Measuring Transformation-induced Uncertainty in Service Matching: A Feasibility Study

Bachelor’s Thesis
Submitted to the Software Engineering Research Group in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

by Mirko Janosch Bunse

Thesis Supervisor:
Prof. Dr. Wilhelm Schäfer and Prof. Dr. Stefan Böttcher

Paderborn, September 2014
Declaration
(Translation from German)

I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Original Declaration Text in German:

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

_____________________________________________  __________________________________________
City, Date  Signature
## Contents

### 1 Introduction
- 1.1 Problem Statement ........................................... 1
- 1.2 Running Example ............................................. 2
- 1.3 Approach .......................................................... 3
  - 1.3.1 UML-to-SSL Transformation ............................... 3
  - 1.3.2 Uncertainty Measurement ................................. 3
- 1.4 Overview .......................................................... 4

### 2 Fundamentals
- 2.1 Service Specification ........................................ 5
- 2.2 Service Matching ............................................... 6
- 2.3 Model Transformations ........................................ 6
- 2.4 SSL Meta-Model .................................................. 7
  - 2.4.1 Repository ................................................... 8
  - 2.4.2 OperationInterface .......................................... 8
  - 2.4.3 ExtendedOperationSignature ............................. 8
  - 2.4.4 ExtendedParameter ......................................... 9
  - 2.4.5 SimpleOntologyDataType .................................. 9
  - 2.4.6 OntologyFailureType ...................................... 10
- 2.5 UML Meta-Model .................................................. 10
  - 2.5.1 NamedElement .............................................. 11
  - 2.5.2 Package ..................................................... 11
  - 2.5.3 Model ........................................................ 11
  - 2.5.4 Interface .................................................... 11
  - 2.5.5 Operation ................................................... 12
  - 2.5.6 Parameter ................................................... 12
  - 2.5.7 Type ........................................................ 13
  - 2.5.8 Property ..................................................... 13

### 3 A UML-to-SSL Transformation
- 3.1 Transformation of Operation Signatures .................... 15
  - 3.1.1 Top-level Model Elements ................................. 15
  - 3.1.2 Operation Interfaces ...................................... 16
  - 3.1.3 Operation Signatures ..................................... 17
  - 3.1.4 Operation Parameters .................................... 17
  - 3.1.5 Parameter Types .......................................... 18
## Contents

3.1.6 Exceptions and Failure Types ........................................ 19
3.1.7 Interface Properties .................................................. 19
3.2 Information Loss Cause: Parameter Multiplicities .................... 20

### 4 Measuring Transformation-induced Uncertainty ................... 23

4.1 Feasibility ........................................................................ 23
4.1.1 Obtaining the Probability of Incorrect Matching Results ........ 24
4.1.2 Information Loss Measurement ....................................... 25
4.2 Representation of Uncertainty .......................................... 26
4.2.1 The Fuzzy Set of Correct Matching Results ...................... 26
4.2.2 Measurement Representation ......................................... 28
4.2.3 Aggregation of Uncertainty ........................................... 28
4.2.4 Sidemark: Uncertainty Induced by Multiplicities ................ 29
4.3 The Measurement Process ............................................... 30
4.3.1 Occurrence Detection ................................................ 31
4.3.2 Occurrence Collection ................................................ 34
4.3.3 Match Validation ...................................................... 35
4.4 Implementation Notes .................................................... 36
4.4.1 Approach .................................................................. 36
4.4.2 User Guide .............................................................. 39

### 5 Evaluation and Discussion ............................................. 45

5.1 Internal Validity Evaluation .............................................. 45
5.2 Outcome Evaluation ........................................................ 46
5.2.1 Evaluation Example .................................................. 47
5.2.2 Graph Result Validation .............................................. 48
5.2.3 Service Choice .......................................................... 48
5.3 Impact Evaluation and Discussion ...................................... 49
5.3.1 Recommendations on Service Specification ...................... 49
5.3.2 SSL Multiplicity Extension ......................................... 49

### 6 Related Work .............................................................. 51

6.1 Vijapurwala: Specification Incompleteness ............................ 51
6.2 Transformation of Uncertain Models .................................... 52
6.3 Automatic Transformation Generation ............................... 53

### 7 Conclusions ............................................................... 55

7.1 Contributions ............................................................... 55
7.2 Future Work ..................................................................... 56
7.2.1 Service Choice Strategies in Uncertain Environments .......... 56
7.2.2 Information Loss Causes in other Service Aspects ............. 56
7.2.3 Dependency Analysis in the Measurement Process ............ 57
7.2.4 Specification Check Use Case ....................................... 58
# Appendix

## A Unconsidered Meta-Class Properties

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Unconsidered SSL Element Properties</td>
<td>59</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Repository</td>
<td>59</td>
</tr>
<tr>
<td>A.1.2</td>
<td>OperationInterface</td>
<td>60</td>
</tr>
<tr>
<td>A.1.3</td>
<td>ExtendedOperationSignature</td>
<td>60</td>
</tr>
<tr>
<td>A.1.4</td>
<td>ExtendedParameter</td>
<td>60</td>
</tr>
<tr>
<td>A.1.5</td>
<td>SimpleOntologyDataType</td>
<td>61</td>
</tr>
<tr>
<td>A.2</td>
<td>Unconsidered UML Element Properties</td>
<td>61</td>
</tr>
<tr>
<td>A.2.1</td>
<td>NamedElement</td>
<td>61</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Namespace</td>
<td>62</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Package</td>
<td>62</td>
</tr>
<tr>
<td>A.2.4</td>
<td>Model</td>
<td>63</td>
</tr>
<tr>
<td>A.2.5</td>
<td>Interface</td>
<td>63</td>
</tr>
<tr>
<td>A.2.6</td>
<td>Operation</td>
<td>63</td>
</tr>
<tr>
<td>A.2.7</td>
<td>Property</td>
<td>64</td>
</tr>
</tbody>
</table>

## B Potential Transformation Source Languages

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td></td>
<td>69</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Folder variant of the restoration interface .......................... 2
1.2 Collection variant of restoration interface ......................... 2
2.1 SSL meta-model .................................................... 7
2.2 UML meta-model, based on [Ob11b] ................................. 10
3.1 Transformation result of the collection variant .................... 21
4.1 Limited collection variant of the restoration interface ........... 24
4.2 Collection request for audio restoration ......................... 24
4.3 Transformation results of limited variant and collection request 25
4.4 Request for the restoration of audio folders ..................... 26
4.5 Interrelation between set membership and uncertainty value 27
4.6 Matching between folder variant and collection request ........ 27
4.7 The uncertainty measurement process ........................... 30
4.8 Recall of collection variant and transformation result .......... 31
4.9 Collection variant with uncertainty annotation .................. 31
4.10 Information loss not inducing uncertainty ....................... 32
4.11 Information loss annotation with assumptions flag .......... 32
4.12 Custom restoration request for discographies ................. 33
4.13 Folder variant and transformation result of discography request 33
4.14 Extended variant of the restoration interface .................. 34
4.15 Annotated collection request .................................. 35
4.16 The implementation architecture ................................ 36
4.17 The uncertainty report Ecore package ......................... 37
4.18 The transformation menu ...................................... 39
4.19 The transformation wizard ..................................... 40
4.20 The matching menu ............................................. 41
4.21 The matcher configuration dialog ............................... 41
4.22 Signature selection ............................................. 42
4.23 Uncertainty report selection ................................... 42
4.24 The Matching Results view ................................. 43
5.1 Evaluation service and request .................................. 47
5.2 Transformation result of evaluation service and request ....... 47
7.1 Uncertainty measurement process with dependency analysis .... 57
List of Tables

5.1 Validation of matching between evaluation service and request . . 48

B.1 Expectations on the fulfillment of language requirements . . . . . 68
1 Introduction

In On-The-Fly (OTF) Computing and other service-oriented paradigms, the automated discovery of software services constitutes one of the core challenges. Such discovery includes the matching of a client’s request with specifications of existing services. Based on these specifications, a matching can decide, how well an existing service fulfills the given request.

Matching service specifications and requests requires them to be specified in the same specification language. However, the OTF Computing paradigm envisions highly dynamic world-wide markets for software services. With the proposal, advance, and standardization of multiple different specification languages, the use of heterogenous service descriptions on these markets has to be expected. For instance, the Unified Modeling Language UML is a prominent representative of software specification languages. It is also proposed for service specification in particular. In order to match specifications from different languages, their transformation to a shared core language is crucial.

1.1 Problem Statement

The UML is a language broadly applied for software specification. Often, UML diagrams are created as development artifacts, which is why many service providers can be expected to already have UML specifications of their software when approaching the OTF market. However, UML specifications cannot be matched with existent OTF matchers. These matchers use the core language SSL that is less expressive than the UML allowing for efficient matching. Currently, UML specifications have to be translated manually to the SSL, what is a time-consuming task done in reoccurring patterns.

Moreover, a translation to the SSL is not guaranteed to preserve all semantics of a UML model. Since the UML is a highly expressive language with a wide scope, it includes concepts that have no equivalent in the SSL. All semantics expressed with these concepts cannot be expressed in the SSL and thus are lost in any transformation. However, lost semantics may impose restrictions on matchings, e.g., imply that some specifications do not match. In that case, matching results can be incorrect because they are only based on semantics that are still present, i.e., not lost. Since the correctness of the result is uncertain, we speak of transformation-induced uncertainty inherent to the matching result.
1. Introduction

Uncertainty (also called “fuzziness”) is a known issue in service matching [Pla13, PvdB98]. Currently, requesters can not assess the risk of having incorrect results because existent matchers do not consider transformation-induced uncertainty.

1.2 Running Example

The quality of audio recordings has constantly improved over the last decades. Today, older recordings of lesser quality can be improved by applying modern knowledge and high-quality audio effects. This process is known as audio restoration, usually performed by skilled audio engineers.

In our example, a service provider wants to launch a service for automatic audio restoration. He developed an algorithm combining several audio effects in a way that an audio engineer would do, enabling users to improve the quality of their old digitalized sound carriers on their own. The algorithm produces the best results for file sets that represent a complete sound carrier (one file for each track). Having such a set as input, the service can process the files uniformly, creating a uniform sounding result.

There are two variants of the service interface that are considered for the market launch, specified in UML. In the folder variant (see Figure 1.1), all audio files have to be located in the same folder. The path of that folder is taken as input parameter of the restoreFolder operation. The return value is a download link for an archive file containing the processed audio files.

![Figure 1.1: Folder variant of the restoration interface](image1)

The collection variant (see Figure 1.2) allows users to select audio files individually from their file system: The restoreFiles operation takes a collection of file paths as argument, modeled by a parameter multiplicity of [0..*]. The individual selection of files is preferable because it does not constraint the arrangement of the user’s files.

![Figure 1.2: Collection variant of restoration interface](image2)

The collection variant depicts an example for a specification that can not be transformed to the SSL completely because the SSL is not able to express the multiplicity of the filePath parameter. As a consequence, matchings on that specification produce uncertain results.
1.3 Approach

The first objective of this thesis is to develop a UML-to-SSL transformation enabling the automatic translation of UML specifications (see Section 1.3.1). This transformation serves as a preliminary for the second objective, the measurement of transformation-induced uncertainty (see Section 1.3.2). This measurement aims at helping requesters in assessing the risk of matching result incorrectness: The more uncertainty is measured, the less reliable is the result.

Since the scope of a Bachelor’s thesis is limited, the operation signature aspect of service specification is focussed exclusively. Both the transformation and the uncertainty measurement only consider that service aspect.

1.3.1 UML-to-SSL Transformation

Translating UML signature specifications to the SSL manually is a time consuming task done in reoccurring patterns. It can be avoided by using a model transformation for the automated translation of service specifications. Therefore, a UML-to-SSL model transformation for operation signatures is developed in the scope of this thesis.

Moreover, that transformation depicts a preliminary for uncertainty measurement: When a specification language contains concepts that have no semantic equivalent in the SSL, all information expressed with them is lost in any transformation, resulting in uncertainty. Therefore, uncertainty measurement requires the identification of all such concepts. They are also called information loss causes in the context of this thesis. The development of the transformation includes the identification of parameter multiplicities as the only information loss cause in the transformation of UML signatures.

1.3.2 Uncertainty Measurement

The second objective is to measure the uncertainty induced by the developed transformation. The measurement obtains an uncertainty value that expresses, how uncertain the corresponding matching result is: The higher the uncertainty value, the less reliable is the matching result. Thus, the uncertainty value helps requesters in assessing the risk of having an incorrect matching result.

It is investigated, which measurements are feasible. It is shown, that it is not feasible to obtain the precise probability of matching result incorrectness. Instead, the information loss of a transformation execution is measured. This requires knowledge on where what kind of information is lost. This knowledge is derived from the original model and persisted in an uncertainty report. That report is put on the market along with the SSL specification it refers to. It can be used to reason about the uncertainty of matchings on the referred specification.
1. Introduction

The measurement is performed as a validation of the results returned by the existent signature matcher that is developed in the CRC 901. It is shown that for parameter multiplicities, only a dichotomic measurement value can be derived, i.e., a measurement value that only expresses, if uncertainty occurred or not. However, the uncertainty report allows to prove specific matching results to be wrong, thus correcting them with certainty.

1.4 Overview

The remainder of this thesis is structured as follows: Chapter 2 provides the foundations for this thesis, including service specification, matching, and model transformations. Moreover, it provides an overview over the meta-models defining the UML and SSL languages. Chapter 3 uses the meta-model definitions to develop a UML-to-SSL transformation. In that chapter, parameter multiplicities are identified as cause of information loss in transformations. The measurement of uncertainty induced by the transformation is approached in Chapter 4, where the feasibility of different measurements is investigated, as well. Chapter 5 evaluates the findings of this thesis, followed by Chapter 6 introducing related work. The conclusions are drawn in Chapter 7 where also future work issues are addressed. Appendix A notes properties from the meta-models that are unconsidered in this thesis together with the reasons for their exclusion. Appendix B reconstructs the process that lead to choosing UML signatures for a transformation to the SSL. It considers many alternative specification languages and aspects by checking them against requirements that promise valuable research findings.
2 Fundamentals

The most fundamental topic for uncertainty measurement is service specification (see Section 2.1). It depicts a preliminary for service matching (see Section 2.2), where uncertainty occurs. For these topics, especially the advances of the CRC 901 are considered here. Model transformations are presented as another foundation of this thesis (see Section 2.3).

To develop a UML-to-SSL model transformation, the meta-models defining the languages have to be mentioned in detail. For the meta-class properties that are considered in this thesis, this is done in Sections 2.4 and 2.5. The irrelevant properties and reasons for their irrelevance are presented in Appendices A.1 and A.2.

2.1 Service Specification

In the context of the CRC 901, the Service Specification Language SSL is developed. Since it is used as core matching language, it contains only the concepts most relevant to service specification: The expressive power of a matching language and the degree of efficiency in matching its instances form a language design trade-off [Pla13, APG+14]. Some of the SSL features are inherited from the Palladio Component Model PCM [RBB+11] which is developed for the specification of component-based software systems and for performance predictions on them. For more details on relevant SSL aspects, see Section 2.4.

The Unified Modeling Language UML [Obj11b] is a language broadly applied for software specification in general. It defines terms and notations for most relevant modeling concepts, resulting in a wide scope. For more details on relevant UML aspects, see Section 2.5.

For the signature aspect, the UML and SSL are quite similar: Both model an operation signature as a set of typed input and output parameters. Additionally, potential failure of the operation can be modeled.

Note that existent services and service requests are both specified in a specification language. A service request is simply the specification of a service, of which the existence is not known. Both the UML and the SSL can be used to specify a service request.
2.2 Service Matching

Service matching is the procedure of deciding, if the specification of a provided service can fulfill a service request. The automatic discovery of software services satisfying a request requires an automated matching of service specifications.

Two specifications match to the extent that the provider specification fulfills the requirements imposed by the request. This extent is expressed in the matching result quality. In the CRC 901, gradual values and binary values are used to express that quality, depending on which model elements are matched. Gradual values enable a ranking of results so that the best match can be selected. Binary results simply express if the request is fulfilled or not, i.e., if a quality threshold is exceeded.

One key requirement of service matching is comprehensiveness, i.e., the consideration of different specification aspects, like operation signatures, pre-/postconditions of operations, and more. The matching of operation signatures returns, how well the types of the operations’ parameters match.

Since usually, there are many possibilities for how requested and provided parameters are assigned to each other, signature matching returns results for all possible parameter mappings called graph results. Each graph result represents one input or output parameter mapping and its quality value. The quality of a graph result is defined by the percentage of matching parameters in that mapping. Parameters either match or not, being compared by their types, only. The quality of the respective input or output matching is then defined by the graph result with the highest quality value.

2.3 Model Transformations

Model(-to-model) transformations distinguish between a source language and target language. Source models are instances of the source language. They are translated to target models which are instances of the target language.

Transformations are defined on language level: Concepts of the source language are mapped to semantically equivalent concepts of the target language. Such concepts can be represented by meta-model classes and their properties. For example, both the UML and SSL have an operation class that has a name property. The UML operation class can be mapped to the SSL operation class, mapping its name property to the name property of the SSL class.

Meta-model mappings depict translation rules defining, how a target model can be created from any source model. Executing a transformation means to perform these rules to translate a source model into the target language. For example, for each UML operation, an SSL operation can be created that has the UML operation’s name.
2.4 SSL Meta-Model

The Service Specification Language SSL is designed to be used for efficient matching of service and request descriptions. The SSL syntax is specified with meta-models extending the Palladio Component Model PCM [RBB+11]. The SSL introduces additional features by extending PCM classes.

An excerpt of the combined SSL and PCM models is shown in Figure 2.1. Generalization associations are denoted as white-headed arrows. The names of abstract classes are written in italics.

Figure 2.1: SSL meta-model

The SSL and PCM classes are furtherly explained in the following subsections. These subsections consider every property that is owned by the class in question. The properties are identified by their name, followed by their type, e.g., “entityName : String”. Some of these properties may be inherited from other classes. If so, the superclass that introduces the property is attached in parantheses, e.g., “(from NamedElement)”.

Furthermore, every property has a cardinality. If none is noted, the cardinality is [1], i.e., the property always has exactly one value. The cardinality [*] means that the property can have any number of values, including zero. Cardinality [0..1] means that a property is optional. Moreover, properties exist that can be derived from others. Since they can be derived, they do not need to be considered in a mapping.

Note that every regarded SSL class is a subclass of the PCM Entity [RBB+11], p.97]. Therefore, every model element has a name represented by the property entityName : String (from NamedElement).
2.4.1 Repository

The SSL Repository class is taken from the Palladio Component Model \[RBB^{+11}\], p. 126. With the SSL extensions, a repository can model sets of software services including all aspects relevant to service matching. In addition to the entityName property, the Repository has the following properties:

- **interfaces**: Interface [*]
  
  The interfaces stored in the repository (see Section 2.4.2).

- **datatypes**: DataType [*]
  
  The types that are stored in the repository. In the SSL, all types included in a repository are SimpleOntologyDataTypes (see Section 2.4.5).

- **failureTypes**: FailureType [*]
  
  Types of failure that may occur in the modelled system. In the SSL, all failure types included in a repository are OntologyFailureTypes (see Section 2.4.6).

2.4.2 OperationInterface

The SSL OperationInterface class is taken from the Palladio Component Model. It was introduced the 25th of May, 2010 \[Kar\]. However, only the abstract superclass Interface is present in the PCM document \[RBB^{+11}\], pp. 123ff.]

In the PCM, interface descriptions should suffice to make the specified functionality understandable for callers. In the service matching context this includes, that an interface description should suffice to enable correct matching results. In addition to the entityName property, the OperationInterface has the following properties:

- **signatures**: OperationSignature [*]
  
  The operation signatures that are specified by this interface. They have to be unique. In the SSL, all operation signatures are ExtendedOperationSignatures (see Section 2.4.3).

- **parentInterface**: Interface [*]
  
  Interfaces that are directly extended by this interface.

2.4.3 ExtendedOperationSignature

The SSL signature class ExtendedOperationSignature extends the PCM OperationSignature by adding operation pre- and postconditions. Note that the OperationSignature was introduced the 25th of May, 2010 \[Kar\]. However, only the abstract superclass Signature is present in the PCM document \[RBB^{+11}\], p. 127ff.]. In addition to the entityName property, the ExtendedOperationSignature has the following properties:
2.4 SSL Meta-Model

- **parameters : Parameter [*]** (from OperationSignature)
  The (ordered) list of parameters of this operation. In the SSL, every parameter is of the type ExtendedParameter (see Section 2.4.4). Note that this property was an association to instances of the Variable class until the revision of the 10th of June 2010 [Kar].

- **failureType : FailureType [*]** (from OperationSignature)
  The collection of failure types of this operation. In the SSL, every failure type is modelled with the OntologyFailureType class (see Section 2.4.7).

### 2.4.4 ExtendedParameter

The SSL ExtendedParameter class extends the PCM Parameter by adding the possibility to define optional parameters. The Parameter class is neither documented in the PCM document, nor in the changelog. These properties are taken from the PCM source code:

- **parameterName : String** (from Parameter)
  The name of the parameter. Note that this is not the entityName property from the Entity class (which is not inherited).

- **modifier : ParameterModifier** (from Parameter)
  The direction modifier of the parameter which may be in, out, or inout.

- **dataType : DataType** (from Parameter)
  The type of the parameter. In the SSL, every type is modeled with the SimpleOntologyDataType class (see Section 2.4.5).

- **optional : Boolean**
  If true, this parameter does not have to be set.

Since the SSL does not express parameter multiplicities, all parameters are specified singularly. Thus, the SSL assumes parameter multiplicities to be either [1] or [0..1]. A [0..1] multiplicity is expressed by optional = true.

### 2.4.5 SimpleOntologyDataType

The SSL SimpleOntologyDataType class extends the abstract PCM class DataType [RBB, pp. 118f.] by adding an ontology reference String. In addition to the entityName property, the ExtendedParameter introduces only one property that has to be considered:

- **ontologyReference : String**
  A reference to an ontology concept that corresponds to this type. Ontology references enable ontological service matching.
2.4.6 OntologyFailureType

The SSL OntologyFailureType class extends the PCM FailureType [RBB⁺11, p. 120] by adding an ontology reference String. Because of this, it is quite similar to the SimpleOntologyDataType: The FailureType properties entityName, ontologyReference, and repository are defined identically to the SimpleOntologyDataType properties of these names (see Section 2.4.5).

2.5 UML Meta-Model

The UML [Obj11b] is formally defined using a MOF-based meta-model (OMG MOF: Meta Object Facility [Obj13]). An excerpt of this model is shown in Figure 2.2. The excerpt is focussed on aspects relevant for operation signature specification. Therefore, not every class in the generalization hierarchie is shown. Also, the superclass Element [Obj11b, pp. 63ff.] is not included because it does not have relevant properties.

![UML meta-model](image)

Figure 2.2: UML meta-model, based on [Obj11b]

In the UML, some properties are explicitly derived. They are denoted by a backslash “\”. Derived properties of the source language do not have to be considered, but they can ease mappings. In the following, only those derived properties are regarded that can beneficially be used in the UML-to-SSL transformation developed in Chapter 3.

The UML classes shown in Figure 2.2 are furtherly described in the following subsections. This is done analogously to the class descriptions from Section 2.4.
2.5 UML Meta-Model

2.5.1 NamedElement

Every class that is regarded in this chapter is a subclass of NamedElement \[Obj11b\], pp.99f.]. Therefore, every regarded model element has the following properties:

- **name : String [0..1]**
  The name of the model element.
- **/qualifiedName : String [0..1]**
  The qualified name allows a global identification of elements.
- **ownedComment : Comment [*] (from Element)**
  Every element can have comments attached to it. A comment is specified by it’s textual body \[Obj11b\], pp.56f.]. Since comments do not define any semantics, they are not considered.

2.5.2 Package

Instances of the Package class do not have to be owned by another model element. Therefore, they can be used as top-level elements of UML models. In addition to the properties inherited from NamedElement (see Section 2.5.1), a Package has the following properties:

- **/packagedElement : PackageableElement [*]**
  This collection can contain any element that can be a member of a package.
- **/nestedPackage : Package [*]**
  This subset of packagedElement contains nested packages.
- **/ownedType : Type [*]**
  This subset of packagedElement contains types.

2.5.3 Model

The UML Model \[Obj11b\], pp.625ff.] is a specialization of the Package class (see Section 2.5.2). However, it has a different interpretation: A model is an abstraction of a complete system including all aspects relevant to a certain viewpoint or purpose.

Since the Model class inherits from Package, model instances can be used as top-level elements of UML models, too. The Model class does not introduce any additional properties that have to be considered.

2.5.4 Interface

The UML Interface \[Obj11b\], pp.86ff.] defines a set of publicly visible features. In addition to the properties inherited from NamedElement (see Section 2.5.1), it has the following properties:
2. Fundamentals

- **ownedOperation**: Operation [∗]
  Reference to operations (see Section 2.5.5) that are provided by the interface. Every interface operation has to be public.

- **ownedAttribute**: Property [∗]
  Reference to structural features like attributes and associations. Every interface attribute has to be public.

- **nestedClassifier**: Classifier [∗]
  Classifiers that are defined inside the interface. They have to be public, too.

- **redefinedInterface**: Interface [∗]
  Interfaces can refine each other. This association references all interfaces that are redefined by this interface.

2.5.5 Operation

The UML Operation [Obj11b, pp. 104ff.] specifies how a classifier’s behavior can be invoked and how the classifier behaves on that invocation. In addition to the properties inherited from NamedElement (see Section 2.5.4), an Operation has the following properties:

- **ownedParameter**: Parameter [∗]
  A set of in, out, inout, and return parameters (see Section 2.5.6).

- **raisedException**: Type [∗]
  References types that represent exceptions raised by this operation.

- **interface**: Interface [0..1]
  The Interface (see Section 2.5.4) this operation belongs to. This property can be derived from the opposite association.

2.5.6 Parameter

The UML Parameter [Obj11b, pp. 122f.] specifies an argument passed into or out of an operation invocation. In addition to the properties inherited from NamedElement (see Section 2.5.4), a Parameter has the following properties:

- **direction**: ParameterDirectionKind
  The direction modifier of the parameter, which may be in, out, inout or return (Obj11b, pp. 123f.).

- **defaultValue**: ValueSpecification [0..1]
  The default value is used, when no other value is supplied for this parameter. This property may define a literal for the primitive UML types Null (Obj11b, pp. 91f.), Boolean (Obj11b, pp. 89f.), Integer (Obj11b, pp. 90f.), Real (Obj11b, pp. 92f.), UnlimitedNatural (Obj11b, pp. 94f.), and String (Obj11b, p. 94). Furthermore, instances of complex types may be specified (Obj11b, p. 85).
2.5 UML Meta-Model

- **type** : Type [0..1] (from TypedElement)
  
The type of this parameter (see Section 2.5.7). For parameters, this property is mandatory.

- **lowerValue** : ValueSpecification [0..1] (from MultiplicityElement)
  
  A lower value on the multiplicity interval of this parameter. Because it is constrained to be a non-negative integer value, this thesis will regard the derived property /lower : Integer [0..1] instead of this property.

- **upperValue** : ValueSpecification [0..1] (from MultiplicityElement)
  
  An upper value on the multiplicity interval of this parameter. Because it is constrained to be an integer value, this thesis will regard the derived property /upper : UnlimitedNatural [0..1] instead of this property.

- **isOrdered** : Boolean (from MultiplicityElement)
  
  For a multivalued multiplicity, this property defines, if elements of this parameter are ordered.

- **isUnique** : Boolean (from MultiplicityElement)
  
  For a multivalued multiplicity, this property defines, if elements of this parameter are unique.

2.5.7 Type

The abstract UML class **Type** constrains the range of values a typed element can take [Obj11b, pp. 127f.]. In addition to the properties inherited from **NamedElement** (see Section 2.5.1), there is no property defined. Since this thesis only considers interface operations (which have to be **public** [Obj11b, p. 86]), every type used in such operation has to be publicly visible. Thus, the **visibility** property of **NamedElement** does not have to be considered.

2.5.8 Property

A **Property** [Obj11b, pp. 124ff.] represents a declared state of classifier instances. It may be an attribute or an association of that classifier. Here, we only consider **Interface** (see Section 2.5.4) properties, which have to be **public**. In addition to the properties inherited from **NamedElement** (see Section 2.5.1), the **Property** class has the following attributes:

- **type** : Type [0..1] (from TypedElement)
  
  The Type (see Section 2.5.7) of this Property. If the Property represents an association, this attribute is the type of the association end.

- **defaultValue** : ValueSpecification [0..1]
  
  The default value is used, when no other value is supplied for this property (see also defaultValue of Parameter, Section 2.5.6).
• **isReadOnly**: Boolean  
  Specifies, whether this Property can be written.

• **lowerValue**: ValueSpecification [0..1] (from MultiplicityElement)  
  A lower value on the multiplicity interval of this property. Because it is constrained to be a non-negative integer value, this thesis will regard the derived attribute /lower : Integer [0..1] instead of this attribute.

• **upperValue**: ValueSpecification [0..1] (from MultiplicityElement)  
  An upper value on the multiplicity interval of this property. Because it is constrained to be an integer value, this thesis will regard the derived attribute /upper : UnlimitedNatural [0..1] instead of this attribute.

• **isOrdered**: Boolean (from MultiplicityElement)  
  For a multivalued multiplicity, this attribute defines, if elements of this Property are ordered.

• **isUnique**: Boolean (from MultiplicityElement)  
  For a multivalued multiplicity, this attribute defines, if elements of this Property are unique.
3 A UML-to-SSL Transformation

Model transformations enable the automated translation of models of a source language to models of a target language. Thereby, mappings between semantically equivalent concepts of these languages depict rules for the translation. This chapter develops a model transformation for operation signatures. The source language of that transformation is the UML excerpt introduced in Section 2.5. The target language is the SSL excerpt from Section 2.4.

Based on the transformation definition (see Section 3.1), causes for information loss are identified (see Section 3.2). Since the transformation is defined by mappings between meta-class properties, every property that has no equivalent in the SSL depicts an information loss cause. The identification of information loss causes is a prerequisite for the uncertainty measurement approached in Chapter 4.

3.1 Transformation of Operation Signatures

UML and SSL have a common understanding of operation signatures, what allows for a transformation of that specification aspect. Both languages model an operation signature as a set of typed input and output parameters. Additionally, potential failure of the operation can be modeled as exception or failure type, what is equivalent.

Transforming operation signatures includes the transformation of data type definitions. Since operation parameters are typed, an operation signature is only complete with the referenced data types. Moreover, structural features containing signatures need to be mapped because operations do not depict valid models on their own. The required structural mappings address interfaces and the respective top-level model element of each language.

3.1.1 Top-level Model Elements

The UML top-level element Model (see Section 2.5.3) is chosen to be mapped to the SSL Repository (see Section 2.4.1) because it fits the repository idea: Both SSL repositories and UML models describe all system aspects relevant to a certain purpose. Alternatively, the UML top-level element Package (see Section 2.5.2) could be mapped to the repository, but it is not as well fitting: A single repository may include arbitrary implementation packages.
In the transformation of models to repositories, repository properties are obtained as follows:

- **entityName : String (from NamedElement)**
  The SSL repository name is obtained from the name property of the UML Model.

- **interfaces : Interface[*]**
  This property can be obtained from the ownedType association of the UML Model. The transformation of interfaces is approached in Section 3.1.2.

- **datatypes : DataType[*]**
  This property can be obtained from the ownedType association of the UML Model. The transformation of types is approached in Section 3.1.5.

- **failureTypes : FailureType[*]**
  This property can be obtained from the ownedType association of the UML Model. The transformation of failure types is approached in Section 3.1.6.

The UML model class has additional properties with no SSL equivalent:

- **nestedPackage : Package[*] (from Package)**
  Nested packages can be mapped to the SSL omitting this association. Their nested character can be expressed by using their qualified names.

- **packagedElement : PackageableElement[*] (from Package)**
  This association may contain interfaces, types and exceptions, which are already mapped. However, there are other packageable elements that depict open issues: Constraints and dependencies can be expected to lead to transformation-induced information loss. However, they are not considered in the scope of this thesis.

### 3.1.2 Operation Interfaces

Both UML and SSL have an Interface class of which they have a common interpretation (see Sections 2.5.4 and 2.4.2). The SSL interface properties can be mapped as follows:

- **entityName : String (from NamedElement)**
  The SSL interface name is obtained from the name property of the UML Interface.

- **signatures : Signature[*]**
  This property can be obtained from the ownedOperation association of the UML Interface. The transformation of operation signatures is approached in Section 3.1.3.

- **parentInterface : Interface[*]**
  In the UML, this attribute is specified as redefinedInterface. It can be mapped directly.
The UML Interface has additional properties that have no SSL equivalent:

- **ownedAttribute : Property [**]
  
  Since all features of an interface have to be publicly visible in the UML, these properties can be mapped to operations that access them (“getters” and “setters”). This way, no semantics are lost. The mapping of interface properties is approached in Section 3.1.7.

- **nestedClassifier : Classifier [**]
  
  Nested classifiers can be mapped to the SSL omitting this association. Their nested character can be expressed by using their qualified names.

### 3.1.3 Operation Signatures

UML Operations (see Section 2.5.5) can be mapped to SSL ExtendedOperationSignatures (see Section 2.4.3). SSL operation properties can be mapped as follows:

- **entityName : String (from NamedElement)**
  
  The SSL signature name is obtained from the name property of the UML Operation.

- **parameters : Parameter [**]
  
  This property can be obtained from the ownedParameter association of the UML Operation. The mapping of parameters is approached in Section 3.1.4.

- **failureType : FailureType [**]
  
  This property can be obtained from the raisedException association of the UML Operation. The mapping of failure types is approached in Section 3.1.6.

The UML operation class has no additional properties that have to be considered.

### 3.1.4 Operation Parameters

UML Parameters (see Section 2.5.6) can be mapped to SSL ExtendedParameters (see Section 2.4.4). Their properties can be mapped as follows:

- **parameterName : String (from Parameter)**
  
  The SSL parameter name is obtained from the name property of the UML Parameter.

- **modifier : ParameterModifier (from Parameter)**
  
  This property can be directly obtained from the direction association of the UML Parameter. Since the SSL does not use the return modifier, it has to be mapped to the semantically equivalent out modifier.

- **dataType : DataType (from Parameter)**
  
  This property can be obtained from the type association of the UML Parameter. The transformation of parameter types is approached in Section 3.1.5.
• **optional**: Boolean
  In the UML, optional parameters are expressed with a cardinality having a lower bound of zero, e.g., \([0..1]\). Another way of defining an optional parameter is by a default value. For parameters with a lower bound of zero or with a default value, the optional flag is set to `true`.

The UML parameter class has additional properties that have no SSL equivalent:

• **lower**: Integer \([0..1]\) (from MultiplicityElement)
  The lower bound of the cardinality of this parameter. A zero indicates that the parameter is optional. A one indicates that it is mandatory, i.e., not optional (as long as no default value is defined). Lower bounds greater than one indicate a collection parameter. These can not be expressed properly. The measurement of uncertainty induced by cardinalities is approached in Chapter 4.

• **upper**: UnlimitedNatural \([0..1]\) (from MultiplicityElement)
  An upper bound greater than one indicates a parameter collection. The measurement of uncertainty induced by cardinalities is approached in Chapter 4.

• **isOrdered**: Boolean (from MultiplicityElement)
  Only relevant for parameter collections (which can not be expressed anyway).

• **isUnique**: Boolean (from MultiplicityElement)
  Only relevant for parameter collections (which can not be expressed anyway).

### 3.1.5 Parameter Types

The UML class `Type` (see Section 2.5.7) can be mapped to the SSL class `SimpleOntologyDataType` (see Section 2.4.5). Note that the UML also contains a class named `DataType`. However, this class is defined as a type, whose instances are identified only by their value [Obj11b, p. 60]. This definition does not fit the SSL idea of data types. SSL types can have the following properties set:

• **entityName**: String (from NamedElement)
  The SSL type name is obtained from the `name` property of the UML Type.

• **ontologyReference**: String
  Types used in UML specifications do not include references to concepts of an ontology. However, these references are needed to enable ontological service matching. UML types have to be attributed with ontology references.

There are no additional properties of UML types that have to be considered.

In order to attribute UML types with ontology references, some alternatives exist. First of all, an external file could make this attribution. The advantage of this approach is that the UML model itself does not have to be changed. However, a syntax for such files would have to be defined.
Another approach is to integrate the attribution into the UML model, e.g., by UML comments [11b, p.56f.] that are attached to types. A transformation can identify ontology reference comments by checking for a dedicated keyword, e.g., “ontologyRef”. Ontology reference comments can even help readers of the model in understanding it more deeply, what perfectly suits the usual purpose of UML comments.

An alternative way of including ontology references into UML models is to make use of the Artifact class [11b, pp.203ff.]. Artifacts can specify any physical piece of information, including ontologies. The class property fileName : String could directly reference ontology concepts. However, the Artifact class has several other properties depicting a huge overhead for simply specifying ontology references.

Since comments are light-weight, common, easy to use and helpful for readers, they are chosen to provide ontology references for types. To successfully transform a UML model, every type used for parameter specification is required to have an ontology reference comment attached to it.

### 3.1.6 Exceptions and Failure Types

In the UML, failures can be modelled by exceptions. These exceptions can be any UML Type. Since the SSL OntologyFailureType has the same properties as the SimpleOntologyDataType, the mapping of failure types can be done analogously to the mapping of parameter types presented in Section 3.1.5.

### 3.1.7 Interface Properties

In the SSL, interfaces can not own properties, i.e., attributes and associations (see Section 2.4.2). The UML does support this feature using the Property class (see Section 2.5.8). For Interfaces, properties are constrained to be public. Therefore, they can be transformed to accessor operations, i.e., “getters” and “setters”. An accessor ExtendedOperationSignature (see Section 2.4.3) can be created as follows:

- **entityName : String (from NamedElement)**
  
  The signature name is obtained from the Property name preceded by a “get” or “set” prefix.

- **parameters : Parameter [***]**
  
  Each accessor will have one parameter that has the Property name. The direction modifier of this parameter depends on the accessor kind. The type is obtained from the type association of the Property. The parameter is never optional.

- **failureType : FailureType [***]**
  
  For accessors, no failure types will be specified.
The UML Property class has additional properties that have no SSL equivalent:

- **defaultValue : ValueSpecification [0..1]**
  No operation has to be called. Therefore, a default value of a Property does not have any implications on the optional flag of the accessor parameter (unlike for general operation parameters, see Section 3.1.4). Therefore, the default value is not considered for interface properties.

- **isReadOnly : Boolean**
  If this property is true, no “setter” operation is created.

- **lower : Integer [0..1] (from MultiplicityElement)**
  The lower bound of the cardinality of this Property. Lower bounds greater than one indicate a collection Property. These can not be expressed properly. The measurement of uncertainty induced by cardinalities is approached in Chapter 4.

- **upper : UnlimitedNatural [0..1] (from MultiplicityElement)**
  An upper bound greater than one indicates a Property collection. The measurement of uncertainty induced by cardinalities is approached in Chapter 4.

- **isOrdered : Boolean (from MultiplicityElement)**
  Only relevant for Property collections (which can not be expressed anyway).

- **isUnique : Boolean (from MultiplicityElement)**
  Only relevant for Property collections (which can not be expressed anyway).

### 3.2 Information Loss Cause: Parameter Multiplicities

The transformation developed in Section 3.1 identifies all UML meta-class properties that have no semantic equivalent in the SSL. These properties are lower (or lowerValue), upper (or upperValue) from the UML meta-class MultiplicityElement. In addition, the properties isOrdered and isUnique of that class can not be expressed: They require the specification of a multiplicity with an upper bound greater than one.

These properties can be grouped into one general information loss cause that is parameter multiplicities. Since interface properties are mapped to accessor operations, property multiplicities belong to that information loss cause, as well. Thus, parameter multiplicities is the only information loss cause in the UML-to-SSL signature transformation.

In the running example (see Section 1.2), the collection variant of the restoreFiles operation makes use of a parameter multiplicity. As a consequence, the transformation result (see Figure 3.1) suffers from information loss: The multiplicity of the filePath parameter can not be expressed in the SSL.
3.2 Information Loss Cause: Parameter Multiplicities

Figure 3.1: Transformation result of the collection variant

Note that lower and upper bounds $\leq 1$ do not induce information loss: They specify, if a parameter is optional (lower bound of zero) or mandatory, i.e., not optional (lower bound of one). This can be expressed by the SSL using the `optional` flag. When properties are mapped to accessors, lower and upper bounds $\leq 1$ have no semantic implications because calling an operation is always optional.
4 Measuring Transformation-induced Uncertainty

Recall that a signature matching result consists from a set of graph results (see Section 2.2). The input and output matching qualities are defined by the highest qualities among the input and output graph results. This depicts an implicit graph result choice: The graph result with the highest quality is assumed to contain the best mapping. However, this assumption does not hold for uncertain matching results because their quality values may be incorrect. Therefore, only the measurement of transformation-induced uncertainty in graph results is approached. The quality value aggregation of the signature matching result is part of future work (see Section 7.2.1).

Measuring transformation-induced uncertainty requires preliminary investigation on what measures to apply. Obtaining the precise probability of having incorrect matching results promises to be a highly beneficial measurement because requesters can directly assess the risk of having these results. However, it becomes apparent that this approach is not feasible (see Section 4.1.1). Measuring the information loss inherent to transformations is a feasible alternative (see Section 4.1.2), of which the credibility of the particular matching result can be derived (see Section 4.2). The lack of credibility is used as uncertainty representation.

Section 4.3 proposes an uncertainty measurement process obtaining uncertainty values for graph results. Moreover, it is able to correct these results in specific cases. In the end of this chapter, some implementation notes are given (see Section 4.4).

4.1 Feasibility

Uncertainty measurements aim at helping requesters in assessing the risk of receiving incorrect matching results. Preferably, the precise probability of having an incorrect result is directly provided. Unfortunately, it becomes apparent that obtaining that probability from SSL models is not feasible (see Section 4.1.1).
Recall that information loss in transformations is the reason for potentially having incorrect matching results. Thus, measuring the information loss promises to give a notion of the probability of matching result incorrectness: The more information is lost, the more likely the matching result is incorrect. Measuring the information loss inherent to a transformation is presented as a feasible approach for the measurement of transformation-induced uncertainty (see Section 4.1.2).

4.1.1 Obtaining the Probability of Incorrect Matching Results

The probability of having an incorrect matching result depends on the matching result itself and on the probability that the compared specifications actually match. The matching result is based on the SSL models generated from the original specifications. To the contrary, the probability of an actual match is based on reasoning about the original specifications. A matching result and the actual match can differ because of information loss induced by transformations.

An altered version of the running example illustrates, how the probability for an actual match is defined by execution probabilities. In that altered version, the size of the file collection is limited to five (see Figure 4.1). Such a limit may be applied because of a market strategy: A limited version is offered for free, while the unlimited version is available for a small price.

Figure 4.1: Limited collection variant of the restoration interface

The limited variant is matched with a request for audio restoration (see Figure 4.2). For that request, the limited variant is a perfect match, except that the requested input set has an unlimited multiplicity. The limited variant fulfills the request whenever five or less inputs are handed over in an execution. However, if more than five inputs are provided, the limit of the service is exceeded. In that case, the service does not match. More generally, the probability that the specifications match is defined by the probability that five or less input paths are handed over in a service call. This is an execution probability. Let $P(5 \text{ or less input paths}) = P(\text{specifications match}) := 70\%$ as exemplary value.

Figure 4.2: Collection request for audio restoration
4.1 Feasibility

As noted before, the SSL is not able to express parameter multiplicities. As a consequence, SSL matchers regard the limited variant as a perfect match for the request (see the transformation results in Figure 4.3). However, since the actual matching probability is only 70%, there is a 30% probability that this matching result is incorrect for a service call. Unfortunately, these probabilities are not obtainable from SSL models: Since parameter multiplicities can not be expressed, no probability on such multiplicity can be expressed either. Even if provided in the original model, the value for $P(5 \text{ or less input paths})$ gets lost in a transformation.

![Figure 4.3: Transformation results of limited variant and collection request](image)

As we can see, the probability of having an incorrect matching result is not solely introduced by model transformations but also by execution probabilities. Since these probabilities can get lost in a transformation, it is infeasible to obtain the probability of matching result incorrectness from SSL specifications. As a consequence, this thesis approaches information loss measurement rather than obtaining probabilities.

4.1.2 Information Loss Measurement

In contrast to obtaining the probability of matching result incorrectness, measuring the information loss inherent to a transformation is feasible: When comparing the transformation result and the original specification, every lost piece of information can be identified. Information is obviously lost, if it is contained in the source model but not in the target model.

With the identification of information loss causes (see Chapter 3), information loss measurement can even be simplified: Whenever an information loss cause is utilized in a source model, the information expressed with it is lost. With that knowledge, a transformation does not even have to be performed to measure its information loss for a source model; only the source model itself has to be checked. For an example on the detection of information loss occurrences, see Section 4.3.1, where this task is approached in detail.
Since obtaining the precise probability of having incorrect matching results is infeasible, this thesis will approach information loss measurement. Based on the information loss inherent to the transformations of the matched model elements, the credibility of matching results is validated.

4.2 Representation of Uncertainty

In an environment without uncertainty, it is clear if a matching result is correct or incorrect. This dichotomic distinction can be modeled by a crisp set of correct results to which a matching result belongs or not. Existent matchers implicitly use such a crisp set by returning exactly those results that are included in it.

Figure 4.4 shows the folder variant and a request for the restoration of audio folders. The request also includes an output parameter for the title of the processed sound carrier. Obviously, the folder variant is a perfect match for the restoration task. Unfortunately, it can not return the title. The correct matching result for the output parameter matching is one half because one of two output parameter matches. Thus, any other matching result, e.g., zero, is incorrect. Such a value would not be returned by a correctly working matcher.

<table>
<thead>
<tr>
<th>MusicRestaurationInterface_FolderVariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>restoreFolder(IN folderPath: String, OUT out: String)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FolderAndTitleRequest</th>
</tr>
</thead>
<tbody>
<tr>
<td>restore(IN path: String, OUT out: String, OUT title: String)</td>
</tr>
</tbody>
</table>

Figure 4.4: Request for the restoration of audio folders

To the contrary, it can not always be said if a matching result is correct or not when the matching environment is (potentially) uncertain. Consequently, a crisp set does not suffice for the definition of correct results. The uncertainty inherent to a matching has to be made explicit by a different representation.

4.2.1 The Fuzzy Set of Correct Matching Results

Since the correctness of a matching result is not always certain, the set of correct matching results has to be defined as a fuzzy set. Just like in the crisp set, a membership value of one defines a (totally) trustworthy matching result. For a membership value of zero, it is certain that the matching result is incorrect. Membership values between zero and one model uncertainty about the correctness of the result. This uncertainty is highest, i.e., the correctness of the result is completely unknown, when the membership value to the set of correct matching results is \( \frac{1}{2} \).
4.2 Representation of Uncertainty

Analogously, the uncertainty of a matching result can be stated: Membership values of 0 or 1 define complete certainty about the set membership of the result; it is either certainly correct or certainly incorrect. Such definite certainty is modeled by an uncertainty value of zero. An uncertainty value of one is equivalent to a membership to the set of correct matching results of 1/2 (see Figure 4.5).

![Figure 4.5: Interrelation between set membership and uncertainty value](image)

Imagine a matching between the folder variant of the restoration service and the collection request from Section 4.1.1 (see Figure 4.6). The request specifies an unlimited cardinality on the path parameter, which is why the folder variant (cardinality of 1) does not fulfill the request (let the example request specify a set of folder paths as inputs, here). The path parameter is marked red to remind of the information loss it suffers from.

![Figure 4.6: Matching between folder variant and collection request](image)

Since the output parameters of both operations specify a download link to the result of the particular operation, they depict a perfect match. Since neither of the outputs suffers from information loss, the matching on these parameters is totally credible, i.e., it has a membership value of 1 to the fuzzy set of correct matching results and an uncertainty value of 0.

Since the input parameters of both operations specify a folder path, they depict a perfect match as well. However, the path parameter of the request suffers from information loss, which is why no credible statement on the correctness of this matching result can be made. Actually, no credible statement can be made at all. Therefore, the matching of the input parameters produces a result that an uncertainty value of 1. This is equivalent to a membership value of 1/2 to the set of correct matching results.
4. Measuring Transformation-induced Uncertainty

4.2.2 Measurement Representation

As already noted and applied in Section 4.2.1, there are two possibilities to represent uncertainty measurements: a) constituting the membership values of matching results to the fuzzy set of correct results and b) obtaining the uncertainty value of that membership.

Actually, both representations carry the same information content: Returning a matching result is only reasonable if that result is correct at least at some kind of certainty, i.e., if it has a membership of at least $\frac{1}{2}$ to the set of correct matching results (membership values of less that $\frac{1}{2}$ model certainty about result incorrectness; see Figure 4.5). Therefore, when the membership value of returned matching results is mapped to the unity interval, we have the certainty value of the membership. Analogously, when the certainty value of a returned result is mapped to the $[\frac{1}{2}..1]$ interval, the membership value is obtained.

So far, the choice of representation is arbitrary. However, membership representation is prone to misinterpretation: For example, in a signature matching, users may misinterpret a membership value of $\frac{1}{2}$ as a statement about the fraction of fitting parameters. Or, they may misinterpret that value as the probability for an actual match. Therefore, this thesis will represent uncertainty measurements as the uncertainty about the membership to the fuzzy set of correct matching results. This representation is not prone to the assumed misinterpretations.

4.2.3 Aggregation of Uncertainty

Let a matching result $M$ consist of a set of partial matching results $M_{part} \in M$. For example, a matching of operation signatures consists of a set of parameter matchings. The quality value of $M$ obviously is an aggregation of the quality values of all $M_{part}$.

When there is an uncertain $M_{part}$, uncertainty of $M$ is induced: If this $M_{part}$ is incorrect, then $M$ incorrect, as well. Therefore, the certainty about the correctness of $M$ can not be greater than the certainty about the correctness of $M_{part}$. Consequently, the certainty about the correctness of any aggregated result is defined by the certainty about its least certain partial result. Let $u(M), u(M_{part}) \in [0..1]$ be the uncertainty values of $M$ and $M_{part}$. Then:

$$u(M) = \max_{M_{part} \in M} u(M_{part}) \quad (4.1)$$

In the example from Section 4.2.1, we can observe this property for the matching of operation signatures: Since the input parameter matching has an uncertainty of 1, no certain statement can be made on the correctness of the signature matching result (at all). Thus, the uncertainty of the signature matching is 1, as well.
4.2 Representation of Uncertainty

4.2.4 Sidemark: Uncertainty Induced by Multiplicities

In general, an uncertainty function may take any value in $[0..1]$. For uncertainty induced by lost multiplicities, this is not the case. Recall the parameter matching from Section 4.2.1, which produced the uncertainty values 0 and 1. For multiplicities, there is no other case, i.e., no uncertainty value in between: When a multiplicity is lost, it is lost entirely. A matching on this multiplicity can only be (completely) certain or (completely) uncertain. Consequently, graph results are either completely certain or completely uncertain (following Section 4.2.3).

Note that this remark considers parameter multiplicities only. For other information loss causes, it is not clear if uncertainty values between 0 and 1 are reasonable. This thesis will continue to represent the uncertainty of matching results as real values in $[0..1]$ to be compatible with other uncertainty measurements that may have reasonable values between 0 and 1.

Another fact worth noting is that lost multiplicities can lead to false positives because they have to match in addition to the parameter type. If the type matching returns a negative result (“does not match”) there is nothing uncertain in that result, no matter what the multiplicity is. Only when the type matching returns a positive result, the uncertainty about multiplicities has to be considered. Consequently, when a multiplicities are lost, the quality of the correct matching result is always only at most the quality of the returned result.
4.3 The Measurement Process

This section proposes an uncertainty measurement process obtaining uncertainty values from sets of information loss occurrences relevant to the matched model elements. That process includes three steps which are the detection of information loss occurrences (see Section 4.3.1), the collection of those occurrences that are relevant to the matching (see Section 4.3.2), and the validation of the matching result based on the prior findings (see Section 4.3.3). Figure 4.7 gives an overview over the measurement process.

![Figure 4.7: The uncertainty measurement process](image)

On top of the process overview, the names of the process steps are printed. Under each name, a symbol represents the respective step metaphorically. The sections, in which the process steps are approached, are noted on the bottom of the figure. Between the process steps, artifacts are shown that are passed from one step to another. Green artifacts (source models and matching result) are taken from other procedures that do not consider uncertainty, i.e., service specification and matching. Red artifacts are obtained from the uncertainty measurement process.

The uncertainty report is the primary artifact of uncertainty measurement. It contains all occurrences of transformation-induced information loss. The occurrence collection step can use that report to select those occurrences that have an impact on the matching result correctness. The set of these relevant occurrences is passed to the match validation step. The validation step (and thus the overall process) returns a value that represents, how uncertain the correctness of the matching result is. Moreover, it can even correct the matching result in some cases.

Note the large gap between the first two steps: Between them, model transformations and a matching are performed. The first and the last two steps usually are performed in different points of time and in different places. On a world-wide dynamic market, the matching and the last two measurement steps will be performed on demand and as an on-market process. The first process step and the transformation of the model are preliminaries for that.
4.3 The Measurement Process

4.3.1 Occurrence Detection

The occurrence detection task retrieves source model elements that can not fully be transformed to the target language. When transformed, these elements suffer from information loss. Therefore, this task detects future occurrences of information loss in the source model already. In the collection variant of the running example (see Section 1.2), the filePath parameter is a model element suffering from information loss because its multiplicity can not be expressed in the SSL (see Figure 4.8).

![Recall of collection variant and transformation result](image)

Figure 4.8: Recall of collection variant and transformation result

When the potential causes for information loss are known (achieved by Chapter 3), the detection of future information loss occurrences is simple: Whenever an information loss cause is utilized, the semantics expressed with it can not be transformed. Therefore, it only has to be checked, which model elements utilize information loss causes. The information loss inherent to the running example can easily be detected because parameter multiplicities are a known cause of information loss.

4.3.1.1 Information Loss Annotations

The results of the occurrence detection can be presented as annotations to model elements. These annotations can include the respective information loss cause, what enables the individual measurement of different causes, e.g., by different metrics. The cause of an information loss occurrence is always known because potential causes of information loss are explicitly checked.

![Collection variant with uncertainty annotation](image)

Figure 4.9: Collection variant with uncertainty annotation

Annotations may refer either to the source model or to the target model. Since the subsequent process steps can not access the source model, they require target model annotations. Therefore, this thesis will annotate the target model. Note that source model annotations are required by an alternative uncertainty measurement use case not regarded in this thesis (see Section 7.2.4).
4.3.1.2 Annotating the Inclusion of SSL Assumptions

Not every information loss occurrence introduces the possibility of having incorrect matching results: If the information lost in the transformation of a provided element includes the SSL assumptions made on that information, then that provided element can be matched reliably even though it suffers from information loss.

Imagine a matching of the collection variant of the restoration service and a request specifying the restoration of a single file (see Figure 4.10). Recall that the `filePath` parameter of the collection variant has a multiplicity of `[0..*]` in the original specification (see Section 1.2). Since both interfaces specify audio restoration, the matching result is positive (“perfect match”).

![Figure 4.10: Information loss not inducing uncertainty](image)

Obviously, the matching is performed on an information loss occurrence. Still, the matching result is correct: The SSL assumes parameter multiplicities to be either `[1]` or `[0..1]` (see Section 2.4.4). This assumption holds for the requested `path` parameter because that parameter does not suffer from information loss. Consequently, the provided `filePath` parameter fulfills the requirements (implicitly) imposed by `path`. The information lost in the `filePath` transformation does not restrain the parameters from matching because it includes the SSL assumption made on that information.

Since the occurrence detection step has access to the source model, it can validate, which information loss occurrence includes the SSL assumptions made on it. The information loss annotations are extended to include the findings by a flag indicating assumption inclusion. Let this flag simply be called *assumptions*, taking the values “included” and “else”, which defines the other case. Figure 4.11 shows an extended annotation attached to the collection variant.

![Figure 4.11: Information loss annotation with assumptions flag](image)
4.3.1.3 Annotating the Disjunction of SSL Assumptions

Think of a case related to assumption inclusion: When the information lost in the transformation of an element and the SSL assumptions made on that information are disjunct, then that element will match to no element that does not suffer from information loss: Since the element not suffering imposes the SSL assumptions, the disjunction (certainly) restrains the elements from matching.

Figure 4.12 shows a request for the restoration of complete discographies. The request is customized to the specific situation of the requester: He has created folders for individual sound carriers, of which sets of three to eight folders form a discography. The request specifies that one of these sets is handed over in a service call.

Figure 4.12: Custom restoration request for discographies

Let the discography request be matched with the folder variant of the restoration service (see Section 1.2). Since the folder variant does not suffer from information loss, the SSL assumption on parameter multiplicities holds for every parameter of that service. Since the multiplicity of the discography request’s folderPath parameter is lost in a transformation, the matcher returns a positive matching result (“perfect match”; see Figure 4.13).

Figure 4.13: Folder variant and transformation result of discography request

However, the specifications do not match at all because the multiplicities imply that there is no service call for which the request actually matches the provided service. Since the multiplicity of the requested input parameter is disjunct to the SSL assumption made on multiplicities, the requested parameter does not match any parameter that does not suffer from information loss.

Unfortunately, a matcher that does not consider uncertainty has no notion of assumption disjunction. Thus, the returned results may be wrong (with certainty). Fortunately, knowledge about assumption disjunction can be used to correct such results. Since inclusion and disjunction can not hold both at the same time, the assumptions flag is extended by a third value “disjunct”.

33
4. Measuring Transformation-induced Uncertainty

Note that for multiplicities, the value "else" can never be taken: Since the optional property of parameters is preserved in a transformation, only one of the assumptions on multiplicities (either [1] or [0..1]) has to be considered. The considered assumption then is either included in the source parameter multiplicity, or disjuncts that multiplicity.

4.3.2 Occurrence Collection

The occurrence collection task aims at gathering information on the uncertainty inherent to specific matching results, e.g., graph results. Therefore, it has to be performed after the matching, when the matching result is known. The information loss occurrences affecting the matched elements define the uncertainty inherent to the matching result.

A model element is affected by information loss, if information on itself or on one of its owned elements is lost. In the collection variant of the running example, matchings on the restoreFiles operation are affected by information loss because the operation owns a model element suffering from information loss, which is the filePath parameter. As a consequence, the whole operation can not always be matched reliably.

In an extended variant of the restoration interface, another operation findRestorations is offered (see Figure 4.14). This operation recognizes the songs of the input path set and finds links to professionally restored versions of the sound carrier available in online shops like Amazon. Obviously, the findRestorations operation owns two occurrences of information loss, which are the multiplicities of both the input and output parameter. Figure 4.14 shows the annotations attached to the target model in a compact form.

Let the restoreFiles operation of the extended variant be matched with the restore operation of the exemplary request recalled in Figure 4.15 (see Section 4.1.1). Obviously, a selection on information loss annotations has to be performed: When the restoreFiles operation is matched, the findRestorations operation should not be considered. Thus, the annotations on that operation have to be filtered out. For the measurement of uncertainty inherent to the matching, only the annotations on the restoreFiles and restore operations are relevant. They are an exemplary result of the occurrence collection step.

Figure 4.14: Extended variant of the restoration interface

MusicRestaurationInterface_ExtendedVariant

: String, OUT out: String)

restoreFiles(IN filePath
: String)

findRestorations(IN filePath: String, OUT out
...
......

34
4.3 The Measurement Process

**Figure 4.15: Annotated collection request**

In general, the occurrence collection returns all occurrences that affect any matched element. Not all of these occurrences may induce uncertainty because of assumption inclusion (see Section 4.3.1.2). Filtering out occurrences not inducing uncertainty is done in the *match validation* step to which the set of relevant occurrences is passed. All the information required to obtain that set is included in the uncertainty report: From that report, every relevant annotation can be selected.

### 4.3.3 Match Validation

The set of information loss occurrences relevant to a matching defines the uncertainty of the matching result. However, returning this set has no informative value for requesters: They usually have neither the time nor the expertise to interpret such a set correctly. Therefore, as proposed in Section 4.2, the uncertainty measurement result is presented as an uncertainty value. An uncertainty value defines, how uncertain the correctness of the validated matching result is. Matching results with high uncertainty values, i.e., values close to one, are not credible.

The set returned by the *occurrence collection* step contains all information loss occurrences that affect the matched elements. However, not all of these occurrences have to induce uncertainty (see Section 4.3.1.2). Based on the role of each matched element, i.e., requested or provided role, and on the *assumptions* value of each information loss annotation, it has to be determined, which occurrences have an actual influence on the credibility of the matching result. Moreover, the matching result may have to be corrected when the annotations prove it to be wrong (see Section 4.3.1.3). Note that for parameters, the role depends on the parameter direction: While provided outputs are provided elements and requested outputs are requested elements, it is the other way round for inputs. Requested inputs are provided elements and provided inputs are requested elements because requested inputs are offered by the requester and provided inputs need to be set to make the service work.

The matching result and the set of relevant occurrences contain all the information required for match validation. This last process step returns a (potentially) corrected matching result and uncertainty value of that result.
4.4 Implementation Notes

The UML-to-SSL transformation and the measurement process are implemented as a set of Eclipse plugins. These plugins extend the Service Specification Environment SSE, an Eclipse-based framework for the specification of software services and service matching. The SSE is developed in the CRC 901 along with the SSL.

Section 4.4.1 describes the plugin realization in more detail. In Section 4.4.2, the usage procedure is introduced similar to the help contents provided with the implementation.

4.4.1 Approach

Figure 4.16 shows the plugins developed for this thesis and a subset of their dependencies in a component diagram. Plugins with the de.upb.crc901.sse prefix are part of the SSE project developed in the CRC 901. The SSL is developed in the structure plugin with that prefix. The de.uka.ipd.sdq.pcm plugin defines the PCM. The Eclipse plugin dependencies contain UML and QVTo (see Section 4.4.1.2) implementations.

![Figure 4.16: The implementation architecture](image)

4.4.1.1 Uncertainty Report

The uncertainty report is realized as Eclipse plugin de.upb.babunse.uncertaintyreport using the Eclipse Modeling Framework EMF. This framework allows to specify meta-models, called Ecore models, in the form of class diagrams.
From these diagrams, the model code can be generated. Moreover, EMF provides automatic generation of serializers and parsers, what comes in handy, when instances of an Ecore model need to be persisted in files. EMF is chosen for the realization of the uncertainty report because it accelerates development and Ecore models can be used as input and output models of the QVTo interpreter (see Section 4.4.2). Moreover, EMF is already used to specify the Eclipse UML realization, the PCM and the SSL, which is why fellow developers can understand and adapt the models quickly.

The Ecore model of the uncertainty report package is shown in Figure 4.17, where the classes are placed on the left side and the enums are placed on the right side. The combination of UncertaintyReportEntry and InformationLossAnnotation results in the generation of a map type (note the “key” attribute and the “value” association). This type maps the IDs of annotated model elements to a set of information loss annotations attached to them. The operations getAnnotations (of an element) and getAnnotation (of an element with a particular information loss cause) use that map for efficiency. Note that the getAnnotations method overloads the generated getter of the annotations association with an element ID parameter.

![Ecore model of the uncertainty report package](image)

Figure 4.17: The uncertainty report Ecore package

In addition to the model package, there is a utility class SslUtil that creates IDs for operation parameters. This is required because the ExtendedParameter class (see Section 2.4.4) does not have an ID attribute. Such an attribute is needed to attach annotations to parameters unambiguously. For more information, take a look at the JavaDoc documentation of the plugin which can be found on the CD annexed to this thesis.
4. Measuring Transformation-induced Uncertainty

4.4.1.2 Transformation

The UML-to-SSL transformation developed in Chapter 3 is placed in the `de.upb.babunse.transformation` plugin, where it is realized in the QVT Operational Mappings (QVTo) transformational language [Oh11a, pp. 63 – 144]. This imperative language is designed for the development of model-to-model transformations, which is why it is preferable over general-purpose languages, like Java, when a transformation has to be implemented. Since Eclipse UML models, SSL models, and uncertainty reports are instances of Ecore models, they can provide the in-outputs of the QVTo interpreter, what makes QVTo the perfect choice in the context of this thesis. The creation of the uncertainty report (see Section 4.3.1) can be done in QVTo, as well.

The transformation comes with a second plugin `de.upb.babunse.transformation.ui` that defines the user interface for the UML-to-SSL transformation. It adds a transformation wizard to the SSE that guides the user through the transformation configuration (see Section 4.4.2). When the transformation is configured, it is executed and the target models are created.

4.4.1.3 Matching

For a matching of operation signatures that is aware of uncertainty, the default ontological signature matcher is encapsulated by a validated ontological signature matcher. The validated matcher calls the inner default matcher and validates its results as shown in Section 4.3.3. The validated matcher is placed in the `de.upb.babunse.matching` plugin.

The validation is realized in two steps: Graph results are validated by a graph result validator that calls a parameter match validator and aggregates its results. The occurrence collection step (see Section 4.3.2) is performed using the `getAnnotations` and `getAnnotation` methods of the uncertainty report (see Section 4.4.1).

The matching comes with a user interface plugin `de.upb.babunse.matching.ui` that integrates the validated matcher into the matching wizard provided by the SSE (see Section 4.4.2). The configuration of that matcher does not offer to specify matching quality thresholds, as the configuration of the default matcher does. This is because the requester can not only make his choice based on matching quality, but also on uncertainty values. Therefore, thresholds of 0 are chosen for the configuration of the inner matcher. Thus, all matching results are returned, regardless of their quality. However, the configuration of the validated matcher offers an alternative aggregation method for uncertainty values: This method takes the percentage of uncertain parameter matches as uncertainty value for graph results (instead of the maximum aggregation presented in Section 4.2.3). The alternative aggregation is discouraged for end-users, but may yield insights for uncertainty research.
4.4 Implementation Notes

4.4.2 User Guide

The plugins developed in the scope of this thesis are designed to seamlessly integrate into the SSE framework and its existing user interfaces. The following subsections provide step-by-step guides to the new functionality for those who are not familiar with the SSE: In Section 4.4.2.1, the new transformation functionality is introduced. Section 4.4.2.2 introduces the new validated signature matcher.

4.4.2.1 UML-to-SSL Signature Transformation

The UML-to-SSL transformation developed in Chapter 3 can be found in the “CRC 901 SSE” menu of the Eclipse main window (see Figure 4.18). A transformation wizard enables users to configure the transformation (see Figure 4.19). The transformation results, i.e., the SSL specification and the uncertainty report, are created under the file path of the input model, but with different file extensions. If SSL model or uncertainty report are already present, they are updated.

Preliminaries: Make sure to have a UML model inside your workspace.

To execute the UML-to-SSL transformation, a .uml file containing a UML Model instance as top-level element has to be inside the workspace. It will serve as source model of the transformation. Such file can be created using the Eclipse UML editor or any other editor able to export to the Eclipse UML file format, e.g., MagicDraw. Consider that each parameter type has to have an ontology reference comment attached to it (see Section 3.1.5). For now, you may use one of the examples from the de.upb.babunse.examples project.

Step 1: Open the transformation wizard (see Figure 4.18).

![Figure 4.18: The transformation menu](image)
Step 2: Configure and execute the transformation (see Figure 4.19).

When a .uml file is selected in the Project Explorer, the path of that file is inserted in the wizard’s UML Model field automatically. You can choose, if an uncertainty report is created along with the SSL model.

![Figure 4.19: The transformation wizard](image)

4.4.2.2 Validated Signature Matching

For the validated matching of signatures, the existing SSE matching wizard is used, which is located in the “CRC 901 SSE” menu (see Figure 4.20). The configuration procedure is that of the existing SSE signature matcher, except that uncertainty reports have to be selected for the requester and provider repositories (see Figure 4.23). The validated matching results are presented in the existing Matching Results view, where the uncertainty values are part of the result message (see Figure 4.24).

**Preliminaries:** Make sure to have an SSE repository and a corresponding uncertainty report inside your workspace.

The matcher takes .sse_repo and .uncertaintyreport files as inputs. You can select the provider and requester signatures by one by two repositories. The repository and uncertainty report files are created using the UML-to-SSL transformation guided in Section 4.4.2.1.
4.4 Implementation Notes

Step 1: Open the SSE matching wizard (see Figure 4.20).

![Figure 4.20: The matching menu](image)

Step 2: Click the “Add New Matching Step” button and configure the matcher. When the “Signature Matching” type is selected, you can select the validated signature matcher. You can disable the matching of inputs or outputs and you can choose an alternative aggregation for graph result uncertainty values. This aggregation takes the percentage of uncertain parameter matches as uncertainty values, instead of the max aggregation proposed in Section 4.2.3.

![Figure 4.21: The matcher configuration dialog](image)
Step 3: Select the matched signatures (see Figure 4.22).

Figure 4.22: Signature selection

Step 4: Select the corresponding uncertainty reports and execute the matching (see Figure 4.23).

Figure 4.23: Uncertainty report selection
4.4 Implementation Notes

**Step 5:** Inspect the validated matching results (see Figure 4.24).

The matching results are presented in the existing Matching Results view. You can expand matching results to inspect their child matchings.

![Matching Results](image)

Figure 4.24: The Matching Results view
5 Evaluation and Discussion

Research findings can be evaluated from several viewpoints. From [10016], three viewpoints are identified that are of special importance to this thesis.

In the beginning, the (internal) validity of the measurement is evaluated (see Section 5.1). This is done by answering the question “Are other sources of uncertainty excluded from the measurements?” If another uncertainty source, e.g., request imprecision, was reflected in the measurements, they would not be valid measurements on transformation-induced uncertainty in particular. It is shown that sources other than transformation-induced information loss are successfully excluded from the measurements.

Another evaluation viewpoint addresses the measurement outcome (see Section 5.2). The central question of that viewpoint is “Is the requester’s service choice improved by uncertainty measurement?” The extent, to which service choices are improved, defines the practical value of the measurements. The question is answered by a comparison between the expected service choices with and without uncertainty values attached to the matching results. It is shown that the requester benefits from uncertainty measurements.

Beyond the outcome, the overall impact of the findings is evaluated (see Section 5.3). Other than the outcome section, the impact evaluation discusses unintentional side effects of the measurements and further recommendations made by this thesis on coping with uncertainty.

5.1 Internal Validity Evaluation

This thesis aims at the measurement of transformation-induced uncertainty in particular. This measurement can only be as valid as far as other types of uncertainty are excluded from it. Such other types are, e.g., the requester, the provider, and the matching algorithm [10013]. The internal validity evaluation shows that the measurement of transformation-induced uncertainty, as proposed in this thesis, is not influenced by these other types.

Requester- and provider-induced uncertainties are the symptom of deficient service specifications, i.e., specifications that are incorrect, insufficiently precise, or incomplete. For example, a requester could specify a parameter multiplicity of [0..*], when (always) only one parameter value is handed over to the service.
When such specification is the source model of a transformation to the SSL, information loss may occur in addition to, or because of the deficiency already present. Whenever information loss occurs in a matched element, uncertainty of the matching is induced.

When transformation-induced information loss occurs together with source model deficiency, the measurement process only measures the transformation-induced uncertainty: Recall that the measurement is solely based on the types of lost information, e.g., parameter multiplicities (see Section 4.1.2). The measurement is ignorant to what the lost information expresses. Whether lost information content is incorrect, insufficiently precise, or incomplete, is not considered in the measurement process. Since such deficient information content is the cause of requester- and provider-induced uncertainty, these types of uncertainty are successfully excluded from the measurement.

Unreliable matching algorithms have no influence on the measurements either: The measurement of transformation-induced uncertainty does not take the matching qualities returned by matchers into account. Therefore, the certainty (and correctness) of these qualities has no influence on the measurement. Note that an uncertain result returned by a matcher can still be corrected with certainty, following the approach from Section 4.3.1.

5.2 Outcome Evaluation

The desired outcome of uncertainty measurement is that the measurement result helps requesters in choosing the service that suits their needs best. The outcome evaluation shows that the measurement of uncertainty improves the requester’s service choice when uncertainty is (potentially) induced by the transformation of UML multiplicity elements. This is done by creating an example in which all cases of reasoning about uncertainty are contained (see Section 5.2.1). Each case is then validated using the measurement process introduced in Section 4.3, what obtains the measurement results (see Section 5.2.2). For the total of all cases it is shown that the requester’s service choice is improved, what depicts the measurement outcome (see Section 5.2.3).

Recall that, in an environment without uncertainty, the graph result with the highest quality value is chosen automatically (see Section 2.2). Its quality then defines the quality of the input/output matching. This approach does not suffice in uncertain environments: When the graph result with the highest quality is uncertain, it is no reliably beneficial choice because that quality may be incorrect (see Section 4.2.1). Therefore, this evaluation supposes that the graph result is chosen by the requester.
5.2 Outcome Evaluation

5.2.1 Evaluation Example

Imagine a pair of service and request, specified to evaluate all cases that may occur in the validation of uncertain graph matching results (see Figure 5.1). These specifications do not depict a particular real-world scenario. When the service and request are matched, the best mapping is \((x \mapsto b, y \mapsto c, z \mapsto a)\) because this is the only mapping under which all parameter multiplicities fit.

![Figure 5.1: Evaluation service and request](image)

Recall that there are three cases for information loss that a parameter can suffer from: information loss including SSL assumptions, information loss disjunct to SSL assumptions, or none (see Section 4.3.1). The interfaces from Figure 5.1 are designed to include each of these cases. Since the request has to provide a value for each service input, there is a total of possible six parameter mappings between the service and the request.

Unfortunately, the parameter multiplicities are lost in the transformation of both specifications (see Figure 5.2). Since all parameters have the same type, the default matcher produces only graph results with quality values of 1. When uncertainty is not measured, the requester assumes a perfect match in any of the mappings and chooses arbitrarily. Unfortunately, since only one of six mappings sets the service inputs correctly, the requester only has a chance of \(\frac{1}{6}\) to get the service working. When uncertainty is considered, the occurrence detection (see Section 4.3.1) attaches information loss annotations to the parameters (see Figure 5.2).

![Figure 5.2: Transformation result of evaluation service and request](image)
5. Evaluation and Discussion

5.2.2 Graph Result Validation

Table 5.1 lists the graph results of all possible parameter mappings. These results are obtained from a standard signature matcher that does not consider uncertainty. Recall that graph results consist from a set of parameter matching results. The matching results of underlined parameters are proven to be correct. The parameters of matching results that are proven incorrect are striked out. In which cases the correctness or incorrectness of a matching result is implied is reflected in Section 4.3.1. A parameter matching is uncertain when neither correctness nor incorrectness is implied. Next to each mapping, quality ($Q$) and uncertainty ($U$) values of each graph result are noted. These values are aggregated from the parameter matching results as shown in Sections 2.2 and 4.2.3. At the end of each row, it is noted if the graph result may reasonably be chosen by the requester. Recall that only mapping 3 is working which is colored green here.

<table>
<thead>
<tr>
<th>Graph Result:</th>
<th>$x \mapsto$</th>
<th>$y \mapsto$</th>
<th>$z \mapsto$</th>
<th>$Q$</th>
<th>$U$</th>
<th>Chosen?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>a</td>
<td>b</td>
<td>$\not{c}$</td>
<td>$\frac{2}{3}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2)</td>
<td>b</td>
<td>$\not{a}$</td>
<td>$\not{c}$</td>
<td>$\frac{1}{3}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3)</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>1</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>4)</td>
<td>a</td>
<td>c</td>
<td>$\not{b}$</td>
<td>1</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>5)</td>
<td>c</td>
<td>$\not{a}$</td>
<td>$\not{b}$</td>
<td>$\frac{2}{3}$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6)</td>
<td>c</td>
<td>b</td>
<td>$\not{a}$</td>
<td>1</td>
<td>1</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 5.1: Validation of matching between evaluation service and request

5.2.3 Service Choice

The first thing to mention about Table 5.1 is that the requester still has to make an arbitrary choice: The mappings 3, 4, and 6 have the same quality and uncertainty values. The probability that the requester makes the right choice here is only $\frac{1}{3}$. However, this is already an improvement when compared with the smaller chance of $\frac{1}{6}$ when uncertainty is not taken into account. Note that the excluded mappings are excluded with certainty. Since input mappings require a quality of 1 to make the service work (each input has to be set), the mappings 1, 2, and 5 do not need to be considered at all: The uncertainty inherent to these graph results expresses that the correct result has at most the specified quality (see Section 4.2.4), which is only $\frac{2}{3}$ or $\frac{1}{3}$, both < 1.

48
The uncertainty inherent to the graph result 3, 4, and 6 informs the requester that the results may not be correct. When considering other provided services, the requester will experience that no service on the market can be matched reliably: The uncertainty is codetermined by the information loss inherent to the request. Possibly, the requester reconsiders his specification and sends another request to the market that has no information loss. He may find a fitting service for that request.

Last but not least it has to be noted that the parameter matchings on the $z$ parameter either certainly match or certainly not match. $z$ defines the case of a requested parameter not suffering from information loss. Recall that requested input parameters have a provided role in parameter matching because they provide the values required by a service to work (see Section 4.3.3). It becomes apparent that for parameters with a provided role not suffering from information loss, a certain matching result can always be produced, even if the parameter with the required role suffers from information loss.

5.3 Impact Evaluation and Discussion

The findings of this thesis propose ways of coping with transformation-induced uncertainty beyond uncertainty measurement. Such overall impact is discussed in this section. It includes recommendations for specification authors, i.e., requesters and providers, as well as the proposal of extending the SSL by parameter multiplicities.

5.3.1 Recommendations on Service Specification

Recall that matchings on parameters with provided role not suffering from information loss do not lead to uncertainty, when information loss is considered in matchings (see Section 5.2.3; the $z$ parameter in Table 5.1). This proposes that especially for those parameters, i.e., requested inputs and provided outputs, specification authors should prevent information loss in their transformations. To the contrary, it is okay for parameters with requested role, i.e., requested outputs and provided inputs, when information is lost, as long as this information includes the SSL assumptions made on it: In that case, a match can be detected with certainty, when the parameter with provided role does not suffer from information loss (see the $b$ parameter in Table 5.1).

5.3.2 SSL Multiplicity Extension

Also, an extension to the SSL can be proposed that includes parameter multiplicities: Multiplicities are a basic concept for expressing typed parameter collec-
tions, what is needed for many applications. Moreover, specifying and matching multiplicities is more efficient than uncertainty measurement: Multiplicities just require two integer values for lower and upper bounds. Thereby, service descriptions specifying multiplicities are smaller and more compact than specifications with information loss annotations. Matching multiplicities can be done efficiently by just comparing the lower and upper bounds of the matched parameters. Such a comparison is even faster than reasoning about information loss annotations.

Still, the concept of information loss annotations and match validation based on these is a promising approach for the measurement of transformation-induced uncertainty in general. When other information loss causes are identified for which an integration in the SSL is infeasible, the measurement process introduced in this thesis promises to be a valuable tool for the measurement of uncertainty induced by these causes.
6 Related Work

Uncertainty in service matching is an issue not yet investigated broadly: A measurement scenario similar to the scenario investigated in this thesis is only found in one publication presented in Section 6.1. However, there is additional research not as similar, but also related. For instance, the field of automatic transformation generation (see Section 6.3) may impose future work issues.

6.1 Vijapurwala: Specification Incompleteness

The Master's thesis “Handling Incomplete Service Specifications using Fuzzy Matching” by Vijapurwala [Vij14] relates to this thesis by approaching the measurement of uncertainties in service matching in a similar measurement scenario. Note that Vijapurwala’s terminology identifies such uncertainties as matching “fuzziness”, just like, e.g., in [Pla13].

Measurement Scenarios

Vijapurwala concentrates on provider-induced uncertainty resulting from specification incompleteness. Since transformations lose information, the measurement scenarios of Vijapurwala’s and this thesis are similar in that transformation-induced uncertainty can also be seen as result of specification incompleteness: A transformations target model can be incomplete in terms of not including all information included in the source model.

The difference between the measurement scenarios is that transformation-induced incompleteness is measured by identifying information that is solely available in a source model, because the target language is not able to express that kind of information (see Section 4.1.2). To the contrary, provider-induced uncertainty is measured by identifying information that is not included in the target model, even though the target language could express it. Such information is identified by checking, if the target model has all the properties defined in its meta-model set. If some attribute provided by the meta-model is not set in the target model, the specification lacks the information expressable with that attribute, i.e., the specification is incomplete.
6. Related Work

Measurement Representation

Since both provider-induced uncertainty and transformation-induced uncertainty deal with specification incompleteness, a common measurement representation is desirable. Vijapurwala’s thesis proposes fuzziness scores (uncertainty scores) ranging from 0 to 100. For these scores, no precise interpretation is defined.

This Bachelor’s thesis chose the uncertainty about the membership to the set of correct matching results as representation for uncertainty (see Section 4.2.2). This representation clearly defines the interpretation of uncertainty values. Using such definition ensures that uncertainty values can always be interpreted correctly by market participants. Moreover, it enables to validate the correctness of uncertainty measurements.

Both representations can be combined by mapping either fuzziness scores to uncertainty values or the other way round. When the measurements are provided in a common representation, they can be aggregated and are comparable to each other. However, since the interpretation of fuzziness scores is undefined, it is not clear, how the representations can be translated into each other.

6.2 Transformation of Uncertain Models

Typically, model transformation techniques rely on the assumption that models do not contain uncertainty. When, to the contrary, uncertainty is present, the transformation may not be executable. The transformation then has to be postponed until more information is available or the uncertainty has to be resolved artificially by including (potentially wrong) assumed information.

The paper “Transformation of Models Containing Uncertainty” by Famelis et al [FSDSC13] addresses this problem by proposing a technique allowing for uncertainty in a transformation’s source model. The approach lifts an existing transformation, i.e., it automatically adapts a transformation to accept a May model as additional input. May models make the uncertainty of a source model explicit by expressing, whether the presence of referenced source model elements is known or unknown. Thus, a May model defines a set of concretizations, i.e., potential concrete models without uncertainty. For each concretization, the lifted transformation returns a result.

The work of Famelis et al shows, how transformations to the SSL can consider provider- and requester-induced uncertainties explicited in May models. When such uncertainties are present, it may not suffice to just execute the transformation and potentially aggregate measurements on the different uncertainty types in advance. To the contrary, the transformation may have to be lifted to produce sound results. Thus, Famelis et al’s paper relates to this thesis by proposing a combined measurement of provider-, requester-, and transformation-induced uncertainty, what can be done in future work.
6.3 Automatic Transformation Generation

Model-Driven (Software) Engineering recommends the extensive use of models defined by their meta-models. Inescapably, similar meta-models with related or even the same objectives are created. We observe this issue in the similarity of UML and SSL interfaces that has lead to the idea of defining a UML-to-SSL transformation in the scope of this thesis.

Since ad-hoc transformations are developed in reoccuring patterns and their manual development results in a high work-load, automatic and semi-automatic transformation generation is a subject of current research \cite{LHDSB06, FHLN08, DFV07, VIR10}. It promises to accelerate development and to increase transformational code quality.

The common approach for transformation generation is to perform a meta-model matching. Such matching returns a weaving model that links semantically corresponding elements from the matched meta-models to each other. Subsequently, that model is used to generate the transformational code.

Meta-model matching does not only relate to service matching in general by using similar approaches for a similar problem. Also, it relates to this thesis in particular by introducing the question, if the identification of information loss causes can be automated, as well. In that case, the transformation-induced uncertainty inherent to a matching could be measured with less manual interference, no matter in which language the matched services are specified. Therefore, automatic uncertainty measurement is a promising future work issue.
7 Conclusions

This chapter concludes this thesis by pointing out its main contributions in Section 7.1. Section 7.2 outlines future work issues.

7.1 Contributions

The UML-to-SSL signature transformation developed in Chapter 3 enables the automated translation of UML signature specifications to the SSL. With the integration of this transformation into the Service Specification Environment SSE (see Section 4.4.1.2), UML signature specifications do not have to be translated manually, any more. For the signature transformation, parameter multiplicities are identified as cause for information loss (see Section 5.3.2). It is discussed to include parameter multiplicities in the SSL to prevent information loss for the operation signature aspect (see Section 5.3.2).

It is shown that the precise probability of an actual match between operation signatures is codetermined by execution probabilities (see Section 4.1.1). Since these probabilities can not be expressed in the SSL, it is infeasible to obtain matching probabilities as uncertainty measurement values. Instead, a fuzzy set of correct matching results defines the measurement interpretation (see Section 4.2.1). The uncertainty about the membership to that set is used as measurement representation applicable for other uncertainty types, as well. For parameter multiplicities, only a dichotomic measurement expressing if uncertainty occurred or not is reasonable (see Section 4.2.4).

A measurement process applicable for uncertainty induced by arbitrary transformations is proposed in Section 4.3. It validates the results returned by a standard matcher by measuring the uncertainty of these results. The obtained uncertainty values help requesters in assessing the risk of matching result incorrectness. Moreover, specific matching results can even be corrected with certainty, improving the requester’s service choice even further. The integration of a validated signature matcher in the SSE (see Section 4.4.1.3) realizes the service choice improvements.

Finally, this thesis identifies several issues for future work on the measurement of transformation-induced uncertainty. These issues are outlined in Section 7.2.
7. Conclusions

7.2 Future Work

Since the scope of a Bachelor's thesis is limited, not every aspect of measuring transformation-induced uncertainty can be approached. Further research on that topic promises to improve the service choice of requesters in uncertain environments even more. Some open issues are identified that are presented here for future work.

7.2.1 Service Choice Strategies in Uncertain Environments

In an environment without uncertainty, the best graph result is the result with the highest quality value. However, when uncertainty may occur, this is not always the case: Results with high quality that suffer from uncertainties do not depict a reliably beneficial choice. Which of the possible parameter mappings are a promising choice, depends on both the quality and the uncertainty values of the corresponding graph results. Often, a non-trivial decision between certain and (supposedly) qualitative matching results has to be made. To make that decision, requesters need promising service choice strategies that are based on their individual risk affinity.

The decision problem between certainty and match quality also induces that the quality values of input and output matching can not simply be the highest contained graph result value. A reasonable aggregation for these qualities has to be based on which mapping is chosen. As already noted, this is not always the mapping with the highest match quality.

7.2.2 Information Loss Causes in other Service Aspects

This thesis only regarded the operation signature aspect of service specification and matching. As a result, the only information loss cause identified is parameter multiplicities (see Section 3.2). For other service aspects, additional information loss causes can be expected in a UML-to-SSL transformation because the UML offers expressive specification infrastructure for multiple service aspects: Protocol statecharts \[\text{Obj11b}, \text{pp. 523 }- \text{584}\] can be used to define interaction protocols for service interfaces. The Object Constraint Language OCL \[\text{Obj14}\] is highly expressive for operation pre- and postconditions. The identification of additional information loss causes and the validation of the other aspect’s matching results is required to make the UML reliably usable on the OTF market.
7.2.3 Dependency Analysis in the Measurement Process

When a model element depends on model elements that suffer from information loss, it may be affected by their information loss: Imagine a software service that has required interfaces. When one of these interfaces suffers from information loss, it can not be guaranteed to be matched correctly. Consequently, the correct functioning of the service can not be guaranteed a priory. The operations that are provided by the service may fail because of the information loss that is inherent to other model elements, the required interfaces.

When these and other dependencies between model elements are detected, the uncertainty measurement can be improved so it returns warnings when elements are matched that depend on elements suffering from information loss. The detection of dependencies can be done in a fourth measurement process step, the dependency analysis. The results of such analysis can be combined with the results from the occurrence detection step, forming a more comprehensive uncertainty report. Figure 7.1 shows, how the dependency analysis is integrated in the uncertainty measurement process.

A full and reliable dependency analysis has to go much deeper than merely analysing provided/required roles: A service can specify to require an interface without using it in all of its provided operations. When an operation is matched that does not actually require the other interface, then the uncertainty measurement should not consider that interface. To analyse dependencies between model elements reliably, a full behavior analysis of the source model is needed. Behavior specifications contain information on how and when which individual model elements are relevant to other elements. However, even a full behavior analysis is only an estimate for dependencies at run-time: The relevance of dependencies still depends on execution probabilities.

Since a reliable dependency analysis is so hard to perform, it is not performed in the scope of this thesis. Note that for example, UML behavior statecharts [Obj11b, p.535] provide much of the infrastructure needed to perform a full dependency analysis.
7. Conclusions

7.2.4 Specification Check Use Case

Besides service matching, there is another use case for detecting information loss: Authors of specifications may want to check their original model before they perform a transformation. Knowing which model elements suffer from information loss when transformed enables authors of specifications to improve their models early. Improvements can be achieved by excluding or replacing the source model elements that are to suffer from information loss. Thus, the certainty of matchings considering the model is increased.

This alternative use case requires the information loss annotations from the occurrence detection step to be attached to the source model (because the source model is checked and altered). With source model annotations, all information required by the use case is provided. Note that source model annotations can be transformed along with the model to obtain target model annotations. Such annotations are needed by the subsequent measurement process steps. A transformation of annotations can easily be done by resolving the referenced model elements. This way, the measurement process can combine the two use cases for information loss detection, the uncertainty measurement use case and the specification check use case.
Appendix A

Unconsidered Meta-Class Properties

Not every property that belongs to SSL and UML meta-model classes is relevant to the specification and transformation of operation signatures. The relevant properties are already regarded in Sections 2.4 and 2.5. The irrelevant properties as well as the reasons for their irrelevance are presented in this appendix.

A.1 Unconsidered SSL Element Properties

The SSL introduces several element properties that are not considered in the transformation. For some, the reason why they are unconsidered is that they exceed the scope for this thesis. Others can be derived, which is why they do not need to be set. And some are inherited from the PCM, but not used in the SSL. This section completes the view on the SSL (that is provided in Section 2.4) by introducing the unconsidered class properties.

A.1.1 Repository

In addition to the properties presented in Section 2.4.1, the Repository has these unconsidered properties:

- components : RepositoryComponent [*]
  The components that are stored in the repository.

- characterisationDefinitions : CharacterisationDefinition [*]
  In the PCM, CharacterisationDefinitions [RBB+11, p. 115] are used to enable performance predictions. The SSL does not focus on this aspect.

- repositoryDescription : String [0..1]
  An optional description of the repository in natural language. The semantics of this description are not defined.
A. Unconsidered Meta-Class Properties

A.1.2 OperationInterface

In addition to the properties presented in Section 2.4.2, the **OperationInterface** has these unconsidered properties:

- **protocols : Protocol [•]**
  The interaction protocols for this interface. The transformation of protocols is not approached in this thesis.

- **ancestorInterfaces : Interface [•]**
  Interfaces that are extended by this interface (directly or indirectly via intermediate interfaces). This property can be derived with the **parentInterface** property.

- **repository : Repository**
  The repository this interface is a part of (see Section 2.4.1). This property can be derived from the opposite association.

A.1.3 ExtendedOperationSignature

In addition to the properties presented in Section 2.4.3, the **ExtendedOperationSignature** has these unconsidered properties:

- **returnType : DataType [0..1] (from OperationSignature)**
  The return type of the operation. In the SSL, this property is not used. Instead, output parameters are specified.

- **exceptions : ExceptionType [•] (from OperationSignature)**
  PCM ExceptionTypes \[RBB +11, p. 120\] are not used in the SSL. Instead, failure types can model exceptions in the execution of an operation (see Section 2.4.6).

- **interface : Interface [•] (from OperationSignature)**
  The interface this operation belongs to (see Section 2.4.2). This property can be derived from the opposite association.

- **preconditions : EString [•]**
  An operation can require a set of preconditions to hold. The transformation of operation conditions is not proposed for this thesis.

- **postconditions : EString [•]**
  An operation can assure a set of postconditions to hold after a call. The transformation of operation conditions is not proposed for this thesis.

A.1.4 ExtendedParameter

In addition to the properties presented in Section 2.4.4, the **ExtendedParameter** has these unconsidered properties:
A.2 Unconsidered UML Element Properties

- **operationSignature**: OperationSignature [0..1] (from Parameter)
  The operation signature this parameter belongs to (see Section 2.4.3). This property can be derived from the opposite association.

- **eventType**: EventType [0..1] (from Parameter)
  Related to deployment, what is not considered in this thesis. This property is not documented. It was created the 10th of June 2010 [Kar].

- **infrastructureSignature**: InfrastructureSignature [0..1] (from Parameter)
  Related to deployment and not documented, as well. This property was created the 25th of May 2010 [Kar].

- **resourceSignature**: ResourceSignature [*] (from Parameter)
  Related to deployment and not documented, as well. This property was created the 25th of May 2010 [Kar].

A.1.5 SimpleOntologyDataType

In addition to the properties presented in Section 2.4.5, the SimpleOntologyDataType has these unconsidered properties:

- **characterisationDefinitions**: CharacterisationDefinition [*] (from DataType)
  In the PCM, CharacterisationDefinitions [RBB+11, p. 115] are used to enable performance predictions. Since the SSL does not focus on this aspect, characterizations are not regarded.

- **repository**: Repository (from DataType)
  The repository this type is a part of (see Section 2.4.1). This property can be derived from the opposite association.

A.2 Unconsidered UML Element Properties

The UML introduces several element properties that are not considered in the transformation. This section completes the view on the UML (that is provided in Section 2.5) by introducing the unconsidered class properties.

A.2.1 NamedElement

In addition to the properties presented in Section 2.5.1, the NamedElement has these unconsidered properties:

- **visibility**: VisibilityKind [0..1]
  Visibility kinds define the visibility of an element outside the Namespace that owns it [Obj11b, pp. 141f.]. Possible values of this property are public (visible to all elements that access the namespace), private (only visible inside namespace), protected (only visible to specializations), and package (visible in the nearest enclosing package).
A. Unconsidered Meta-Class Properties

- **clientDependency** : Dependency [*]
  
  Every named element can depend on other elements what can be specified using the Dependency class [Obj11b, pp. 61f.]. Dependency associations do not have any runtime semantics.

Service requesters stand outside the UML Model (see Section 2.5.3). Therefore, only those elements that have a public visibility can be offered to them. As a consequence, this thesis will only consider publicly visible elements.

A.2.2 Namespace

Not all, but many of the classes that are presented in this chapter inherit from Namespace [Obj11b, pp. 100ff.]. Inside a namespace, all elements are required to be distinguishable by their names. All Namespace properties are unconsidered in this thesis. In addition to the properties inherited from NamedElement (see Section 2.5.1), Namespaces have the following properties:

- **elementImport** : ElementImport [*]
  
  An element import allows to refer to an element from another namespace without its qualified name [Obj11b, pp. 64ff.]. This property does not define any semantics.

- **packageImport** : PackageImport [*]
  
  A package import is similar to an element import, but with the difference that it addresses all members of the imported package [Obj11b, pp. 112f.].

- **ownedRule** : Contraint [*]
  
  Contraints are restrictions on the model and may be expressed in a natural or machine-readable language, e.g., OCL [Obj11b, pp. 57ff.]. Contraints are not considered in the scope of this thesis.

A.2.3 Package

In addition to the properties presented in Section 2.5.2, the Package has these unconsidered properties:

- **URI** : String [0..1]
  
  Provides a unique identifier of a package, but does not define any semantics. Therefore, it is not considered in the scope of this thesis.

- **packageMerge** : Package [*]
  
  A package merge is similar to a package import (see Section A.2.2), but with the difference that elements representing the same concept are combined [Obj11b, p. 113]. There is no difference between a package with explicit merge definitions and a package on which all these merges are performed. This thesis will assume, that the UML model only contains packages on which all merges are performed.
A.2.4 Model

The UML Model (see Section 2.5.3) introduces one unconsidered property:

- viewpoint : String [0..1]
  The viewpoint name may refer to a UML profile definition, but does not have to. UML profiles are not considered by this thesis. Viewpoints that do not reference a profile do not define any semantics.

A.2.5 Interface

In addition to the properties presented in Section 2.5.4, the Interface has these unconsidered properties:

- isAbstract: Boolean (from Classifier)
  True, if the classifier can not be instantiated. Interfaces are always abstract.

- isFinalSpecialization: Boolean (from Classifier)
  True, if the classifier can not be specialized. Since all interfaces are abstract, they are never final.

- generalization : Generalization [*] (from Classifier)
  Reference to more general classifiers. For interfaces, this is already given with the redefinedInterface association.

- substitution : Substitution [*] (from Classifier)
  Reference to classifiers that may be substituted by this classifier. Substitution associations [Obj11, pp. 136f.] are used for domains that do not support specialization. The software-service domain usually does support specialization.

- powertypeExtent : GeneralizationSet [*] (from Classifier)
  For these collections, this classifier is a supertype. Having the generalization property, this property does not define additional semantics.

- isLeaf : Boolean (from RedefinableElement)
  Indicates, whether this element can furtherly be refined. Refinement constraints of that kind do not have to be regarded in service matching.

A.2.6 Operation

In addition to the properties presented in Section 2.5.5, the Operation has these unconsidered properties:

- isQuery: Boolean
  If true, this operation leaves the state of the system unchanged. Query operations can also be modeled by not specifying any postconditions. This property is not considered, because pre- and postconditions are not in the scope of this thesis.
A. Unconsidered Meta-Class Properties

- **precondition**: Constraint [\*]
  A set of constraints on the system state that must hold to invoke this operation. Operation conditions are not regarded by this thesis.

- **postcondition**: Constraint [\*]
  A set of constraints that are assured to hold after termination of this operation. Operation conditions are not regarded by this thesis.

- **bodycondition**: Constraint [0..1]
  An optional constraint on the result of this operation. Operation conditions are not regarded by this thesis.

- **class**: Class [0..1]
  The class this operation may belong to. Since we only consider interfaces, this association is never set.

- **isStatic**: Boolean (from Feature)
  If true, this operation references the classifier owning this operation. If false, it references individual instances of the classifier. If an operation references a classifier or an instance is basically just a question of where data is stored. Therefore, this flag is not considered.

- **redefinedOperation**: Operation [\*]
  References redefined operations. An operation that redefines another operation is substitutable for that other operation. Service matching checks that substitutability by a semantics analysis. Therefore, there is no need to consider this property.

A.2.7 Property

In addition to the properties presented in Section 2.5.8, the Property has these unconsidered attributes:

- **association**: Association [0..1]
  If this Property represents an association, that association is referenced here. In this case, many of the other properties of this Property are derived from the association. Instances of the Association class are not considered in this thesis.

- **associationEnd**: Property [0..1]
  In a bidirectional association, this association represents the opposite association end which owns a qualifier Property (if any).

- **qualifier**: Property [\*]
  This is a subset of the attributes of the classifier that owns this Property. This subset is used by the opposite association end to qualify the classifier that owns this Property. This is only a structural information.

- **aggregation**: AggregationKind
  An association can take the aggregation kinds *none* (for no aggregation),
shared (what is a semantic variation point), and composite (the property owner manages the property values) \[O\text{b}i111\], pp. 35f.]. This is only a structural information.

- **isID**: Boolean
  
  Specifies, whether this Property can be used to identify the classifier that owns it. This is only a structural information.

- **redefinedProperty**: Property [*]
  
  Properties can redefine each other. This association references the properties that are redefined by this property.

- **subsettedProperty**: Property [*]
  
  Properties can subset each other. This association references the properties that are subsetted by this property.

- **isDerived**: Boolean
  
  Specifies, whether Property values can be computed from other information. This is only a structural information.

- **isDerivedUnion**: Boolean
  
  Specifies, whether Property values are given by the union of subsetting properties. This is only a structural information.

- **interface**: Interface [0..1]
  
  The Interface (see Section 2.5.4) this Property belongs to. This association can be derived from the opposite association.

- **class**: Class [0..1]
  
  The class this Property may belong to. Since we only consider interfaces, this association is never set.

- **datatype**: DataType, [0..1]
  
  The data type this Property may belong to. Since we only consider interfaces, this association is never set.

- **isStatic**: Boolean (from Feature)
  
  If true, this operation references the classifier owning this operation. If false, it references individual instances of the classifier. If an operation references a classifier or an instance is basically just a question of where data is stored. Therefore, this flag is not considered.

- **isLeaf**: Boolean (from RedefinableElement)
  
  Indicates, whether this element can furtherly be refined. Definition constraints of that kind do not have to be regarded in service matching.
Appendix B

Potential Transformation Source Languages

Generally, every specification language usable for service description is a candidate for a transformation to the SSL. Three requirements for candidate languages promise to bring valuable research findings:

**R1** For the occurrence of transformation-induced uncertainty, a source language has to be more expressive than the target language SSL. That means, it should contain concepts, that can not directly be mapped to SSL concepts.

**R2** On the other hand, the occurring information loss has to be moderate: If the bigger part of a model’s semantics is lost in a transformation, the transformation result is not expedient for matching.

**R3** Moreover, languages with high practical relevance in industry or academia are preferable: Supporting practically relevant languages promises to increase the practical relevance of the SSL and SSL matchers.

In Table B.1, candidate languages are coarsely checked for their fulfillment of R1 – R3. R1 and R2 are combined, since both express requirements on the expressive power of a language. Note that a sound check on these requirements needs more in-depth research, what can not be done here. Mere pre-/postcondition languages were not considered, since their transformation may require ontologies to be modified, what is infeasible for most market participants. For this thesis, UML signature specifications were chosen based on this coarse check.
### B. Potential Transformation Source Languages

<table>
<thead>
<tr>
<th>Specification Language</th>
<th>Included Aspect(s)</th>
<th>Expectations</th>
</tr>
</thead>
</table>
| **(SA-)WSDL**          | Signatures        | R1/2 As ontology-annotated signature specification, SA-WSDL is conceptually close to the SSL: A transformation is expected to preserve all semantics.  
                        |                   | R3 Standardized, widely applied for signature specification. |
| Java                   | Signatures        | R1/2 No information loss to expect.  
                        |                   | R3 In Java web service implementation, EJB web service specifications are typically translated to WSDL using JAX-WS. That may further motivate WSDL investigation. |
| **UML**                | Signatures        | R1/2 Multiplicities are not part of the SSL  
                        |                   | R3 Common for modelling software in general. The transformation of UML signatures is a prerequisite for other UML transformations. |
| **OWL-S**              | Signatures, Pre-/Postconditions, Protocols, QoS | R1/2 Plugs other languages in to describe a service profile, -model, and -grounding. Therefore, a transformation is not directly applicable.  
                        |                   | R3 To be standardized. Used for a highly comprehensive service specification. |
| **UML (Protocol) Statecharts** | Protocols, (Pre-/Postconditions) | R1/2 Transition conditions promise to yield moderate uncertainty. However, they are usually specified in OCL.  
                        |                   | R3 Common for modelling software in general. The use of statecharts to model protocols for a comprehensive service description language is proposed in [HGEJ12]. |
| **WS-BPEL**            | Protocols, (Pre-/Postconditions) | R1/2 Variables, exceptions and delayed executions carry the risk of gaining transformation results not expedient for further processing  
                        |                   | R3 Intended for service composition specification. |
| **CSP**                | Protocols, (Pre-/Postconditions) | R1/2 Timeouts and interrupts may exceed moderate uncertainty occurrence.  
                        |                   | R3 Research relevance since decades. Mainly applied to verify process properties. |
| **Petri-Nets**         | Protocols, (Pre-/Postconditions) | R1/2 The many variations of petri-nets require a more precise definition of which petri-nets to investigate.  
                        |                   | R3 Well-known. For protocols usually used for correctness verification. The choice of variation comes close to defining a new language. Thus, no relevance gain is promised. |

Table B.1: Expectations on the fulfillment of language requirements
Bibliography


[HGEJ12] Zille Huma, Christian Gerth, Gregor Engels, and Oliver Juwig. A
Bibliography


volltexte/1000022503.


