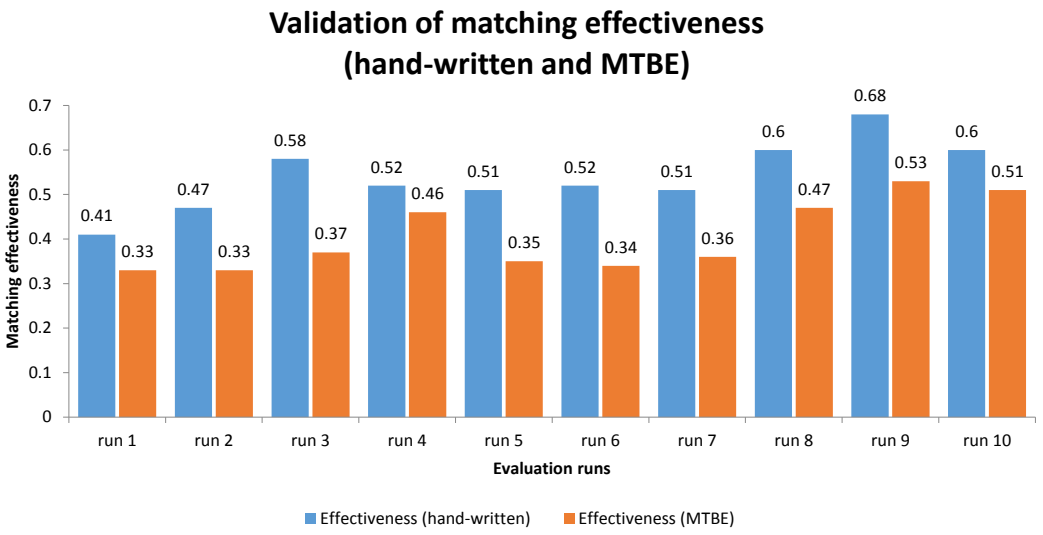


Figure 5.37: Matching effectiveness for the elitism strategy

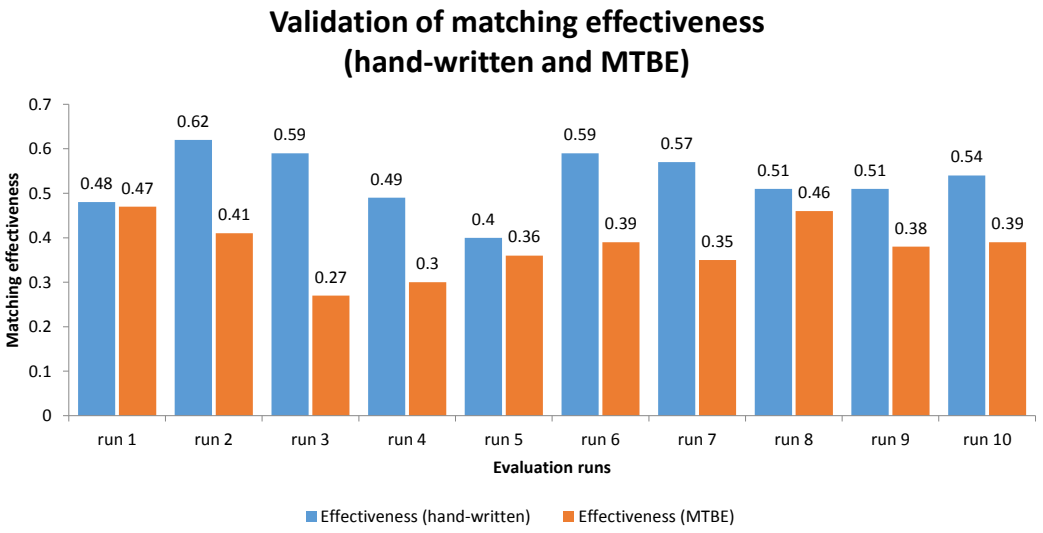


Figure 5.38: Matching effectiveness for the non-elitism strategy

model transformations is lower for both strategies. The difference in the average matching effectiveness is `0.13` for the elitism strategy and `0.15` for the non-elitism strategy. The maximum delta in the case of the elitism strategy is `0.21` and `0.32` in the case of the non-elitism strategy.

Thereby, as described in Section 5.2.2.3, the average fitness of the model transformations generated with the elitism strategy was lower, however the delta was better. The better delta in the matching effectiveness can be explained by the fact that the service semantics preserved by these model transformations results

in a better matching effectiveness. Furthermore, the distribution of positives and negatives in the validation sets can also influence the delta.

To sum up, the evaluation achieves its goal and shows that the matching effectiveness computed on specifications transformed using the mtbe approach is comparable to the matching effectiveness computed on specifications obtained using the semantic-preserving hand-written transformations.

5.2.3 Evaluation of the Requirements

This section explains how the requirements stated in Section 3.1.2 are fulfilled by the presented solution approach mtbe.

R.2.1.1 – mtbe applies to languages having a formal definition of the abstract syntax in the form of a metamodel described in Ecore as well (see Section 5.1.1). mtbe support a transformation into the comprehensive core language as its abstract syntax is defined in Ecore (see Section 4.2.2). Since Ecore is a broadly-used language for defining the abstract syntax of specification languages in OTF markets, mtbe is broadly applicable in these markets as well.

R.2.1.2 – mtbe can generate model transformations described in different transformation languages using different realizations of the genetic operator `Decoder` (see Section 5.1.4). These model transformations are directly executable on specifications of the source languages as they are defined on the formal definition of their abstract syntax. The evaluation presented in Section 5.2.2 shows that the matching effectiveness obtained with the generated model transformation is sufficient. Thus, no additional manual refinement of generated model transformations is required.

R.2.1.3 – mtbe can generate model transformations for whole languages as well as only for their parts used in the example mappings (explained in Section 5.1.2). Using the coverage calculation of the languages by models from the example mappings, how good the relevant parts of the language are covered by the example mappings can be evaluated. Thus, mtbe provides to the market actors the means to cope with the complexity of their specification languages.

R.2.1.4 – mtbe generates model transformations with arbitrary correspondences between language constructs of the source and target languages (i.e., 1-to-1, 1-to-N, N-to-1, N-to-N). This is achieved by the genetic operator `Creator`, which arbitrary determines the sizes and language constructs for the rule sides in a model transformation (see Section 5.1.3). As a result, the correspondences between the left- and the right-hand side of transformation rules are used to map the language constructs of the source and target languages.

R.2.1.5 – mtbe generates model transformations that lead to acceptable matching results of specifications transformed in the optimal core language using this transformation. The evaluation in Section 5.2.2 shows that the matching effectiveness calculated for specifications transformed using the generated model transformation is sufficient for market actors in the service markets.

R.2.2.1 – mtbe uses the technique of Model Transformation By-Example (see Section 2.2.2.2) to generate model transformations. As depicted in Figure 5.1, mtbe takes as input a set of model mappings consisting of pairs of semantically equivalent example models. Market actors create these pairs of specifications using the concrete syntax of the chosen source and target languages. Thus, such mappings are created on the example level and do not require an expertise in language design from market actors.

R.2.2.2 – mtbe works on formal language definitions of the source and target languages as described in Section 5.1.1. The modeling language Ecore is used to formally specify the abstract syntax in the form of metamodels. As explained in *R.2.2.5*, depending on the choice of the model transformation language, the resulting model transformations are directly executable on models of the source language.

R.2.2.3 – mtbe provides means to evaluate the quality of the given example models. As described in Section 5.1.2, the coverage of the source or target language by the corresponding models from the input mappings can be calculated. Thus, market actors have the possibility to control and improve the quality of the created models.

R.2.2.4 – mtbe provides means to create a model transformation not only for the whole source and target languages but also for their parts used in the model mappings (see Section 5.1.2). The presented coverage calculation can be applied to the language parts and enables to evaluate the quality of the models with respect to the chosen language parts.

R.2.2.5 – mtbe leverages the knowledge from mappings between example models. The genetic operator `Creator` collects statistics about the source and target languages based on the input model mappings. This statistics contains language constructs used in the model mappings as well as their occurrence probabilities. Using the knowledge from this statistics, the `Creator` generates random genotypes (see Section 5.1.3). The genetic operator `Decoder` generates model transformations based on genotypes (see Section 5.1.4). Thus, the knowledge from the model mappings encoded in the genotype are propagated. The genetic operator `Evaluator` uses the input model mappings to evaluate the fitness of the obtained model transformations (see Section 5.1.5). Based on the calculated fitness,

the genetic operator `Mutator` adapts the genotypes using the information from the model mappings (see Section 5.1.7). Thus, the knowledge from the model mappings is also used to evolve the model transformations.

R.2.2.6 – mtbe uses the technique of genetic algorithms, i.e., a metaheuristic, to derive model transformations. The genetic operators of mtbe are designed to allow a faster convergence of the genetic algorithm as explained in Section 5.1.1. As a result, the presented mtbe approach is able to cope with the large search space of model transformations possible as solutions for the given source and target languages.

R.2.2.7 – mtbe generates model transformations with arbitrary correspondences between language constructs in their rules and the control flow of arbitrary complexity. The genetic operator `Creator` is used to generate left-hand sides and right-hand sides of rules containing arbitrary chosen language constructs from the given source and target languages. Thus, the correspondences are determined on the level of rule sides.

R.2.2.8 – mtbe is independent from any concrete model transformation language as model transformations are built based on genotypes having a language-independent representation. The corresponding implementation of the `Decoder` used by mtbe allows to choose the model transformation language, in which resulting model transformations are described.

R.2.2.9 – mtbe defines coverage of models from example mappings by the resulting model transformation as the fitness. Furthermore, the matching effectiveness is computed for example models, in order to evaluate the reliability of matching results obtained using the generated model transformations. If the sufficient matching effectiveness can be achieved, then the generated model transformations have a sufficient fitness.

6 Conclusion and Future work

This chapter presents the conclusions drawn for this PhD thesis and an outlook of the future work. Section 6.1 describes the main contributions of this PhD thesis. Section 6.2 introduces further topics of research, which can be tackled as a future work based on this PhD thesis.

6.1 Conclusion

This PhD thesis is written in the scope of the Collaborative Research Centre 901, whose researchers develop concepts and techniques for a new software development paradigm called On-The-Fly (OTF) Computing. OTF Computing is based on the idea of specialized service markets called OTF markets. These specialized OTF markets have different properties and their market actors use different modeling techniques to perform service engineering in these markets. This PhD thesis proposes a solution to address the problem of heterogeneity in OTF markets with the goal to foster the success of OTF Computing.

The first main contribution is the approach Language OPTimizer (LOPT) presented in Chapter 4. This approach provides a systematic process for the design of a service specification language optimal for efficient automated market operations in an OTF market (see Section 1.2.1 for problem statement). This PhD thesis considers six market operations, e.g., `Specify a service`, which market actors perform in an OTF market. Since market actors use different specification languages to specify their services, other operations, e.g., `Match a service`, would have to be defined for all existing specification languages.

Using the approach LOPT, a core language can be obtained automatically for each OTF market after the properties of this market have been formalized manually. A core language serves as an intermediate representation for service specifications of market actors. The automated market operations are defined for the core language only that spares the effort to define these operations for all existing specification languages as well as increases the quality of the operation definition. Using LOPT, the core language is designed in a way that it optimally supports the execution of the market operations in a considered OTF market.

For designing an optimal core language, the approach LOPT introduces an extended language definition. The traditional definition is extended by the language pragmatics defined in the form of operations performed on specifications of a language. Firstly, operations being a part of the language pragmatics have to be defined for the language. Secondly, a measurable definition of the quality

of the execution of each operation has to be given. The optimality of a language can be evaluated by measuring the quality of the execution of these operations. For OTF markets, the pragmatics of core languages is defined by the operation of service matching performed on specifications written in the core language (see Section 4.1.1). The notion of the core language optimality is defined with respect to the trade-off between the efficiency and effectiveness of the service matching (see Section 4.1.2).

In order to obtain a core language optimal according to the presented optimality definition, the approach LOpt uses a comprehensive core language and a configuration procedure for it (see Section 4.2). The comprehensive core language integrates existing specification languages and covers various structural, behavioral, and non-functional service properties. The reuse of existing specification languages increases the acceptance of the comprehensive core language by the market actors. As a result, the comprehensive core language serves as a taxonomy of service properties for OTF markets and allows to create formal comprehensive service specifications.

Since the core language optimality is defined regarding the operation of service matching, using the comprehensive core language for the execution of market operations yields reliable matching results. The reason are different service properties, whose consideration increases the reliability of the matching results. However, the efficiency of the service matching for the comprehensive core language is low, because comparing all these properties takes more computation time. Therefore, LOpt customizes the comprehensive core language, in order to optimally support the service matching in the given OTF market. The customization is performed with respect to the trade-off between the efficiency and effectiveness of the service matching as described in the optimality definition.

The configuration approach of LOpt performs the customization based on the characteristics of a service market formalized by market actors in the form of market properties (see Section 4.2.3). An example property is the level of standardization in an OTF market. The formal market properties allow to describe characteristics of service markets in a thorough standardized way. This enables the automated processing of the market properties as well as the reuse of the optimal core language created for service markets with the same properties.

Based on market properties, the configuration approach automatically creates an optimal core language using a knowledge base containing the expertise for the configuration. This configuration expertise is formalized in the form of configuration rules. Configuration rules define the kind of customization of the comprehensive core language depending on a value of a market property. The configuration is performed on both coarse- and fine-grained levels, i.e., the whole service property can be omitted from the comprehensive core language or certain language constructs describing a service property in detail can be omitted while the service property remains in the optimal core language. This principle improves efficiency of the configuration.

The approach LOpt was successfully evaluated for several service markets, for

which optimal core languages were obtained with LOpt (see Section 4.3). The evaluation was performed using the tool suite SeSAME containing the component `LM configurator` realizing the concepts of LOpt. Section 4.3.3 presents how the approach LOpt satisfies the requirements stated in Section 3.1.1.

The second main contribution of this solution is the approach Model Transformation By-Example (mtbe). This approach allows the market actors to define language transformations from their proprietary specification languages to the optimal core language in a user-friendly manner. With its help, market actors can enter an OTF market without the need to rewrite their existing specifications. Using the obtained transformations, market actors can transform their existing service specifications into specifications in the optimal core language, which the automated market operations can be executed on. As a result, mtbe enables the market actors to enter the OTF market with a few effort and to customize their existing specifications so that the automated market operations are executed optimally. This fosters the success of market actors in the OTF market and, thus, the overall success of OTF Computing.

The approach mtbe aims at creating language transformations in a user-friendly manner. The approach mtbe automatically generates a model transformation for a proprietary language of a market actor and the optimal core language of the considered OTF market. For that, a market actor creates example mappings between concrete specifications in her language and their semantically corresponding specifications in the optimal core language. The approach mtbe derives a directly executable transformation between the languages preserving their semantics and based on the specified example mappings. The derivation approach applies the idea of genetic algorithms having rich genetic operators, which allow the effective and efficient exploration of the solution space of possible transformations.

The approach mtbe starts by introducing a method to create model mappings of a high quality, based on which a transformation of a high quality can be derived. Mappings of high quality are obtained by improving the coverage of the source and target languages by given example models (see Section 5.1.2). By increasing the quality of the model mappings, the quality of the model transformations built based on these mappings increases as well.

The approach mtbe applies the idea of genetic algorithms to realize the user-friendly technique of Model Transformation By-Example (see Figure 5.2). The genetic operators defined for the mtbe approach are less-random than in the standard genetic algorithms and extensively use knowledge from the given example mappings. This allows a fast conversion of the algorithm towards a solution with a high fitness.

The genetic operator `Creator` is responsible for creating the population of solutions for each new evolution run (see Section 5.1.3). Each solution contains the genetic representation called genotype, which encodes a model transformation in a language-independent manner, i.e. independent from a concrete model transformation language. Based on the information encoded in the genotype,

the genetic operator `Decoder` generates a model transformation corresponding to this genotype using a concrete model transformation language. Thus, the `Creator` produces an encoding of a model transformation, which can be decoded in different model transformation languages. The genotype encodes a model transformation including left-hand sides and right-hand sides of its rules and its control flow. Thereby, the `Creator` leverages the knowledge from the given mappings between example models (language statistics information). Based on this knowledge, the introduced methods are designed to create a genotype that encodes a possibly correct model transformation, thus, leading to a faster convergence of the genetic algorithm.

The operator `Decoder` generates model transformations based on genotypes (see Section 5.1.4). The `Decoder` generates the explicit control flow chosen in the genotype as well as the generated rule sides. The `Decoder` in `mtbe` is defined for the graph-based model transformation language Henshin [5]. The chosen language Henshin is close to the presented genotype definition in Listing 5.2 that gives an advantage of a rather straight-forward generation of the phenotype. Henshin also supports the data flow between different objects that allows a stronger mapping of the object from the left- and right-hand side of a rule. In order to check and to improve the quality of resulting model transformations, Henshin provides extensive possibilities, among others its graph matching. Section 5.1.8 describes existing approaches, which provide techniques to verify such properties as syntactical and semantic correctness for model transformations.

The operator `Evaluator` is used to calculate the fitness of the model transformations generated by the `Decoder` (see Section 5.1.5). As a result of this evaluation, each model transformation gets a fitness value characterizing its suitability as a solution for the given MTBE problem. The `Evaluator` introduces coverage of the input model mappings by a generated model transformation as a fitness measurement that has to be maximized during the optimization. For the coverage computation, the graph matching of Henshin is extended, in order to compute possible complete and partial matches of each transformation rule of a model transformation with each model from given example model. Using the information about matches, a model transformation is evaluated based on the existence of a complete match (match coverage) but also based on the extent of their coverage of the models (model coverage). The collected information is also used by the genetic operator `Mutator` that modifies model transformations with the goal to improve their fitness.

After the fitness of model transformations is evaluated, the genetic operator `Selector` chooses the fittest solutions according to a certain selection strategy (see Section 5.1.6). Several selection strategies were investigated in this thesis. The elitism strategy that preserves a subset of the best solutions for the next population and selects a subset of the solutions for the mutation to fill the remaining part was realized. The `Selector` following this strategy has the advantage of preserving the best solution over the generations till the final evolution run. Additionally, the fitness of the mutated solutions is normally better

than the fitness of newly randomly generated solutions because of the large solution space of possible model transformations. This facts fosters the convergence of the genetic algorithm. Another selection strategy is similar to the described above with the difference of adding a small subset of new solutions to the next population. New solutions might have a genotype that can be mutated to more optimal solutions than existing ones. This strategy was also implemented by the `Selector` and tried out in the evaluation.

The final genetic operator `Mutator` modifies genotypes of the model transformations given by the `Selector` with the goal to improve their fitness (see Section 5.1.7). Typically, approaches using genetic algorithms perform an arbitrary mutation realizing elementary changes on the genotype. Such elementary changes imitate the evolutionary process in nature and aim to converge to an optimal solution, when the process runs over a large amount of generations. The `Mutator` of mtbe is designed as a heavy-weighted operator realizing larger changes with the goal to foster the convergence of the genetic algorithm. For that, the `Mutator` leverages the information about the objects hampering a complete match of a rule to a model, about the objects producing incorrect attribute values, and about the model objects missing in the rule. Using this information, mtbe obtains transformations with a high coverage faster.

The evaluation of the approach mtbe is presented in Section 5.2. It was evaluated on a test collection of service specifications written in SAWSDL [45] and the operation of service matching. The test collection is provided by the S3 Contest on Semantic Service Selection [123]. The goal of the evaluation is to show that the matching effectiveness computed on specifications transformed in the comprehensive core language using the mtbe approach is comparable to the matching effectiveness computed on specifications obtained using the semantic-preserving hand-written transformations. Section 5.2.2 shows that the approach produces expected results and Section 5.2.3 explains how the requirements formulated in Section 3.1.2 are fulfilled by the approach.

6.2 Future Work

This section introduces how the solution presented in this PhD thesis can be developed further. It starts with possibilities to extend and generalize the described approaches LOpt and mtbe. Afterwards, further concepts to improve the success of OTF markets are proposed.

Future work for LOpt In this PhD thesis, the notion of language optimality is investigated with respect to the operation of service matching. As described in Chapter 1, such automated market operations as service composition and service analysis have to be performed in OTF markets as well. During the service composition, a given requirements specification is analyzed and possible constituent services are identified for a composed service. Service specifications of the constituent services are generated based on this requirements specification. Based on these specifications, suitable services are discovered for the usage in the composed service. Both kinds of specifications are written in the optimal core language of the considered OTF market as service matching defined for the optimal core language is used for service discovery. After the composed service is built, its formal specification written in the optimal core language is used in the service analysis, which is performed to check whether functional and non-functional requirements on the composed service hold.

The presented approach LOpt has to consider further automated market operations, in order to create a core language, which optimally supports these operations as well. For that, the notion of the core language optimality has to be extended by the optimality definition regarding further market operations. In the next step, the comprehensive core language has to be extended by the concepts necessary for the new market operations, e.g., special properties to describe a composed service. Then, the configuration approach has to be adapted. The set of market properties has to be extended by properties relevant for the new market operations. The set of configuration rules has to be extended by rules defining the configuration with respect to the new operations. As a result, the obtained optimal core language would support the optimal execution of all automated market operations defined for OTF markets.

In this PhD thesis, the approach LOpt aims to design an optimal core language for service specifications in OTF markets. However, this approach is not only limited to the domain of service specifications and might also be applicable to other kinds of specification languages. As future work, the presented approach can be generalized to develop an optimal specification language in an arbitrary domain for arbitrary automated operations performed on specifications in this language. For that purpose, the approach LOpt has to be applied in other domains, the missing requirements have to be identified and the approach has to be adapted correspondingly.

The approaches developed in this PhD thesis are aimed for market actors having a technical expertise. However, the entrance in the markets for market actors

without such knowledge has to be facilitated as well. One important group of such users consists of service requesters, who wants to use services provided in OTF markets but lack an expertise to formulate their requirements in a formal way. For such service requesters, a user-friendly approach for requirements specification is needed. This approach would facilitate the market entrance for service requesters without much technical expertise and, thus, contribute to the success of the worldwide service market.

Future work for mtbe The presented approach mtbe aims at learning a language transformation based on example mappings of semantically-equivalent specifications written in the given formal languages. As future work, this approach can be extended by enabling the by-example specification of requirements. Service requesters, who do not have the necessary expertise in creating formal specifications, could formulate their requirements in the form of concrete example data. The requesters would need to specify how these data have to be transformed by a service, thus, giving examples of its behavior. Then, the information hidden in the concrete data has to be extracted and formalized in a requirements specification suitable for the execution of automated market operations in OTF markets.

For learning specifications from concrete data, different techniques can be utilized. For the generation of service operation signatures and their data types, the given examples have to be typed over the terminology of the corresponding OTF market. Existing semantics about the terms in the market can be used to enrich the specifications by behavioral properties, e.g., pre-/postconditions for operation signatures. Also non-functional properties established for similar services in the considered OTF market have to be taken into account for the formal specification. As additional assistance for service requesters, possibilities to provide a user-friendly concrete syntax for their specification editors have to be investigated as well.

Future work on quality management of service specifications The success of market actors in OTF markets depends on the quality of their service specifications with respect to their needs and the execution of the automated market operations. Service providers are successful, if their services are used by service requesters as often as possible. However, only those services are used, which functional and non-functional properties really satisfy the requirements stated in the considered requirements specifications. Thus, a service provider is successful, if their services match for as many relevant requirements specifications as possible. Additionally, those services, which properties do not match the given requirements, have to be identified and should not be delivered as a result. Service requesters are successful, if their requirements specification expresses their requirements in a way that the most suitable composed service can be built or the most suitable existing services can be discovered.

When market actors enter an OTF market, they already have their own specifications. During the transformation of these specifications into the optimal core language, syntactic and semantic information can get lost. However, the quality of the transformed specifications influences the quality of the execution of the automated market operations. In order to improve the reliability of the operation results, existing specifications could be adapted before the transformation, in order to obtain most suitable specifications in the core language. Thus, a framework for the quality management of existing specifications has to be designed. This framework has to consist of two parts: 1. to evaluate the quality of a given specification, and 2. to suggest improvements for the specification.

In OTF Computing, the quality of a service or requirements specification given in a certain modeling language has to be evaluated with respect to the transformation into the optimal core language. It is important to identify the difference in service properties between existing specifications and the core language. Properties missing in specifications but present in the core language could be added by the market actor. In the case, if a market actor creates a specification from scratch, a specification language, whose specifications can be transformed into the optimal core language the best, can be recommended.

In her Master thesis [73], Kavitha Jagannath tackled the problem to identify the most appropriate design-by-contract specification language for a given context. Design-by-contract specifications describe the behavior of services using pre- and postconditions of service operations. A precondition describes the state of the system before the execution of an operation, and the postcondition describes the state after. Kavitha Jagannath investigated various design-by-contract specification languages and created a comparison scheme, using which the most suitable design-by-contract technique can be identified for a given context. This approach can be used in the scope of the framework for the quality management, in order to evaluate existing specifications and to choose most suitable specification languages for pre-/postconditions.

In the next step, existing specifications have to be improved so that the transformation into the optimal core language yields specifications leading to reliable results of the automated market operations. Vahide Taherinajafabadi worked on this topic in her Master thesis [147]. She designed a framework for continuous monitoring, analyzing and improving the quality of specifications. This framework monitors matching results obtained for a specification and, if these results do not pass the predefined quality gates, this specification has to be adapted. The adaptation is based on a knowledge base, which contains rules recommending an improvement. For example, a specification should be extended by a certain non-functional property because it is considered during the matching and the specification gets rated lower as this property is missing in it. As a result, market actors profit from the quality management because the results of the automated market operations for their specifications get improved.

Bibliography

[1] ACM, Inc. ACM Digital Library. Accessible online under `http://dl.acm.org`. Last access: 10.07.2014.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[3] David H. Akehurst, Steffen Zschaler, and W. Gareth J. Howells. OCL: Modularising the Language. *Electronic Communications of the European Association for the Study of Science and Technology (ECEASST)*, 9:1–20, 2008.

[4] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Publishing Company, Inc., 1st edition, 2010.

[5] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems Part I, MODELS 2010, Oslo, Norway, October 3-8, 2010*, volume 6394 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2010.

[6] Svetlana Arifulina. Towards a Framework for the Integration of Modeling Languages. In Ulrich W. Eisenecker and Christian Bucholdt, editors, *Proceedings of the Doctoral Symposium of the 5th International Conference on Software Language Engineering 2012, Dresden, Germany (SLE (Doctoral Symposium))*, volume 935 of *CEUR Workshop Proceedings*, pages 23–26. CEUR-WS.org, 2012.

[7] Svetlana Arifulina, Felix Mohr, Gregor Engels, Marie Christin Platenius, and Wilhelm Schäfer. Market-Specific Service Compositions: Specification and Matching. In Liang-Jie Zhang and Rami Bahsoon, editors, *2015 IEEE World Congress on Services, SERVICES 2015, New York City, NY, USA, June 27 - July 2, 2015*, pages 333–340. IEEE, 2015.

[8] Svetlana Arifulina, Marie Christin Platenius, Steffen Becker, Christian Gerth, Gregor Engels, and Wilhelm Schäfer. Market-Optimized Service

Specification and Matching. In *Proceedings of the 12th International Conference on Service-Oriented Computing, ICSOC 2014, Paris, France, November 3-6, 2014*, volume 8831 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2014.

[9] Svetlana Arifulina, Christian Soltenborn, and Gregor Engels. Coverage Criteria for Testing DMM Specifications. In L. Lambers A. Fish, editor, *Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2012), Tallinn, Estonia*, volume 47 of *Electronic Communications of the EASST*. European Association of Software Science and Technology (EASST), 2012.

[10] Svetlana Arifulina, Sven Walther, Matthias Becker, and Marie Christin Platenius. SeSAME: Modeling and Analyzing High-quality Service Compositions. In Ivica Crnkovic, Marsha Chechik, and Paul Grünbacher, editors, *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 839–842. ACM, 2014.

[11] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[12] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: the Common Algebraic Specification Language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[13] Jung Ho Bae and Heung Seok Chae. UMLSlicer: A Tool for Modularizing the UML Metamodel Using Slicing. In *Proceedings of 8th IEEE International Conference on Computer and Information Technology, CIT 2008, Sydney, Australia, July 8-11, 2008*, pages 772–777. IEEE, 2008.

[14] Zoltan Balogh and Dániel Varró. Model Transformation by Example Using Inductive Logic Programming. *Software and System Modeling*, 8(3):347–364, 2009.

[15] Alistair Barros and Daniel Oberle, editors. *Handbook of Service Description: USDL and it's Methods*, volume XXVI. Springer Science+Business Media, 2012.

[16] Alistair Barros, Daniel Oberle, Uwe Kylau, and Steffen Heinzl. *Handbook of Service Description: USDL and Its Methods*, chapter Design Overview of USDL, pages 187–225. Springer Science+Business Media, 2012.

[17] Rabih Bashroush, Ivor T. A. Spence, Pater Kilpatrick, T. John Brown, Wasif Gilani, and Matthias Fritzsche. ALI: An Extensible Architecture Description Language for Industrial Applications. In *Proceedings of the 15th AnnualIEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008), Belfast, Ireland, March 31-April 4, 2008*, pages 297–304. IEEE, 2008.

[18] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio Component Model for Model-driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, January 2009.

[19] Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios S. Kolovos, Ivan Kurtev, and Richard F. Paige. A Canonical Scheme for Model Composition. In *Proceedings of the 2nd European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006*, volume 4066 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2006.

[20] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A Survey on Metaheuristics for Stochastic Combinatorial Optimization. *Natural Computing*, 8(2):239–287, 2009.

[21] Enrico Biermann. Local Confluence Analysis of Consistent EMF Transformations. *Electronic Communication of the European Association of Software Science and Technology (ECEASST)*, 38, 2011.

[22] Paul Börding, Melanie Bruns, and Marie Christin Platenius. Comprehensive Service Matching with MatchBox. In Elisabetta Di Nitto, Mark Harman, and Patrick Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 974–977. ACM, 2015.

[23] Kristopher Born, Thorsten Arendt, Florian Heß, and Gabriele Taentzer. Analyzing Conflicts and Dependencies of Rule-Based Transformations in Henshin. In Alexander Egyed and Ina Schaefer, editors, *Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering, FASE 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015.*, volume 9033 of *Lecture Notes in Computer Science*, pages 165–168. Springer, 2015.

[24] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[25] Matthias Bräuer and Henrik Lochmann. Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In *Proceedings of the 4th International Workshop on (Software) Language Engineering (ATEM'07) co-located with the 10th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2007), Nashville, Tennessee, September 30-October 5, 2007*. IEEE, 2007.

[26] Charles E. Campbell, Andrew Eisenberg, and Jim Melton. XML Schema. *SIGMOD Rec.*, 32(2):96–101, June 2003.

[27] Albertas Caplinskas, Audrone Lupeikiene, and Olegas Vasilecas. A Framework to Analyse and Evaluate Information Systems Specification Languages. In *Proceedings of the 6th East European Conference on Advances in Databases and Information Systems, ADBIS 2002, Bratislava, Slovakia, September 8-11, 2002*, volume 2435 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2002.

[28] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation REC-wsdl20-20070626, `http://www.w3.org/TR/2007/REC-wsdl20-20070626`, July 2007. Last access: 07.08.2014.

[29] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., 2000.

[30] Krzysztof Czarnecki and Simon Helsen. Classification of Model Transformation Approaches. In *Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, CA, USA*, 2003.

[31] Krzysztof Czarnecki and Simon Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[32] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged Configuration Through Specialization and Multilevel Configuration of Feature Models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[33] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. A Comprehensive Approach for the Development of Modular Software Architecture Description Languages. *ACM Transactions on Software Engineering and Methodology*, 14(2):199–245, April 2005.

[34] Eric M. Dashofy, van der André Hoek, and Richard N Taylor. A Highly-Extensible, XML-Based Architecture Description Language. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001), Amsterdam, The Netherlands, August 28-31, 2001*, WICSA '01, pages 103–112. IEEE, 2001.

[35] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The Web Service Modeling Language WSML: An Overview. In *Proceedings of the 3rd European Semantic Web Conference, The Semantic Web: Research and Applications, ESWC 2006, Budva, Montenegro, June 11-14, 2006*, ESWC'06, pages 590–604. Springer, 2006.

[36] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In Marc Schoenauer, Kalyanmoy Deb, Gnther Rudolph, Xin Yao, Evelyne Lutton, JuanJulian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 849–858. Springer Berlin Heidelberg, 2000.

[37] Davide Di Ruscio, Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio. Developing Next Generation ADLs Through MDE Techniques. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 85–94, 2010.

[38] Corinna Dohle and Leena Suhl. An Optimization Model for the Optimal Usage of Water Tanks in Water Supply Systems. In *Proceedings of the International Conference on Applied Mathematical Optimization and Modelling (APMOD)*, pages 404–408. Books on Demand, 2012.

[39] Xavier Dolques, Aymen Dogui, Jean-Rémy Falleri, Marianne Huchard, Clémentine Nebut, and François Pfister. Easing Model Transformation Learning with Automatically Aligned Examples. In *Proceedings of the 7th European Conference on Modelling Foundations and Applications, ECMFA 2011, Birmingham, UK, June 6 - 9, 2011*, volume 6698 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2011.

[40] Xavier Dolques, Marianne Huchard, and Clémentine Nebut. From Transformation Traces to Transformation Rules: Assisting Model Driven Engineering approach with Formal Concept Analysis. In *Proceedings of the 17th International Conference on Conceptual Structures (ICCS 2009), Moscow, Russian Federation*, volume 483, pages 15–29. CEUR-WS, 2009.

[41] Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Philippe Reitz. Learning Transformation Rules from Transformation Examples: An

Approach Based on Relational Concept Analysis. In *Workshops Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference, EDOCW 2010, Vitória, Brazil, 25-29 October 2010*, pages 27–32. IEEE Computer Society, 2010.

[42] Hartmut Ehrig, Julia Padberg, and Fernando Orejas. From Basic Views and Aspects to Integration of Specification Formalisms. In *Current Trends in Theoretical Computer Science, Entering the 21th Century*, pages 202–214. World Scientific, 2001.

[43] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer-Verlag New York, Inc., 1st edition, 2007.

[44] Zhiqiang Fan, Tao Yue, and Li Zhang. A Generic Framework for Deriving Architecture Modeling Methods for Large-scale Software-intensive Systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013*, SAC '13, pages 1750–1757. ACM, 2013.

[45] Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. W3C Recommendation REC-sawsdl-20070828, `http://www.w3.org/TR/2007/REC-sawsdl-20070828`, August 2007. Last access: 07.08.2014.

[46] Martin Faunes, Houari A. Sahraoui, and Mounir Boukadoum. Genetic-Programming Approach to Learn Model Transformation Rules from Examples. In *Proceedings of the 6th International Conference on Theory and Practice of Model Transformations, ICMT 2013, Budapest, Hungary, June 18-19, 2013*, volume 7909 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013.

[47] Daniel Ferrante. Software Licensing Models: What's Out There? *IT Professional*, 8:24–29, November 2006.

[48] Anthony Finkelstein, Jeff Kramer, Bashar Nuseibeh, L. Finkelstein, and Michael Goedicke. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–57, 1992.

[49] Franck Fleurey, Benoit Baudry, Pierre-Alain Muller, and Yves Le Traon. Qualifying Input Test Data for Model Transformations. *Software and System Modeling*, 8(2):185–203, 2009.

[50] D.B. Fogel and J.W. Atmar. Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes using Linear Systems. *Biological Cybernetics*, 63(2):111–114, 1990.

[51] The Eclipse Foundation. Henshin. `https://www.eclipse.org/henshin`. Last access: 12.06.2015.

[52] Iván García-Magariño, Rubén Fuentes-Fernández, and Jorge J. Gómez-Sanz. A Framework for the Definition of Metamodels for Computer-Aided Software Engineering Tools. *Information and Software Technology*, 52(4):422–435, April 2010.

[53] Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández. Model Transformation By-Example: An Algorithm for Generating Many-to-Many Transformation Rules in Several Model Transformation Languages. In *Proceedings of the 2nd International Conference Theory and Practice of Model Transformations, ICMT 2009, Zurich, Switzerland, June 29-30, 2009*, volume 5563 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2009.

[54] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. Model-Driven Language Engineering: The ASMETA Case Study. In *Proceedings of the Third International Conference on Software Engineering Advances, ICSEA 2008, October 26-31, 2008, Sliema, Malta*, pages 373–378. IEEE Computer Society, 2008.

[55] A. Gligor, T. Turc, C. D. Dumitru, and Al. Morar. Development of an Extensible Description Language for Virtual Instrumentation. *International Conference on Automation, Quality and Testing, Robotics*, 3:1–6, 2010.

[56] Thomas Goldschmidt, Steffen Becker, and Erik Burger. Towards a Tool-Oriented Taxonomy of View-Based Modelling. In *Proceedings of the Modellierung 2012, 14.-16. März 2012, Bamberg, Deutschland*, volume 201 of *LNI*, pages 59–74. GI, 2012.

[57] Google. Google. Accessible online under `http://google.de/`. Last access: 05.08.2014.

[58] Google. Google scholar. Accessible online under `http://scholar.google.de`. Last access: 01.07.2014.

[59] Tyrone Grandison and Morris Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys & Tutorials*, 3(4):2–16, October 2000.

[60] Vincenzo Grassi, Raffaela Mirandola, Enrico Randazzo, and Antonino Sabetta. KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability. In *The Common Component Modeling Example*, volume 5153 of *Lecture Notes in Computer Science*, pages 327–356. Springer, 2008.

[61] Martin Große-Rhode. On Model Integration and Integration Modelling: Introduction to the Subject Area Integration Modelling. In *SoftSpez Final Report on Integration of Software Specification Techniques for Applications in Engineering, Priority Program SoftSpez of the German Research Foundation (DFG)*, volume 3147 of *Lecture Notes in Computer Science*, pages 567–581. Springer, 2004.

[62] Giancarlo Guizzardi, Luis F. Pires, and Marten van Sinderen. An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages. In *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005*, pages 691–705, 2005.

[63] Scott Hamilton and Norman L. Chervany. Evaluating Information System Effectiveness - Part I: Comparing Evaluation Approaches. *MIS Quarterly*, 5(3):55–69, 1981.

[64] Jan Hendrik Hausmann. *Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages*. PhD thesis, University of Paderborn, 2005.

[65] Brian Henderson-Sellers, Muhammad Atif Qureshi, and Cesar Gonzalez-Perez. Towards an Interoperable Metamodel Suite: Size Assessment as One Input. *International Journal of Software and Informatics*, 6(2):111–124, 2012.

[66] Mamoun Hirzalla, Jane Cleland-Huang, and Ali Arsanjani. A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures. In George Feuerlicht and Winfried Lamersdorf, editors, *Revised Selected Papers of the International Workshops of Service-Oriented Computing - ICSOC 2008 Workshops, Sydney, Australia, December 1st, 2008*, volume 5472 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 2008.

[67] Mathias Hülsbusch, Barbara König, Arend Rensink, Maria Semenyak, Christian Soltenborn, and Heike Wehrheim. Full Semantics Preservation in Model Transformation - A Comparison of Proof Techniques. In S. Merz D. M'ery, editor, *Proceedings of the 8th International Conference on Integrated Formal Methods (IFM 2010)*, volume 6396 of *LNCS*, pages 183–198, Berlin/Heidelberg, 2010. Springer.

[68] IEEE. IEEE Xplore Digital Library. Accessible online under `http://ieeexplore.ieee.org/Xplore/home.jsp`. Last access: 10.07.2014.

[69] ISO/IEC. "Information Technology - Syntactic Metalanguage - Extended BNF". http://www.cl.cam.ac.uk/ mgk25/iso-14977.pdf, 1996. International Standard ISO/IEC 14977:1996.

[70] ISO/IEC. Software engineering - Product quality - Part 1: Quality model. International Standard ISO/IEC 9126-1:2001, 2001.

[71] ISO/IEC. Information technology - Object Management Group : Meta Object Facility (MOF) Core. International Standard ISO/IEC 19508:2014(E), 2014.

[72] Kazunori Iwasa, Jacques Durand, Tom Rutt, Mark Peel, Sunil Kunisetty, and Doug Bunting. Web Services Reliable Messaging TC WS-Reliability 1.1. OASIS Standard, 15 November 2004. Accessible online under `http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/`. Last access: 15.08.2014.

[73] Kavitha Jagannath. Service Specification in On-The-Fly Computing, Contract-Based Specifications. Master's thesis, Univesity of Paderborn, 2012.

[74] Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2006, Genova, Italy, October 1-6, 2006*, MoDELS'06, pages 528–542. Springer, 2006.

[75] Gerti Kappel, Philip Langer, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Model Transformation By-Example: A Survey of the First Wave. In *Conceptual Modelling and Its Theoretical Foundations - Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday*, volume 7260 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2012.

[76] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling , DSM'09, Helsinki School of Economics. TR no B-108. Orlando, Florina, USA, October 2009*, pages 7–13, 2009.

[77] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web Services Choreography Description Language Version 1.0. W3C Candidate Recommendation CR-ws-cdl-10-20051109, `http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109`, November 2005. Last access: 08.08.2014.

[78] James P. Kelly. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[79] Marouane Kessentini, Houari A. Sahraoui, and Mounir Boukadoum. Model Transformation as an Optimization Problem. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2008, Toulouse, France, September 28 - October 3, 2008*, volume 5301 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2008.

[80] Marouane Kessentini, Houari A. Sahraoui, Mounir Boukadoum, and Omar Benomar. Search-based Model Transformation by Example. *Software and System Modeling*, 11(2):209–226, 2012.

[81] Barbara Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen G. Linkman. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Information & Software Technology*, 51(1):7–15, 2009.

[82] Barbara Kitchenham and Stuart Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[83] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, volume 5 of *Genetic Programming Series*. Springer US, 2003.

[84] Holger Krahn, Bernhard Rumpe, and Steven Völkel. MontiCore: Modular Development of Textual Domain Specific Languages. In *Proceedings of the 46th International Conference on Objects, Components, Models and Patterns, TOOLS EUROPE 2008, Zurich, Switzerland, June 30 - July 4, 2008*, volume 11 of *Lecture Notes in Business Information Processing*, pages 297–315. Springer, 2008.

[85] John Krogstie and Sofie de Flon Arnesen. Assessing Enterprise Modeling Languages Using a Generic Quality Framework. In *Information Modeling Methods and Methodologies*, pages 63–79. Idea Group, 2005.

[86] Philip Langer, Manuel Wimmer, and Gerti Kappel. Model-to-Model Transformations By Demonstration. In *Proceedings of the 3rd International Conference on Theory and Practice of Model Transformations, ICMT 2010, Malaga, Spain, June 28-July 2, 2010*, volume 6142 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2010.

[87] Henry Lieberman. Programming by Example: Introduction. *Communications of the ACM*, 43(3):72–74, 2000.

[88] Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. Opt4J: A Modular Framework for Meta-heuristic Optimization. In Natalio Krasnogor and Pier Luca Lanzi, editors, *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Dublin, Ireland, July 12-16, 2011*, pages 1723–1730. ACM, 2011.

[89] Ivano Malavolta, Henry Muccini, and Patrizio Pelliccione. DUALLY: A Framework for Architectural Languages and Tools Interoperability. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2008, 15-19 September 2008, L'Aquila, Italy*, pages 483–484. IEEE, 2008.

[90] Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Damien A. Tamburri. Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies. *IEEE Transactions on Software Engineering*, 36(1):119–140, 2010.

[91] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission, `http://www.w3.org/Submission/OWL-S`, November 2004. Last access: 08.08.2014.

[92] Erika McCallister, Tim Grance, and Karen Scarfone. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). Special Publication 800-122, Recommendations of the National Institute of Standards and Technology, `http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf`, 2010. Last access: 25.03.2015.

[93] Deborah L. McGuinness and Frank van Harmelen. OWL 2 Web Ontology Language. W3C Recommendation owl2-overview, `http://www.w3.org/TR/owl2-overview`, December 2012. Last access: 08.08.2014.

[94] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[95] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, March 2006.

[96] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and How to Develop Domain-specific Languages. *ACM Computing Surveys*, 37(4):316–344, December 2005.

[97] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., 1st edition, 1988.

[98] Bart Meyers, Antonio Cicchetti, Esther Guerra, and Juan de Lara. Composing Textual Modelling Languages in Practice. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling, Innsbruck, Austria*, MPM '12, pages 31–36. ACM, 2012.

[99] Peter D. Mosses. CoFI: The Common Framework Initiative for Algebraic Specification and Development. In *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, TAPSOFT'97, Lille, France, April 14-18, 1997*, volume 1214 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 1997.

[100] Liping Mu, Terje Gjøsæter, Andreas Prinz, and Merete Skjelten Tveit. Specification of Modelling Languages in a Flexible Meta-model Architecture. In *Proceedings of the 4th European Conference on Software Architecture (ECSA'10): Companion Volume, Copenhagen, Denmark, August 23-26, 2010*, ACM International Conference Proceeding Series, pages 302–308. ACM, 2010.

[101] Anthony Nadalin, Marc Goodner, Martin Gudgin, David Turner, Abbie Barbir, and Hans Granqvist. WS-Trust 1.4. OASIS Standard, 25 April 2012. Accessible online under `http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/ws-trust-1.4-errata01-complete.pdf`. Last access: 15.08.2014.

[102] Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). OASIS Standard 200401, March 2004. Accessible online under `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`. Last access: 15.08.2014.

[103] OASIS. Advancing open standards for the information society. Accessible online under `https://www.oasis-open.org/standards`. Last access: 08.08.2014.

[104] Object Management Group, Inc. OMG Homepage. Accessible online under `http://www.omg.org`. Last access: 15.08.2014.

[105] Object Management Group, Inc. OMG Specifications. Accessible online under `http://www.omg.org/spec/index.htm`. Last access: 15.08.2014.

[106] Object Management Group (OMG). UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification (Version 1.1). Standard document URL: `http://www.omg.org/spec/QFTP/1.1/PDF`, April 2008.

[107] Object Management Group (OMG). OMG Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems (Version 1.1). Standard document URL: `http://www.omg.org/spec/MARTE/1.1`, June 2011.

[108] Object Management Group (OMG). OMG Unified Modeling Language (OMG UML), Superstructure (Version 2.4.1). Standard document URL: `http://www.omg.org/spec/UML/2.4.1/Superstructure`, August 2011.

[109] Object Management Group (OMG). Object Constraint Language (Version 2.4). `http://www.omg.org/spec/OCL/2.4`, February 2014. Internation Standard ISO/IEC 19507:2012(E).

[110] The National Institute of Standards and Technology (NIST). `http://www.nist.gov/`. Last access: 25.03.2015.

[111] Arto Ojala. Software-as-a-Service Revenue Models. *IT Professional*, 15(3):54–59, 2013.

[112] Sven Overhage. UnSCom: A Standardized Framework for the Specification of Software Components. In *Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a NetworkedWorld, Net.ObjectDays 2004, Erfurt, Germany, September 27-30, 2004*, volume 3263 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2004.

[113] Richard F. Paige, Jonathan S. Ostroff, and Phillip J. Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665 – 675, 2000.

[114] Patrizio Pelliccione, Paola Inverardi, and Henry Muccini. CHARMY: A Framework for Designing and Verifying Architectural Specifications. *IEEE Transactions on Software Engineering*, 35(3):325–346, 2009.

[115] Philip Lief Group. Roget's 21st Century Thesaurus, Third Edition Copyright. Accessible online under `http://thesaurus.com`. Last access: 24.07.2014.

[116] Marie Christin Platenius, Svetlana Arifulina, Ronald Petrlic, and Wilhelm Schäfer. Matching of Incomplete Service Specifications Exemplified by Privacy Policy Matching. In Guadalupe Ortiz and Cuong Tran, editors, *Advances in Service-Oriented and Cloud Computing*, volume 508 of *Communications in Computer and Information Science*, pages 6–17. Springer International Publishing, 2015.

[117] Marie Christin Platenius, Svetlana Arifulina, and Wilhelm Schäfer. MatchBox: A Framework for Dynamic Configuration of Service Matching Processes. In Philippe Kruchten, Steffen Becker, and Jean-Guy Schneider, editors, *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering, CBSE 2015, Montreal, QC, Canada, May 4-8, 2015*, pages 75–84. ACM, 2015.

[118] Rachel Pottinger and Philip A. Bernstein. Merging Models Based on Given Correspondences. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, Berlin, Germany*, VLDB '03, pages 862–873. VLDB Endowment, 2003.

[119] Ralf Reussner, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Koziolek, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg. *The Palladio Component Model*. Karlsruhe Reports in Informatics; 2011,14. Karlsruhe, 2011. `http://nbn-resolving.org/urn:nbn:de:swb:90-225038`.

[120] Jason E. Robbins, Nenad Medvidovic, David F. Redmiles, and David S. Rosenblum. Integrating Architecture Description Languages with a Standard Design Method. In *Proceedings of the 20th International Conference on Software Engineering, ICSE 98, Kyoto, Japan, April 19-25, 1998*, ICSE '98, pages 209–218. IEEE Computer Society, 1998.

[121] Michael Rosemann and Wil M. P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1 – 23, 2007.

[122] Michael Rudolf. Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *6th International Workshop on Theory and Application of Graph Transformations (TAGT'98), Selected Papers, Paderborn, Germany, November 16-20, 1998*, volume 1764 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 1998.

[123] S3 Contest Organisation Committee. Annual international contest s3 on semantic service selection. `http://www-ags.dfki.uni-sb.de/~klusch/s3/`. Last access: 18.10.2015.

[124] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Generation of Operational Transformation Rules from Examples of Model Transformations. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012*, volume 7590 of *Lecture Notes in Computer Science*, pages 546–561. Springer, 2012.

[125] Hajer Saada, Xavier Dolques, Marianne Huchard, Clémentine Nebut, and Houari A. Sahraoui. Learning Model Transformations from Examples using FCA: One for All or All for One? In *Proceedings of The Ninth International Conference on Concept Lattices and Their Applications, Fuengirola (Málaga), Spain, October 11-14, 2012*, volume 972 of *CEUR Workshop Proceedings*, pages 45–56. CEUR-WS.org, 2012.

[126] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1983.

[127] Sebastian Schlauderer and Sven Overhage. How Perfect are Markets for Software Services? An Economic Perspective on Market Deficiencies and Desirable Market Features. In *Proceedings of the 19th European Conference on Information Systems, ECIS 2011, Helsinki, Finland, June 9-11, 2011*, 2011.

[128] Bran Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2007), 7-9 May 2007, Santorini Island, Greece*, ISORC '07, pages 2–9. IEEE Computer Society, 2007.

[129] Adel Smeda, Mourad Oussalah, and Tahar Khammaci. MADL: Meta Architecture Description Language. In *Proceedings of the Third ACIS Int'L Conference on Software Engineering Research, Management and Applications (SERA 2005), 11-13 August 2005, Mt. Pleasant, MI, USA*, SERA '05, pages 152–159. IEEE Computer Society, 2005.

[130] Harry M. Sneed. Measuring Web Service Interfaces. In Giuseppe A. Di Lucca and Holger M. Kienle, editors, *Proceedings of the 12th IEEE International Symposium on Web Systems Evolution, WSE 2010, September 17-18, 2010, Timisoara, Romania*, pages 111–115. IEEE Computer Society, 2010.

[131] Monique Snoeck, Stephan Poelmans, and Guido Dedene. A Layered Software Specification Architecture. In *Proceedings of the 19th International Conference on Conceptual Modeling (Conceptual Modeling - ER 2000), Salt Lake City, Utah, USA, October 9-12, 2000*, volume 1920 of *Lecture Notes in Computer Science*, pages 454–469. Springer, 2000.

[132] Christian Soltenborn. *Quality Assurance with Dynamic Meta Modeling*. PhD thesis, University of Paderborn, 2013.

[133] Christian Soltenborn and Gregor Engels. Towards Test-Driven Semantics Specification. In B. Selic A. Schürr, editor, *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*

(MODELS 2009), Denver, Colorado (USA), volume 5795 of *LNCS*, pages 378–392. Springer, 2009.

[134] Diomidis Spinellis. Notable Design Patterns for Domain-specific Languages. *Journal of Systems and Software*, 56(1):91–99, February 2001.

[135] Springer, Part of Springer Science+Business Media. Springer Link. Accessible online under `http://link.springer.com`. Last access: 10.07.2014.

[136] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[137] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven Software Development - Technology, Engineering, Management*. Pitman, 2006.

[138] Thomas Stahl, Markus Völter, Sven Efftinge, and Arno Haase. *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*, volume 2. dpunkt.verlag, 2007.

[139] Athanasios Staikopoulos and Behzad Bordbar. A Comparative Study of Metamodel Integration and Interoperability in UML and Web Services. In *Proceedings of the 1st European Conference on Model Driven Architecture: Foundations and Applications, ECMDA-FA 2005, Nuremberg, Germany, November 7-10, 2005*, ECMDA-FA'05, pages 145–159. Springer, 2005.

[140] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2nd edition, 2009.

[141] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[142] Michael Strommer and Manuel Wimmer. A Framework for Model Transformation By-Example: Concepts and Tool Support. In *Proceedings of the 46th International Conference on Objects, Components, Models and Patterns, TOOLS EUROPE 2008, Zurich, Switzerland, June 30 - July 4, 2008*, volume 11 of *Lecture Notes in Business Information Processing*, pages 372–391. Springer, 2008.

[143] Daniel Strüber, Gabriele Taentzer, Stefan Jurack, and Tim Schäfer. Towards a Distributed Modeling Process Based on Composite Models. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering, FASE 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013*, volume 7793 of *Lecture Notes in Computer Science*, pages 6–20. Springer, 2013.

[144] Gabriele Taentzer. What Algebraic Graph Transformations Can Do For Model Transformations. *Electronic Communication of the European Association of Software Science and Technology (ECEASST)*, 30, 2010.

[145] Gabriele Taentzer, Thorsten Arendt, Claudia Ermel, and Reiko Heckel. Towards Refactoring of Rule-based, In-place Model Transformation Systems. In *Proceedings of the First Workshop on the Analysis of Model Transformations, Innsbruck, Austria*, AMT '12, pages 41–46. ACM, 2012.

[146] Gabriele Taentzer, Karsten Ehrig, Esther Guerra, Juan De Lara, Tihamer Levendovszky, Ulrike Prange, Daniel Varro, and et al. Model Transformations by Graph Transformations: A Comparative Study. In *Model Transformations in Practice Workshop (MTIP) at MoDELS Conference, Montego Bay, Jamaica*, October 2005.

[147] Vahide Taherinajafabadi. Quality Management of Service Specifications in On-The-Fly Computing. Master's thesis, Univesity of Paderborn, 2014.

[148] The World Wide Web Consortium (W3C). Web of Services. `http://www.w3.org/standards/webofservices`. Last access: 07.08.2014.

[149] Matthias Tichy, Christian Krause, and Grischa Liebel. Detecting Performance Bad Smells for Henshin Model Transformations. In Benoit Baudry, Jürgen Dingel, Levi Lucio, and Hans Vangheluwe, editors, *Proceedings of the Second Workshop on the Analysis of Model Transformations (AMT 2013), Miami, FL, USA, September 29, 2013*, volume 1077 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[150] Kenneth J. Turner. Specification Architecture Illustrated in a Communications Context . *Computer Networks and {ISDN} Systems*, 29(4):397 – 411, 1997. Specification Architecture.

[151] Universität Paderborn, SFB 901 "On-The-Fly Computing". Official webpage of the Collaborative Research Centre 901 "On-The-Fly Computing". Accessible under `http://sfb901.uni-paderborn.de/`. Last access: 13.10.2015.

[152] Universität Paderborn, SFB 901 "On-The-Fly Computing", Subproject B1. Official webpage of the Collaborative Research Centre 901 "On-The-Fly Computing", SSE - Service Specification Environment. Accessible under `http://sfb901.uni-paderborn.de/sfb-901/projects/tools-demonstration-systems/service-specification-environment.html`. Last access: 13.10.2015.

[153] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, 1979.

[154] Dániel Varró. Model Transformation by Example. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2006, Genova, Italy, October 1-6, 2006*, volume 4199 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2006.

[155] Dániel Varró and Zoltan Balogh. Automating Model Transformation by Example Using Inductive Logic Programming. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, pages 978–984. ACM, 2007.

[156] F. Vernadat. UEML: Towards a Unified Enterprise Modelling Language. *International Journal of Production Research*, 40(17):4309–4321, 2002.

[157] Markus Vöelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart C. L. Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013.

[158] Markus Völter and Iris Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *Proceedings of the 11th International Conference on Software Product Lines (SPLC 2007), Kyoto, Japan, September 10-14, 2007*, pages 233–242. IEEE Computer Society, 2007.

[159] Tobias Walter and Jürgen Ebert. Combining DSLs and Ontologies Using Metamodel Integration. In *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages, DSL 2009, Oxford, UK, July 15-17, 2009*, volume 5658 of *Lecture Notes in Computer Science*, pages 148–169. Springer, 2009.

[160] Sven Walther and Heike Wehrheim. Verified Service Compositions by Template-Based Construction. In Ivan Lanese and Eric Madelaine, editors, *Formal Aspects of Component Software*, pages 31–48. Springer, 2015.

[161] Christian Wende, Nils Thieme, and Steffen Zschaler. A Role-Based Approach towards Modular Language Engineering. In *Proceedings of the Second International Conference on Software Language Engineering, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers*, volume 5969 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2010.

[162] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards Model Transformation Generation By-Example. In *Proceedings of the 40th Hawaii International International Conference on Systems Science (HICSS-40 2007), CD-ROM / Abstracts Proceedings, 3-6 January 2007, Waikoloa, Big Island, HI, USA*, pages 285–294. IEEE Computer Society, 2007.

[163] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 2nd (Morgan Kaufmann Series in Data Management Systems) edition, 2005.

[164] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100. CIMNE, Barcelona, Spain, 2002.

[165] Moshé M. Zloof. Query by Example. In *Proceedings of the American Federation of Information Processing Societies (AFIPS): 1975 National Computer Conference, 19-22 May 1975, Anaheim, CA, USA*, volume 44 of *AFIPS Conference Proceedings*, pages 431–438. AFIPS Press, 1975.

[166] Steffen Zschaler, Dimitrios S. Kolovos, Nikolaos Drivalos, Richard F. Paige, and Awais Rashid. Domain-specific Metamodelling Languages for Software Language Engineering. In *Proceedings of the Second International Conference on Software Language Engineering, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers*, volume 5969 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2010.